

Lane A. Hemaspaandra  
Mitsunori Ogiwara

# The Complexity Theory Companion



Springer

Lane A. Hemaspaandra • Mitsunori Ogihara

# The Complexity Theory Companion

With 43 Figures



Springer



#### Authors

Prof. Dr. Lane A. Hemaspaandra  
Prof. Dr. Mitsunori Ogihara  
Department of Computer Science  
Rochester, NY 14627  
USA  
{lane,ogihara}@cs.rochester.edu

#### Series Editors

Prof. Dr. Wilfried Brauer  
Institut für Informatik  
Technische Universität München  
Arcisstrasse 21, 80333 München, Germany  
brauer@informatik.tu-muenchen.de

Prof. Dr. Grzegorz Rozenberg  
Leiden Institute of Advanced Computer Science  
University of Leiden  
Niels Bohrweg 1, 2333 CA Leiden, The Netherlands  
rozenber@liacs.nl

Prof. Dr. Arto Salomaa  
Data City  
Turku Centre for Computer Science  
20 500 Turku, Finland  
asalomaa@utu.fi

QA  
267.7  
.H46  
2002

Library of Congress Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek – CIP-Einheitsaufnahme

Hemaspaandra, Lane A.:  
The complexity theory companion/Lane A. Hemaspaandra; Mitsunori Ogihara. –  
Berlin; Heidelberg; New York; Barcelona; Hong Kong; London; Milan;  
Paris; Tokyo: Springer, 2002  
(Texts in theoretical computer science)  
ISBN 3-540-67419-5

ACM Computing Classification (1998): F.1, F.2.2, F.4

ISBN 3-540-67419-5 Springer-Verlag Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilm or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable for prosecution under the German Copyright Law.

Springer-Verlag Berlin Heidelberg New York,  
a member of BertelsmannSpringer Science+Business Media GmbH

© Springer-Verlag Berlin Heidelberg 2002  
Printed in Germany

The use of general descriptive names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

Cover Design: KünkelLopka, Heidelberg  
Typesetting: Camera-ready by the authors  
Printed on acid-free paper SPIN 10723529 45/3142SR – 5 4 3 2 1 0

*This book is dedicated to our families—the best of companions.*



# Preface

## Invitation

**Secret 1** *Algorithms are at the heart of complexity theory.*

That is the dark secret of complexity theory. It is recognized by complexity theorists, but would be literally incredible to most others. In this book, we hope to make this secret credible. In fact, the real secret is even more dramatic.

**Secret 2** *Simple algorithms are at the heart of complexity theory.*

A corollary of Secret 2 is that every practitioner of computer science or student of computer science already possesses the ability required to understand, enjoy, and employ complexity theory.

We realize that these secrets fly in the face of conventional wisdom. Most people view complexity theory as an arcane realm populated by pointy-hatted (if not indeed pointy-headed) sorcerers stirring cauldrons of recursion theory with wands of combinatorics, while chanting incantations involving complexity classes whose very names contain hundreds of characters and sear the tongues of mere mortals. This stereotype has sprung up in part due to the small amount of esoteric research that fits this bill, but the stereotype is more strongly attributable to the failure of complexity theorists to communicate in expository forums the central role that algorithms play in complexity theory.

Throughout this book—from the tree-pruning and interval-pruning algorithms that shape the first chapter to the query simulation procedures that dominate the last chapter—we will see that proofs in complexity theory usually employ algorithms as their central tools. In fact, to more clearly highlight the role of algorithmic techniques in complexity theory, *this book is organized by technique rather than by topic*. That is, in contrast to the organization of other books on complexity theory, each chapter of this book focuses on one technique—what it is, and what results and applications it has yielded.

The most thrilling times in complexity theory are when a new technique is introduced and sweeps like fire over the field. In addition to highlighting the centrality of algorithms in the proof arsenal of complexity theory, we feel that our technique-based approach more vividly conveys to the reader the flavor and excitement of such conflagrations. We invite the reader to come

with us as we present nine techniques, usually simple and algorithmic, that burned away some of the field's ignorance and helped form the landscape of modern complexity theory.

## Usage

We intend this book as a companion for students and professionals who seek an accessible, algorithmically oriented, research-centered, up-to-date guide to some of the most interesting techniques of complexity theory. The authors and their colleague Joel Seiferas have test-driven the book's approach in two different courses at the University of Rochester. We have used this technique-based approach in Rochester's one-semester basic complexity theory course, which is taken by all first-year computer science graduate students and also by those undergraduates specializing or specially interested in theoretical computer science, and in our second course on complexity theory, which is taken by all second-year graduate students as their theory "breadth" course.

We found in both these course settings that the technique-based approach allowed us to impart to students a significant amount of the feel and experience of complexity theory research and led to more student interest and involvement than occurred in earlier course incarnations using other texts. We expect that this will not only benefit the complexity theory students in the courses, but will also help all the course's students become prepared to do work that is theoretically aware, informed, and well-grounded.

At times, we stop the flow of a proof or discussion with a "Pause to Ponder." These are places at which we encourage the reader to pause for a moment and find his or her own solution to the issue raised. Even an unsuccessful attempt to craft a solution will usually make the proof/discussion that follows clearer and more valuable, as the reader will better understand the challenges posed by the hurdle that the proof/discussion overcomes.

With some exceptions due to result dependencies, the non-appendix chapters are generally ordered to put the easier chapters near the start of the book and the more demanding chapters near the end of the book.

## Acknowledgments

We are extraordinarily indebted to the following people, who proofread one or more chapters, for their invaluable help, suggestions, corrections, and insights: Eric Allender, Russell Bent, Alina Beygelzimer, Matthew Boutell, Samuel Chen, Yin-He Cheng, Louis Deaett, Gregory Goldstein, Fred Green, Ulrich Hertrampf, Chris Homan, Gabriel Istrate, Jason Ku, David Lagakos, Andrew Learn, Tao Li, Ioan Macarie, Proshanto Mukherji, Kenneth Regan, William Scherer III, Alan Selman, D. Sivakumar, Howard Straubing, Robert Swier,



Mayur Thakur, Jonathan Tomer, Jacobo Torán, Leen Torenvliet, Dieter van Melkebeek, Heribert Vollmer, Julie Zhong, and Marius Zimand. We also thank the many other people who have helped us with advice, comments, corrections, literature pointers, most-recent-version information, and suggestions: Andris Ambainis, Vikraman Arvind, Richard Beigel, Nate Blaylock, Daniel Bovet, Jin-Yi Cai, Diane Cass, Stephen Fenner, Lance Fortnow, William Gasarch, Viliam Geffert, Oded Goldreich, Juris Hartmanis, Edith Hemaspaandra, Paul Ilardi, Sven Kosub, Richard Lipton, Alexis Maciel, Wolfgang Merkle, Christos Papadimitriou, Thanos Papathanasiou, Eduardo Pinheiro, Robert Rettinger, Jörg Rothe, Alex Samorodnitsky, Marcus Schaefer, Michael Shear, Uwe Schöning, Joel Seiferas, Samik Sengupta, Carl Smith, Scott Stoness, Madhu Sudan, Chunqiang Tang, Jun Tarui, Thomas Thierauf, Luca Trevisan, Chris Umans, Osamu Watanabe, Chee Yap, and Stathis Zachos. Any remaining errors are the responsibility of the authors.

We thank our thesis advisors, Juris Hartmanis and Kojiro Kobayashi; their insightful, dedicated, joyous approach to research has been a continuing inspiration to us.

We appreciate the grants—NSF-CCR-8957604, NSF-INT-9116781/JSPS-ENGR-207, NSF-CCR-9322513, NSF-INT-9513368/DAAD-315-PROfo-ab, NSF-CCR-9701911, NSF-CCR-9725021, NSF-INT-9726724, NSF-INT-9815095/DAAD-315-PPP-gü-ab NSF-DUE-9980943, DARPA-F30602-98-2-0133, NIA-R01-AG18231—that have supported our research programs during the planning and writing of this book.

For generous advice, help, and support, we thank the Springer-Verlag series editors and staff, namely, Wilfried Brauer, Grzegorz Rozenberg, Arto Salomaa, Alfred Hofmann, Frank Holzwarth, Ingeborg Mayer, Sherryl Sundell, and Hans Wössner. Our own department's technical and secretarial staff—especially Jill Forster, Elaine Heberle, and Jim Roche—was invaluable in keeping our computers up and our manuscript copied and circulating, and we much appreciate their help.

We are grateful to those colleagues—Peter van Emde Boas, Harald Hempel, Jörg Rothe, Alan Selman, Seinosuke Toda, Leen Torenvliet, Heribert Vollmer, Klaus Wagner, Osamu Watanabe, and Gerd Wechsung—who have generously hosted our research visits during the planning and writing of this book, and to the many colleagues, as cited in the Bibliographic Notes sections, who have collaborated with us on the research described in some sections of this book.

Above all, we thank Edith, Ellen, Emi, and Erica for their advice, love, and support.

*Lane A. Hemaspaandra*

*Mitsunori Ogihara*

Rochester, NY  
October 2001



# Contents

<b>Preface</b> .....	vii
Invitation .....	vii
Usage .....	viii
<b>1. The Self-Reducibility Technique</b> .....	1
1.1 GEM: There Are No Sparse NP-Complete Sets Unless $P=NP$ ..	2
1.2 The Turing Case .....	18
1.3 The Case of Merely Putting Sparse Sets in NP – P: The Hartmanis–Immerman–Sewelson Encoding .....	22
1.4 OPEN ISSUE: Does the Disjunctive Case Hold? .....	26
1.5 Bibliographic Notes .....	26
<b>2. The One-Way Function Technique</b> .....	31
2.1 GEM: Characterizing the Existence of One-Way Functions ..	32
2.2 Unambiguous One-Way Functions Exist If and Only If Bounded-Ambiguity One-Way Functions Exist .....	35
2.3 Strong, Total, Commutative, Associative One-Way Functions Exist If and Only If One-Way Functions Exist ....	36
2.4 OPEN ISSUE: Low-Ambiguity, Commutative, Associative One-Way Functions? .....	42
2.5 Bibliographic Notes .....	43
<b>3. The Tournament Divide and Conquer Technique</b> .....	45
3.1 GEM: The Semi-feasible Sets Have Small Circuits .....	45
3.2 Optimal Advice for the Semi-feasible Sets .....	48
3.3 Unique Solutions Collapse the Polynomial Hierarchy .....	56
3.4 OPEN ISSUE: Are the Semi-feasible Sets in $P/\text{linear}$ ? .....	63
3.5 Bibliographic Notes .....	63
<b>4. The Isolation Technique</b> .....	67
4.1 GEM: Isolating a Unique Solution .....	68
4.2 Toda’s Theorem: $PH \subseteq P^{PP}$ .....	72
4.3 $NL/\text{poly} = UL/\text{poly}$ .....	82

4.4	OPEN ISSUE: Do Ambiguous and Unambiguous Nondeterminism Coincide? . . . . .	87
4.5	Bibliographic Notes . . . . .	87
5.	<b>The Witness Reduction Technique . . . . .</b>	91
5.1	Framing the Question: Is $\#P$ Closed Under Proper Subtraction? . . . . .	91
5.2	GEM: A Complexity Theory for Feasible Closure Properties of $\#P$ . . . . .	93
5.3	Intermediate Potential Closure Properties . . . . .	99
5.4	A Complexity Theory for Feasible Closure Properties of $\text{Opt}P$ . . . . .	103
5.5	OPEN ISSUE: Characterizing Closure Under Proper Decrement . . . . .	105
5.6	Bibliographic Notes . . . . .	106
6.	<b>The Polynomial Interpolation Technique . . . . .</b>	109
6.1	GEM: Interactive Protocols for the Permanent . . . . .	110
6.2	Enumerators for the Permanent . . . . .	119
6.3	$IP = PSPACE$ . . . . .	122
6.4	$MIP = NEXP$ . . . . .	133
6.5	OPEN ISSUE: The Power of the Provers . . . . .	163
6.6	Bibliographic Notes . . . . .	163
7.	<b>The Nonsolvable Group Technique . . . . .</b>	167
7.1	GEM: Width-5 Branching Programs Capture Nonuniform- $NC^1$ . . . . .	168
7.2	Width-5 Bottleneck Machines Capture $PSPACE$ . . . . .	176
7.3	Width-2 Bottleneck Computation . . . . .	181
7.4	OPEN ISSUE: How Complex Is Majority-Based Probabilistic Symmetric Bottleneck Computation? . . . . .	192
7.5	Bibliographic Notes . . . . .	192
8.	<b>The Random Restriction Technique . . . . .</b>	197
8.1	GEM: The Random Restriction Technique and a Polynomial-Size Lower Bound for Parity . . . . .	197
8.2	An Exponential-Size Lower Bound for Parity . . . . .	207
8.3	$PH$ and $PSPACE$ Differ with Probability One . . . . .	218
8.4	Oracles That Make the Polynomial Hierarchy Infinite . . . . .	222
8.5	OPEN ISSUE: Is the Polynomial Hierarchy Infinite with Probability One? . . . . .	231
8.6	Bibliographic Notes . . . . .	231

<b>9. The Polynomial Technique</b>	235
9.1 GEM: The Polynomial Technique	236
9.2 Closure Properties of PP	241
9.3 The Probabilistic Logspace Hierarchy Collapses	252
9.4 OPEN ISSUE: Is PP Closed Under Polynomial-Time Turing Reductions?	259
9.5 Bibliographic Notes	260
<b>A. A Rogues' Gallery of Complexity Classes</b>	263
A.1 P: Determinism	264
A.2 NP: Nondeterminism	266
A.3 Oracles and Relativized Worlds	268
A.4 The Polynomial Hierarchy and Polynomial Space: The Power of Quantifiers	270
A.5 E, NE, EXP, and NEXP	274
A.6 P/Poly: Small Circuits	276
A.7 L, NL, etc.: Logspace Classes	277
A.8 NC, AC, LOGCFL: Circuit Classes	279
A.9 UP, FewP, and US: Ambiguity-Bounded Computation and Unique Computation	281
A.10 #P: Counting Solutions	286
A.11 ZPP, RP, coRP, and BPP: Error-Bounded Probabilism	288
A.12 PP, C = P, and SPP: Counting Classes	290
A.13 FP, NPSV, and NPMV: Deterministic and Nondeterministic Functions	291
A.14 P-Sel: Semi-feasible Computation	294
A.15 $\oplus P$ , $\text{Mod}_k P$ : Modulo-Based Computation	297
A.16 SpanP, OptP: Output-Cardinality and Optimization Function Classes	297
A.17 IP and MIP: Interactive Proof Classes	299
A.18 PBP, SF, SSF: Branching Programs and Bottleneck Computation	300
<b>B. A Rogues' Gallery of Reductions</b>	305
B.1 Reduction Definitions: $\leq_m^p$ , $\leq_T^p$ , ...	305
B.2 Shorthands: R and E	307
B.3 Facts about Reductions	307
B.4 Circuit-Based Reductions: $\text{NC}^k$ and $\text{AC}^k$	308
B.5 Bibliographic Notes	308
<b>References</b>	309
<b>Index</b>	335



# 1. The Self-Reducibility Technique

A set  $S$  is *sparse* if it contains at most polynomially many elements at each length, i.e.,

$$(\exists \text{ polynomial } p)(\forall n)[|\{x \mid x \in S \wedge |x| = n\}| \leq p(n)]. \quad (1.1)$$

This chapter studies one of the oldest questions in computational complexity theory: Can sparse sets be NP-complete?

As we noted in the Preface, the proofs of most results in complexity theory rely on algorithms, and the proofs in this chapter certainly support that claim. In Sect. 1.1, we<sup>1</sup> will use a sequence of increasingly elaborate deterministic tree-pruning and interval-pruning procedures to show that sparse sets cannot be  $\leq_m^P$ -complete, or even  $\leq_{btt}^P$ -hard, for NP unless  $P = NP$ . (The appendices contain definitions of and introductions to the reduction types, such as  $\leq_m^P$  and  $\leq_{btt}^P$ , and the complexity classes, such as  $P$  and  $NP$ , that are used in this book.)

Section 1.2 studies whether NP can have  $\leq_T^P$ -complete or  $\leq_T^P$ -hard sparse sets.  $P^{NP[\mathcal{O}(\log n)]}$  denotes the class of languages that can be accepted by some deterministic polynomial-time Turing machine allowed at most  $\mathcal{O}(\log n)$  queries to some NP oracle. In Sect. 1.2, we will—via binary search, self-reducibility algorithms, and nondeterministic algorithms—prove that sparse sets cannot be  $\leq_T^P$ -complete for NP unless the polynomial hierarchy collapses to  $P^{NP[\mathcal{O}(\log n)]}$ , and that sparse sets cannot be  $\leq_T^P$ -hard for NP unless the polynomial hierarchy collapses to  $NP^{NP}$ .

As is often the case in complexity-theoretic proofs, we will typically use in the construction of our algorithms the hypothesis of the theorem that the algorithm is establishing (e.g., we will build a  $P$  algorithm for SAT, and will use in the algorithm the—literally hypothetical—sparse  $\leq_m^P$ -complete set for NP). In fact, this “theorems via algorithms under hypotheses” approach is employed in each section of this chapter.

Furthermore, most of Sects. 1.1 and 1.2 are unified by the spirit of their algorithmic attack, which is to exploit the “(disjunctive) self-reducibility” of SAT—basically, the fact that a boolean formula is satisfiable if and only if either it is satisfiable with its first variable set to False or it is satisfiable with

---

<sup>1</sup> In this book, “we” usually refers to the authors and the readers as we travel together in our exploration of complexity theory.

its first variable set to True. A partial exception to the use of this attack in those sections is the left set technique, which we use in Sect. 1.1.2. This technique, while in some sense a veiled tree-pruning procedure inspired by a long line of self-reducibility-based tree-pruning procedures, adds a new twist to this type of argument, rather than being a direct invocation of SAT's self-reducibility.

Section 1.3 studies not whether there are sparse NP-complete sets, but rather whether  $\text{NP} - \text{P}$  contains *any* sparse sets at all. Like the previous sections, this section employs explicit algorithmic constructions that themselves use objects hypothesized to exist by the hypotheses of the theorems for which they are providing proofs. The actual result we arrive at is that  $\text{NP} - \text{P}$  contains sparse sets if and only if deterministic and nondeterministic exponential time differ.

Throughout this book, we will leave the type of quantified variables implicit when it is clear from context what that type is. For example, in equation 1.1, the “ $(\forall n)$ ” is implicitly “ $(\forall n \in \{0, 1, 2, \dots\})$ ,” and “ $(\forall x)$ ” is typically a shorthand for “ $(\forall x \in \Sigma^*)$ .” We will use a colon to denote a constraint on a variable, i.e., “ $(\forall x : R(x)) [S(x)]$ ” means “ $(\forall x) [R(x) \implies S(x)]$ ,” and “ $(\exists x : R(x)) [S(x)]$ ” means “ $(\exists x) [R(x) \wedge S(x)]$ .” For any set  $A$  and any natural number  $n$ , we will use  $A^{\leq n}$  to denote the strings of  $A$  that are of length at most  $n$ , and we will use  $A^{=n}$  to denote the strings of  $A$  that are of length exactly  $n$ . Given a Turing machine  $M$ , we will use  $L(M)$  to denote the language accepted by the machine (with respect to whatever the acceptance mechanism of the machine is).

## 1.1 GEM: There Are No Sparse NP-Complete Sets Unless $\text{P}=\text{NP}$

### 1.1.1 Setting the Stage: The Pruning Technique

Before we turn to Mahaney's Theorem—NP has sparse complete sets only if  $\text{P} = \text{NP}$ —and its generalization to bounded-truth-table reductions, we first prove two weaker results that display the self-reducibility-based tree-pruning approach in a simpler setting. (Below, in a small abuse of notation we are taking “1” in certain places—such as in expressions like “1\*” —as a shorthand for the regular expression representing the set  $\{1\}$ .)

**Definition 1.1** *A set  $T$  is a tally set exactly if  $T \subseteq 1^*$ .*

**Theorem 1.2** *If there is a tally set that is  $\leq_m^p$ -hard for NP, then  $\text{P} = \text{NP}$ .*

**Corollary 1.3** *If there is a tally set that is NP-complete, then  $\text{P} = \text{NP}$ .*

We note in passing that if  $\text{P} = \text{NP}$ , then the singleton set  $\{1\}$  is trivially both NP-complete and coNP-complete. Thus, all the “if...then...” theorems



of this section (Sect. 1.1) have true converses. We state them in “if...then...” form to stress the interesting direction.

**Proof of Theorem 1.2** Let  $T$  be a tally set that is  $\leq_m^P$ -hard for NP. Since the NP-complete set

$$\text{SAT} = \{f \mid f \text{ is a satisfiable boolean formula}\}$$

is in NP and  $T$  is  $\leq_m^P$ -hard for NP, it follows that  $\text{SAT} \leq_m^P T$ . Let  $g$  be a deterministic polynomial-time function many-one reducing SAT to  $T$ . Let  $k$  be an integer such that  $(\forall x)[|g(x)| \leq |x|^k + k]$ ; since  $g$  is computable by some deterministic polynomial-time Turing machine, such a  $k$  indeed must exist since that machine outputs at most one character per step.

We now give, under the hypothesis of the theorem, a deterministic polynomial-time algorithm for SAT, via a simple tree-pruning procedure. The input to our algorithm is a boolean formula  $F$ . Without loss of generality, let its variables be  $v_1, \dots, v_m$  and let  $m \geq 1$ . We will denote the result of assigning values to some of the variables of  $F$  via expressions of the following form:  $F[v_1 = \text{True}, v_3 = \text{False}]$ , where True denotes the constant true and False denotes the constant false. For example, if  $F = v_1 \vee v_2 \vee v_3$  then

$$F[v_1 = \text{True}, v_3 = \text{False}] = \text{True} \vee v_2 \vee \text{False},$$

and

$$(F[v_1 = \text{True}])[v_3 = \text{False}] = \text{True} \vee v_2 \vee \text{False}.$$

Our algorithm has stages numbered  $0, 1, \dots, m+1$ . At the end of each stage (except the final one), we pass forward a collection of boolean formulas. Initially, we view ourselves as having just completed Stage 0, and we view ourselves as passing forward from Stage 0 a collection,  $C$ , containing the single formula  $F$ .

**Stage  $i$ ,  $1 \leq i \leq m$ , assuming that the collection at the end of Stage  $i-1$  is the following collection of formulas:  $\{F_1, \dots, F_\ell\}$ .**

**Step 1** Let  $C$  be the collection

$$\{F_1[v_i = \text{True}], F_2[v_i = \text{True}], \dots, F_\ell[v_i = \text{True}], \\ F_1[v_i = \text{False}], F_2[v_i = \text{False}], \dots, F_\ell[v_i = \text{False}]\}.$$

**Step 2** Set  $C'$  to be  $\emptyset$ .

**Step 3** For each formula  $f$  in  $C$  (in arbitrary order) do:

1. Compute  $g(f)$ .
2. If  $g(f) \in 1^*$  and for no formula  $h \in C'$  does  $g(f) = g(h)$ , then add  $f$  to  $C'$ .

**End Stage  $i$  [ $C'$  is the collection that gets passed on to Stage  $i+1$ ]**

The action of our algorithm at Stage  $m+1$  is simple:  $F$  is satisfiable if and only if some member of the (variable-free) formula collection output by Stage  $m$  evaluates to being true.

As to the correctness of our algorithm, note that after Stage 0 it certainly holds that

$$\begin{aligned} &\text{the collection, } C, \text{ contains some satisfiable formula} \\ &\iff \\ &F \text{ is satisfiable,} \end{aligned} \tag{1.2}$$

since after Stage 0 formula  $F$  is the only formula in the collection. Note also that, for each  $i$ ,  $1 \leq i \leq m$ ,

$$\begin{aligned} &\text{the collection input to Stage } i \text{ contains some} \\ &\text{satisfiable formula} \\ &\iff \\ &\text{the collection output by Stage } i \text{ contains some} \\ &\text{satisfiable formula.} \end{aligned} \tag{1.3}$$

Will now argue that this is so, via using a self-reducibility-based argument. In the present context, the relevant self-reducibility fact is that for any formula  $F$  containing  $v$  as one of its variables,

$$\begin{aligned} &F \text{ is satisfiable} \iff \\ &((F[v = \text{True}] \text{ is satisfiable}) \vee (F[v = \text{False}] \text{ is satisfiable})), \end{aligned}$$

since any satisfying assignment must assign some value to each variable. So Step 1 of Stage  $i$  does no damage to our invariant, equation 1.3. What about Steps 2 and 3? (In terms of the connection to Step 1, it is important to keep in mind that if, for example, formula  $F$  having variable  $v$  is in our collection at the start of the stage and is satisfiable, then it must be the case that

$$(F[v = \text{True}] \text{ is satisfiable}) \vee (F[v = \text{False}] \text{ is satisfiable}),$$

so it must be the case that

$$g(F[v = \text{True}]) \in T \vee g(F[v = \text{False}]) \in T.$$

And of course,  $T \subseteq 1^*$ .) Steps 2 and 3 “prune” the formula set as follows. Each formula  $f$  from Step 1 is kept unless either

- a.  $g(f) \notin 1^*$ , or
- b.  $g(f) \in 1^*$  but some  $h \in C'$  has  $g(f) = g(h)$ .

Both these ways, (a) and (b), of dropping formulas are harmless. Recall that  $\text{SAT} \leq_m^p T$  via function  $g$ , and so if  $f \in \text{SAT}$  then  $g(f) \in T$ . However, regarding (a),  $T \subseteq 1^*$  so if  $g(f) \notin 1^*$  then  $g(f) \notin T$ , and so  $f \notin \text{SAT}$ . Regarding (b), if  $g(f) = g(h)$  and  $h$  has already been added to the collection to be output by Stage (i), then there is no need to output  $f$  as—since  $\text{SAT} \leq_m^p T$  via reduction  $g$ —we know that

$$f \in \text{SAT} \iff g(f) \in T$$

and

$$h \in \text{SAT} \iff g(h) \in T.$$

Thus,  $f \in \text{SAT} \iff h \in \text{SAT}$ , and so by discarding  $f$  but leaving in  $h$ , we do no damage to our invariant, equation 1.3. So by equations 1.2 and 1.3 we see that  $F$  is satisfiable if and only if some formula output by Stage  $m$  is satisfiable. As the formulas output by Stage  $m$  have no free variables, this just means that one of them must evaluate to being true, which is precisely what Stage  $m + 1$  checks.

Thus, the algorithm correctly checks whether  $F$  is satisfiable. But is this a polynomial-time algorithm?  $|F|$  will denote the length of  $F$ , i.e., the number of bits in the representation of  $F$ . Let  $|F| = p$ . Note that after any stage there are at most  $p^k + k + 1$  formulas in the output collection, and each of these formulas is of size at most  $p$ . This size claim holds as each formula in an output collection is formed by one or more assignments of variables of  $F$  to being True or False, and such assignments certainly will not cause an *increase* in length (in a standard, reasonable encoding). We will say that a string  $s$  is a *tally string* exactly if  $s \in 1^*$ . The  $p^k + k + 1$  figure above holds as (due to the final part of Step 3 of our algorithm) we output at most one formula for each tally string to which  $(n^k + k\text{-time function}) g$  can map, and even if  $g$  outputs a 1 on each step,  $g$  can output in  $p^k + k$  steps no tally string longer than  $1^{p^k+k}$ . So, taking into account the fact that the empty string is a (degenerate) tally string, we have our  $p^k + k + 1$  figure. From this, from the specification of the stages, and from the fact that  $g$  itself is a polynomial-time computable function, it follows clearly that the entire algorithm runs in time polynomial in  $|F|$ .  $\square$

In the proof of Theorem 1.2, we used self-reducibility to split into two each member of a set of formulas, and then we pruned the resulting set using the fact that formulas mapping to non-tally strings could be eliminated, and the fact that only one formula mapping to a given tally string need be kept. By repeating this process we walked down the self-reducibility tree of any given formula, yet we pruned that tree well enough to ensure that only a polynomial number of nodes had to be examined at each level of the tree. By the self-reducibility tree—more specifically this is a disjunctive self-reducibility tree—of a formula, we mean the tree that has the formula as its root, and in which each node corresponding to a formula with some variables unassigned has as its left and right children the same formula but with the lexicographically first unassigned variable set respectively to True and to False.

In the proof of Theorem 1.2, we were greatly helped by the fact that we were dealing with whether *tally sets* are hard for NP. Tally strings are easily identifiable as such, and that made our pruning scheme straightforward. We now turn to a slightly more difficult case.

**Theorem 1.4** *If there is a sparse set that is  $\leq_m^p$ -hard for coNP, then  $P = NP$ .*

**Corollary 1.5** *If there is a sparse coNP-complete set, then  $P = NP$ .*

The proof of Theorem 1.4 goes as follows. As in the proof of Theorem 1.2, we wish to use our hypothesis to construct a polynomial-time algorithm for SAT. Indeed, we wish to do so by expanding and pruning the self-reducibility tree as was done in the proof of Theorem 1.2. The key obstacle is that the pruning procedure from the proof of Theorem 1.2 no longer works, since unlike tally sets, sparse sets are not necessarily “P-capturable” (a set is P-capturable if it is a subset of some sparse P set). In the following proof, we replace the tree-pruning procedure of Theorem 1.2 with a tree-pruning procedure based on the following counting trick. We expand our tree, while pruning only duplicates; we argue that if the tree ever becomes larger than a certain polynomial size, then the very failure of our tree pruning proves that the formula is satisfiable.

**Proof of Theorem 1.4** Let  $S$  be the (hypothetical) sparse set that is  $\leq_m^p$ -hard for coNP. For each  $\ell$ , let  $p_\ell(n)$  denote the polynomial  $n^\ell + \ell$ . Let  $d$  be such that  $(\forall n)(\|S^{\leq n}\| \leq p_d(n))$ .<sup>2</sup> Since  $\text{SAT} \in \text{NP}$ , it follows that  $\overline{\text{SAT}} \leq_m^p S$ . Let  $g$  be a deterministic polynomial-time function many-one reducing  $\overline{\text{SAT}}$  to  $S$ . Let  $k$  be an integer such that  $(\forall x)(|g(x)| \leq p_k(n))$ ; since  $g$  is computed by a deterministic polynomial-time Turing machine, such a  $k$  indeed exists.

We now give, under the hypothesis of this theorem, a deterministic polynomial-time algorithm for SAT, via a simple tree-pruning procedure. As in the proof of Theorem 1.2, let  $F$  be an input formula, and let  $m$  be the number of variables in  $F$ . Without loss of generality, let  $m \geq 1$  and let the variables of  $F$  be named  $v_1, \dots, v_m$ . Each stage of our construction will pass forward a collection of formulas. View Stage 0 as passing on to the next stage the collection containing just the formula  $F$ . We now specify Stage  $i$ . Note that Steps 1 and 2 are the same as in the proof of Theorem 1.2, Step 3 is modified, and Step 4 is new.

**Stage  $i$ ,  $1 \leq i \leq m$ , assuming the collection at the end of Stage  $i - 1$  is  $\{F_1, \dots, F_\ell\}$ .**

**Step 1** Let  $\mathcal{C}$  be the collection

$$\{F_1[v_i = \text{True}], F_2[v_i = \text{True}], \dots, F_\ell[v_i = \text{True}], \\ F_1[v_i = \text{False}], F_2[v_i = \text{False}], \dots, F_\ell[v_i = \text{False}]\}.$$

**Step 2** Set  $\mathcal{C}'$  to be  $\emptyset$ .

**Step 3** For each formula  $f$  in  $\mathcal{C}$  (in arbitrary order) do:

1. Compute  $g(f)$ .
2. If for no formula  $h \in \mathcal{C}'$  does  $g(f) = g(h)$ , then add  $f$  to  $\mathcal{C}'$ .

<sup>2</sup> The  $\|S^{\leq n}\|$ , as opposed to the  $\|S^{=n}\|$  that implicitly appears in the definition of “sparse set” (equation 1.1), is not a typographical error. Both yield valid and equivalent definitions of the class of sparse sets. The  $\|S^{=n}\|$  approach is, as we will see in Chap. 3, a bit more fine-grained. However, the proof of the present theorem works most smoothly with the  $\|S^{\leq n}\|$  definition.

**Step 4** If  $C'$  contains at least  $p_d(p_k(|F|)) + 1$  elements, stop and immediately declare that  $F \in \text{SAT}$ . (The reason for the  $p_d(p_k(|F|)) + 1$  figure will be made clear below.)

**End Stage  $i$**  [ $C'$  is the collection that gets passed on to Stage  $i + 1$ ]

The action of our algorithm at Stage  $m + 1$  is as follows: If some member of the (variable-free) formula collection output by Stage  $m$  evaluates to being true we declare  $F \in \text{SAT}$ , and otherwise we declare  $F \notin \text{SAT}$ .

Why does this algorithm work? Let  $n$  represent  $|F|$ . Since  $p_d(p_k(n)) + 1$  is a polynomial in the input size,  $F$ , it is clear that the above algorithm runs in polynomial time. If the hypothesis of Step 4 is never met, then the algorithm is correct for reasons similar to those showing the correctness of the proof of Theorem 1.2.

If Step 4 is ever invoked, then at the stage at which it is invoked, we have  $p_d(p_k(n)) + 1$  distinct strings being mapped to by the non-pruned nodes at the current level of our self-reducibility tree. (Recall that by the self-reducibility tree—more specifically this is a disjunctive self-reducibility tree—of a formula, we mean the tree that has the formula as its root, and in which each node corresponding to a formula with some variables unassigned has as its left and right children the same formula but with the lexicographically first unassigned variable set respectively to True and False.) Note that each of these mapped-to strings is of length at most  $p_k(n)$  since that is the longest string that reduction  $g$  can output on inputs of size at most  $n$ . However, there are only  $p_d(p_k(n))$  strings in  $S^{\leq p_k(n)}$ . As usual,  $\Sigma$  denotes our alphabet, and as usual we take  $\Sigma = \{0, 1\}$ . So since the formulas in our collection map to  $p_d(p_k(n)) + 1$  distinct strings in  $(\Sigma^*)^{\leq p_k(n)}$ , *at least one formula in our collection, call it  $H$ , maps under the action of  $g$  to a string in  $\bar{S}$ .*<sup>3</sup> So  $g(H) \notin S$ . However,  $\bar{\text{SAT}}$  reduces to  $S$  via  $g$ , so  $H$  is satisfiable. Since  $H$  was obtained by making substitutions to some variables of  $F$ , it follows that  $F$  is satisfiable. Thus, if the hypothesis of Step 4 is ever met, it is indeed correct to halt immediately and declare that  $F$  is satisfiable.  $\square$  Theorem 1.4

**Pause to Ponder 1.6** *In light of the comment in footnote 3, change the proof so that Step 4 does not terminate the algorithm, but rather the algorithm drives forward to explicitly find a satisfying assignment for  $F$ . (Hint: The crucial point is to, via pruning, keep the tree from getting too large. The following footnote contains a give-away hint.)*<sup>4</sup>

<sup>3</sup> Note that in this case we know that such an  $H$  exists, but we have no idea which formula is such an  $H$ . See Pause to Ponder 1.6 for how to modify the proof to make it more constructive.

<sup>4</sup> Change Step 4 so that, as soon as  $C'$  contains  $p_d(p_k(n)) + 1$  formulas, no more elements are added to  $C'$  at the current level.

### 1.1.2 The Left Set Technique

**1.1.2.1 Sparse Complete Sets for NP.** So far we have seen, as the proofs of Theorems 1.2 and 1.4, tree-pruning algorithms that show that “thin” sets cannot be hard for certain complexity classes. Inspired by these two results, Mahaney extended them by proving the following lovely, natural result.

**Theorem 1.7** *If NP has sparse complete sets then  $P = NP$ .*

**Pause to Ponder 1.8** *The reader will want to convince him- or herself of the fact that the approach of the proof of Theorem 1.4 utterly fails to establish Theorem 1.7. (See this footnote for why.<sup>5</sup>)*

We will not prove Theorem 1.7 now since we soon prove, as Theorem 1.10, a more general result showcasing the left set technique, and that result will immediately imply Theorem 1.7. Briefly put, the new technique needed to prove Theorems 1.7 and 1.10 is the notion of a “left set.” Very informally, a left set fills in gaps so as to make binary search easier.

Theorem 1.7 establishes that if there is a sparse NP-complete set then  $P = NP$ . For NP, the existence of sparse NP-hard sets and the existence of sparse NP-complete sets stand or fall together. (One can alternatively conclude this from the fact that Theorem 1.10 establishes its result for  $NP\text{-}\leq_{btt}^P$ -hardness rather than merely for  $NP\text{-}\leq_{btt}^P$ -completeness.)

**Theorem 1.9** *NP has sparse  $\leq_m^P$ -hard sets if and only if NP has sparse  $\leq_m^P$ -complete sets.*

**Proof** The “if” direction is immediate. So, we need only prove that if NP has a  $\leq_m^P$ -hard sparse set then it has a  $\leq_m^P$ -complete sparse set. Let  $S$  be any sparse set that is  $\leq_m^P$ -hard for NP. Since  $S$  is  $\leq_m^P$ -hard, it holds that  $SAT \leq_m^P S$ . Let  $f$  be a polynomial-time computable function that many-one reduces SAT to  $S$ . Define

$$S' = \{0^k \# y \mid k \geq 0 \wedge (\exists x \in SAT)[k \geq |x| \wedge f(x) = y]\}.$$

The rough intuition here is that  $S'$  is almost  $f(SAT)$ , except to make the proof work it via the  $0^k$  also has a padding part. Note that if  $0^k \# z \in S'$  then certainly  $z \in S$ .  $S'$  is clearly in NP, since to test whether  $0^k \# z$  is in  $S'$  we nondeterministically guess a string  $x$  of length at most  $k$  and we nondeterministically guess a potential certificate of  $x \in SAT$  (i.e., we guess a complete assignment of the variables of the formula  $x$ ), and (on each guessed path) we accept if the guessed string/certificate pair is such that  $f(x) = z$

<sup>5</sup> The analogous proof would merely be able to claim that if the tree were getting “bushy,” there would be at least one *unsatisfiable* formula among the collection. This says nothing regarding whether some other formula might be satisfiable. Thus, even if the set  $C'$  is getting very large, we have no obvious way to prune it.

and the certificate proves that  $x \in \text{SAT}$ . Given that  $S$  is sparse, it is not hard to see that  $S'$  is also sparse. Finally,  $S'$  is NP-hard because, in light of the fact that  $\text{SAT} \leq_m^p S$  via polynomial-time reduction  $f$ , it is not hard to see that  $\text{SAT} \leq_m^p S'$  via the reduction  $f'(x) = 0^{|x|} \# f(x)$ .  $\square$

We now turn to presenting the left set technique. We do so via proving that if some sparse set is NP-complete under bounded-truth-table reductions then  $P = NP$ .

**Theorem 1.10** *If there is a sparse set then  $P = NP$  that is  $\leq_{btt}^p$ -hard for NP, then  $P = NP$ .*

In the rest of the section, we prove Theorem 1.10.

**1.1.2.2 The Left Set and  $w_{\max}$ .** Let  $L$  be an arbitrary member of NP. There exist a polynomial  $p$  and a language in  $A \in P$  such that, for every  $x \in \Sigma^*$ ,

$$x \in L \iff (\exists w \in \Sigma^{p(|x|)})(\langle x, w \rangle \in A).$$

For each  $x \in \Sigma^*$  and  $w \in \Sigma^*$ , call  $w$  a *witness* for  $x \in L$  with respect to  $A$  and  $p$  if  $|w| = p(|x|)$  and  $\langle x, w \rangle \in A$ . Define the *left set with respect to  $A$  and  $p$* , denoted by  $\text{Left}[A, p]$ , to be

$$\{\langle x, y \rangle \mid x \in \Sigma^* \wedge y \in \Sigma^{p(|x|)} \wedge (\exists w \in \Sigma^{p(|x|)})(w \geq y \wedge \langle x, w \rangle \in A)\},$$

i.e.,  $\text{Left}[A, p]$  is the set of all  $\langle x, y \rangle$  such that  $y$  belongs to  $\Sigma^{p(|x|)}$  and is “to the left” of some witness for  $x \in L$  with respect to  $A$  and  $p$ . For each  $x \in \Sigma^*$ , define

$$w_{\max}(x) = \max\{y \in \Sigma^{p(|x|)} \mid \langle x, y \rangle \in A\};$$

if  $\{y \in \Sigma^{p(|x|)} \mid \langle x, y \rangle \in A\}$  is empty, then  $w_{\max}(x)$  is undefined. In other words,  $w_{\max}(x)$  is the lexicographic maximum of the witnesses for  $x \in L$  with respect to  $A$  and  $p$ . Clearly, for every  $x \in \Sigma^*$ ,

$$x \in L \iff w_{\max}(x) \text{ is defined,}$$

and

$$x \in L \iff (\exists y \in \Sigma^{p(|x|)})(\langle x, y \rangle \in \text{Left}[A, p]).$$

Furthermore, for every  $x \in \Sigma^*$ , the set

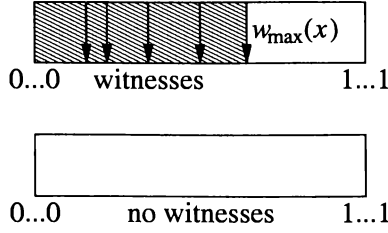
$$\{y \in \Sigma^{p(|x|)} \mid \langle x, y \rangle \in \text{Left}[A, p]\}$$

equals  $\{y \in \Sigma^{p(|x|)} \mid 0^{p(|x|)} \leq y \leq w_{\max}(x)\}$  if  $x \in L$  and equals  $\emptyset$  otherwise (see Fig. 1.1). More precisely,

$$(\forall x \in \Sigma^*)(\forall y \in \Sigma^{p(|x|)})(\langle x, y \rangle \in \text{Left}[A, p] \iff y \in w_{\max}(x)).$$

Also,

$$\begin{aligned} (\forall x \in \Sigma^*)(\forall y, y' \in \Sigma^{p(|x|)})(\langle x, y \rangle \in \text{Left}[A, p] \wedge (y' < y)) \\ \implies \langle x, y' \rangle \in \text{Left}[A, p]. \end{aligned} \quad (1.4)$$



**Fig. 1.1** The left set  $Left[A, p]$ . *Top:* The case when  $x \in L$ . The arrows above are witnesses for  $x \in L$  with respect to  $A$  and  $p$ ,  $w_{\max}(x)$  is the rightmost arrow, and the shaded area is  $\{y \in \Sigma^{p(|x|)} \mid \langle x, y \rangle \in Left[A, p]\}$ . *Bottom:* The case when  $x \notin L$ .  $w_{\max}(x)$  is undefined and  $\{y \in \Sigma^{p(|x|)} \mid \langle x, y \rangle \in Left[A, p]\} = \emptyset$ .

Note that  $Left[A, p]$  is in NP via the nondeterministic Turing machine that, on input  $\langle x, y \rangle$ , guesses  $w \in \Sigma^{p(|x|)}$ , and accepts if

$$(y \in \Sigma^{p(|x|)}) \wedge (y \leq w) \wedge (\langle x, y \rangle \in A)$$

and rejects otherwise.

Below, we introduce a useful characterization of  $\leq_{btt}^p$  reductions. Let  $k \geq 1$ . A *k-truth-table condition* is a  $(k+1)$ -tuple  $C$  such that the first component of  $C$  is a boolean function of arity  $k$  and each of the other components of  $C$  is a member of  $\Sigma^*$ . For a *k-truth-table condition*  $C = (\alpha, v_1, \dots, v_k)$ , we call  $\alpha$  the *truth-table of C*, call  $\{w \mid (\exists i : 1 \leq i \leq k)[w = v_i]\}$  the *queries of C*, and, for each  $i$ ,  $1 \leq i \leq k$ , call  $v_i$  the *i-th query of C*. For a language  $D$ , we say that the *k-truth-table condition*  $(\alpha, v_1, \dots, v_k)$  is satisfied by  $D$  if  $\alpha(\chi_D(v_1), \dots, \chi_D(v_k)) = 1$ .

**Proposition 1.11** *Suppose that a language  $C$  is  $\leq_{btt}^p$ -reducible to a language  $D$ . Then there exist an integer  $k \geq 1$  and a polynomial-time computable function  $f$  from  $\Sigma^*$  to the set of all *k-truth-table conditions*, such that for all  $u \in \Sigma^*$ ,*

$$u \in C \iff f(u) \text{ is satisfied by } D.$$

**Proof of Proposition 1.11** Suppose that, for some  $k \geq 1$ , a language  $C$  is  $\leq_{k-tt}^p$ -reducible to a language  $D$  via  $(f_0, B_0)$  such that  $f_0 \in \text{FP}$  and  $B_0 \in \text{P}$ . For all  $u \in \Sigma^*$ , there exist some  $l$ ,  $1 \leq l \leq k$ , and  $v_1, \dots, v_l \in \Sigma^*$ , such that

- $f_0(u) = v_1 \# \dots \# v_l \#$ , and
- $u \in Left[A, p] \iff u \# \chi_D(v_1) \dots \chi_D(v_l) \in B_0$ ,

where  $\# \notin \Sigma$ . Let  $f_1$  be the function from  $\Sigma^*$  to  $(\Sigma^* \#)^k$  defined for all  $u \in \Sigma^*$  by

$$f_1(u) = v_1 \# \dots \# v_l \#^{k+1-l},$$



where  $f_0(u) = v_1\# \cdots \# v_l\#$ . Define  $B_1$  to be the set of all  $u\#b_1 \cdots b_k$ ,  $u \in \Sigma^*$ ,  $b_1, \dots, b_k \in \Sigma$ , such that

$$u\#b_1 \cdots b_l \in B_0,$$

where  $l$  is the integer such that  $f_0(u) \in (\Sigma^*\#)^l$ . Since  $B_0 \in P$  and  $f_0 \in FP$ , it holds that  $B_1 \in P$  and  $f_1 \in FP$ .

Let  $\beta = \chi_D(\epsilon)$ . Let  $u \in \Sigma^*$  and let  $f_0(u) = v_1\# \cdots \# v_l\#$ . Then

$$u\#\chi_D(v_1) \cdots \chi_D(v_l) \in B_0 \iff u\#\chi_D(v_1) \cdots \chi_D(v_l)\beta^{k-l} \in B_1.$$

Since  $f_1(u) = v_1\# \cdots \# v_l\#^{k+1-l}$ , we have that the pair  $(f_1, B_1)$  witnesses that  $C \leq_{k-tt}^p D$ .

Define function  $f$  from  $\Sigma^*$  to the set of all  $k$ -truth-table conditions as follows: For each  $u \in \Sigma^*$ ,

$$f(u) = (\alpha, v_1, \dots, v_k),$$

where  $f_1(u) = v_1\# \cdots \# v_k\#$  and  $\alpha$  is the boolean function defined for all  $b_1, \dots, b_k \in \{0, 1\}^k$  by

$$\alpha(b_1, \dots, b_k) = \chi_{B_1}(u\#b_1 \cdots b_k).$$

Since  $f_1 \in FP$  and  $k$  is a constant,  $f \in FP$ . For every  $u \in \Sigma^*$ ,

$$u \in C \iff u\#\chi_D(v_1) \cdots \chi_D(v_k) \in B_1$$

and

$$u\#\chi_D(v_1) \cdots \chi_D(v_k) \in B_1 \iff \alpha(\chi_D(v_1), \dots, \chi_D(v_k)) = 1,$$

where  $f_1(u) = v_1\# \cdots \# v_k\#$ . So, for all  $u \in \Sigma^*$ ,

$$u \in C \iff f(u) \text{ is satisfied by } D.$$

Thus, the statement of the proposition holds.  $\square$  Proposition 1.11

Suppose that NP has a sparse  $\leq_{btt}^p$ -hard set,  $S$ . Since  $L$  was an arbitrary member of NP, it suffices to prove that  $L \in P$ . Since  $Left[A, p] \in NP$ ,  $Left[A, p] \leq_{btt}^p S$ . So, by Proposition 1.11, there exist some  $k \geq 1$  and  $f \in FP$  such that, for all  $u \in \Sigma^*$ ,  $f(u)$  is a  $k$ -truth-table condition, and

$$u \in Left[A, p] \iff f(u) \text{ is satisfied by } S.$$

In preparation for the remainder of the proof, we define some polynomials. Let  $p_1$  be a strictly increasing polynomial such that for all  $x \in \Sigma^*$  and  $y \in \Sigma^{p(|x|)}$ ,  $|\langle x, y \rangle| \leq p_1(|x|)$ . Let  $p_2$  be a strictly increasing polynomial such that for all  $u \in \Sigma^*$ , every query of  $f(u)$  has length at most  $p_2(|u|)$ . Let  $p_3$  be a strictly increasing polynomial such that for all integers  $n \geq 0$   $|S^{\leq n}| \leq p_3(n)$ . Define  $q(n) = p_3(p_2(p_1(n)))$ . Then, for all  $x \in \Sigma^*$ ,

$$|\{w \in S \mid (\exists y \in \Sigma^{p(|x|)})[w \text{ is a query of } f(\langle x, y \rangle)]\}| \leq q(|x|).$$

Define  $r(n) = k!2^k(2q(n)+1)^k$ . Also, for each  $d$ ,  $0 \leq d \leq k$ , define  $r_d(n) = (k-d)!2^{k-d}(2q(n)+1)^{k-d}$ . Note that  $r_0 = r$  and  $r_k$  is the constant 1 polynomial.

**1.1.2.3 A Polynomial-Time Search Procedure for  $w_{\max}$ .** To prove that  $L \in P$ , we will develop a polynomial-time procedure that, on input  $x \in \Sigma^*$ , generates a list of strings in  $\Sigma^{p(|x|)}$  such that, if  $w_{\max}(x)$  is defined then the list is guaranteed to contain  $w_{\max}(x)$ . The existence of such a procedure implies  $L \in P$  as follows: Let  $M$  be a Turing machine that, on input  $x \in \Sigma^*$ , runs the enumeration procedure to obtain a list of candidates for  $w_{\max}(x)$ , and accept if the list contains a string  $y$  such that  $\langle x, y \rangle \in A$  and reject otherwise. Since the enumeration procedure runs in polynomial time and  $A \in P$ ,  $M$  can be made polynomial-time bounded. Since the output list of the enumeration procedure is guaranteed to contain  $w_{\max}(x)$  if it is defined,  $M$  accepts if  $x \in L$ . If  $x \notin L$ , there is no  $y \in \Sigma^{p(|x|)}$  such that  $\langle x, y \rangle \in A$ , so  $M$  rejects  $x$ . Thus,  $M$  correctly decides  $L$ . Hence,  $L \in P$ .

In the rest of the proof we will focus on developing such an enumeration procedure. To describe the procedure we need to define some notions.

Let  $n \geq 1$  be an integer. Let  $I$  be a subset of  $\Sigma^n$ . We say that  $I$  is an *interval over  $\Sigma^n$*  if there exist  $y, z \in \Sigma^n$  such that

$$y \leq z \text{ and } I = \{u \in \Sigma^n \mid y \leq u \leq z\}.$$

We call  $y$  and  $z$  respectively the *left end* and the *right end* of  $I$ , and write  $[y, z]$  to denote  $I$ . Let  $I = [u, v]$  and  $J = [y, z]$  be two intervals over  $\Sigma^n$ . We say that  $I$  and  $J$  are *disjoint* if they are disjoint as sets, i.e., either  $v < y$  or  $z < u$ . If  $I$  and  $J$  are disjoint and  $v < y$ , we say that  $I$  is *lexicographically smaller than  $J$* , and write  $I < J$ .

Let  $x \in \Sigma^*$  and let  $\Lambda$  be a set of pairwise disjoint intervals over  $\Sigma^n$ . We say that  $\Lambda$  is *nice for  $x$*  if

$$x \in L \implies (\exists I \in \Lambda)[w_{\max}(x) \in I].$$

Note that for all  $x \in \Sigma^*$

- $\{[0^{p(|x|)}, 1^{p(|x|)}]\}$  is nice for  $x$  regardless of whether  $x \in L$ , and
- if  $x \notin L$ , then every set of pairwise disjoint intervals over  $\Sigma^{p(|x|)}$  is nice for  $x$ .

Let  $\tau$  be an ordered (possibly empty) list such that, if  $\tau$  is not empty then each entry of  $\tau$  is of the form  $(w, b)$  for some  $w \in \Sigma^*$  and  $b \in \{0, 1\}$ . We call such a list a *hypothesis list*. We say that a hypothesis list  $\tau$  is *correct* if every pair  $(w, b)$  in the list satisfies  $\chi_S(w) = b$ . Let  $x \in \Sigma^*$ , let  $\Lambda$  be a set of pairwise disjoint intervals over  $\Sigma^{p(|x|)}$ , let  $\Gamma$  be a subset of  $\Lambda$ , and let  $\tau$  be a hypothesis list. We say that  $\Gamma$  is a *refinement of  $\Lambda$  for  $x$  under  $\tau$*  if

$$((\Lambda \text{ is nice for } x) \wedge (\tau \text{ is correct})) \implies \Gamma \text{ is nice for } x.$$

The following fact states some useful properties of refinements.

**Fact 1.12**

1. If  $\Lambda = \Gamma_1 \cup \dots \cup \Gamma_m$  and  $\Gamma'_1, \dots, \Gamma'_k$  are refinements of  $\Gamma_1, \dots, \Gamma_k$  for  $x$  under  $\tau$ , respectively, then  $\Gamma'_1 \cup \dots \cup \Gamma'_k$  is a refinement of  $\Lambda$  for  $x$  under  $\tau$ .
2. If  $\Theta$  is a refinement of  $\Gamma$  for  $x$  under  $\tau$  and  $\Theta'$  is a refinement of  $\Theta$  for  $x$  under  $\tau$ , then  $\Theta'$  is a refinement of  $\Gamma$  for  $x$  under  $\tau$ .

To generate candidates for  $w_{\max}(x)$ , starting with the initial value

$$\Lambda = \{ [0^{p(|x|)}, 1^{p(|x|)}] \},$$

we repeat the following two-phase process  $p(|x|)$  times.

- **Splitting** Split each interval in  $\Lambda$  into upper and lower halves.
- **Culling** If  $|\Lambda| \leq r(|x|)$ , skip this phase. If  $|\Lambda| \geq r(|x|) + 1$ , do the following: Set  $\Upsilon$  to the empty list. Call a subroutine **CULL** on input  $(x, \Lambda, \tau)$  to obtain  $\Upsilon \subsetneq \Lambda$  that has cardinality less than or equal to  $r(|x|)$  and is nice for  $x$ . Replace  $\Lambda$  with  $\Upsilon$ .

When the two-phase process has been executed  $p(|x|)$  times, each interval in  $\Lambda$  has size exactly 1, i.e., is of the form  $[u, u]$  for some  $u \in \Sigma^{p(|x|)}$ . The output of the enumeration procedure is the list of all strings  $u \in \Sigma^{p(|x|)}$  such that  $[u, u] \in \Lambda$ .

Note that if  $\Lambda$  is nice for  $x$  at the beginning of the splitting phase then it is nice for  $x$  at the end of the splitting phase. Since both  $p$  and  $r$  are polynomials, if **CULL** runs in polynomial time, the entire generation procedure runs in polynomial time. Since **CULL** is guaranteed to output a refinement, if  $x \in L$  then there is always one interval in  $\Lambda$  that contains  $w_{\max}(x)$ . So, if  $x \in L$ ,  $w_{\max}(x)$  is included in the list of candidates at the end. So, we have only to show that a polynomial-time procedure **CULL** exists that, on input  $(x, \Lambda, \tau)$  with  $|\Lambda| \geq r(|x|) + 1$ , finds  $\Upsilon \subsetneq \Lambda$  having cardinality at most  $r(|x|)$  such that  $\Upsilon$  is a refinement of  $\Lambda$  for  $x$  under  $\tau$ .

For the sake of simplicity, in the following discussion, let  $x \in \Sigma^*$  be fixed. Since only splitting and elimination are the operations executed to modify intervals, we can assume that the intervals in  $\Lambda$  are pairwise disjoint during the entire enumeration procedure. So, for every pair of distinct intervals,  $I$  and  $J$ , appearing in the input to **CULL**, we will assume that they are disjoint, and thus, either  $I < J$  or  $I > J$ . We also induce a mapping from the set of all interval over  $\Sigma^{p(|x|)}$  to the set of all  $k$ -truth-table conditions. Let  $I = [u, v]$  be an interval over  $\Sigma^{p(|x|)}$ . The image of  $I$  induced by  $f$ , denoted by  $f[I]$ , is  $f(\langle x, u \rangle)$ . Let  $f[I] = (\alpha, w_1, \dots, w_k)$ . For each  $i$ ,  $1 \leq i \leq k$ ,  $Q[I, i]$  to denote  $w_i$ .

**1.1.2.4 The Structure of the Culling Method.** Each input  $(x, \Gamma, \tau)$  to **CULL** is supposed to satisfy the following conditions:

- $\tau$  is a hypothesis list and its length,  $|\tau|$ , is less than or equal to  $k$ .

- $\Gamma$  is a set of pairwise disjoint intervals over  $\Sigma^{p(|x|)}$  having cardinality strictly greater than  $r_{|\tau|}(|x|)$ .
- If  $|\tau| \geq 1$ , then the following condition holds: Let  $d = |\tau|$  and  $\tau = [(w_1, b_1), \dots, (w_d, b_d)]$  for some  $w_1, \dots, w_d \in \Sigma^*$  and  $b_1, \dots, b_d \in \{0, 1\}$ . Then, for all  $I \in \Gamma$ , there exist  $d$  pairwise distinct elements of  $\{1, \dots, k\}$ ,  $j_1, \dots, j_d$ , such that for all  $r$ ,  $1 \leq r \leq d$ ,  $Q[I, j_r] = w_r$ .

Given an input  $(x, \Gamma, \tau)$  that meets this specification, **CULL** may call itself recursively in the case when  $|\tau| < k$ . The number of recursive call that **CULL** makes is less than or equal to  $2(k - |\tau|)(2q(|x|) + 1)$  and the input to each recursive call is a triple of the form  $(x, \Gamma', \tau')$  for some  $\Gamma' \subseteq \Gamma$  and a hypothesis list  $\tau'$  such that  $|\tau'| = |\tau| + 1$ . Thus, the computation of **CULL** on input  $(x, \Gamma, \tau)$  can be viewed as a tree of depth bounded by  $k - |\tau|$ .

The hypothesis list  $\tau$  is used to refine, for each interval  $I \in \Gamma$ , the  $k$ -truth-table condition  $f[I]$  to a  $(k - |\tau|)$ -truth-table condition. We denote the refinement of  $f[I]$  with respect to  $\tau$  by  $f_\tau[I]$ . Suppose that  $\tau = [(w_1, b_1), \dots, (w_d, b_d)]$  for some  $d \geq 1$ ,  $w_1, \dots, w_d \in \Sigma^*$ , and  $b_1, \dots, b_d \in \{0, 1\}$ . Let  $I$  be an arbitrary interval in  $\Gamma$  and let  $f[I] = (\alpha, v_1, \dots, v_k)$ . Then,  $f_\tau[I] = (\beta, v_{j_1}, \dots, v_{j_{k-d}})$ , where  $\beta$  and  $v_{j_1}, \dots, v_{j_{k-d}}$  are defined as follows:

- For  $s = 1, \dots, d$ , in that order, let  $\rho_s$  be the smallest of  $r$ ,  $1 \leq r \leq k$ , such that  $(Q[I, r] = w_s) \wedge (\forall t : 1 \leq t \leq s - 1)[r \neq \rho_t]$ .
- For every  $i$ ,  $1 \leq i \leq k - d$ , let  $j_i$  be the  $i$ th smallest element in  $\{1, \dots, k\} - \{\rho_1, \dots, \rho_d\}$ .
- $\beta$  is the boolean function of arity  $(k - d)$  that is constructed from  $\alpha$  by simultaneously fixing for all  $s$ ,  $1 \leq s \leq d$ , the argument at position  $\rho_s$  to  $b_s$ .

We will write  $\beta_\tau[I]$  to denote the truth-table of  $f_\tau[I]$  and  $Q_\tau[I]$  to denote the queries of  $f_\tau[I]$ . Note that if the hypothesis list  $\tau$  is correct, then for all  $I \in \Gamma$ ,  $f[I]$  is satisfied by  $S$  if and only if  $f_\tau[I]$  is satisfied by  $S$ .

Suppose that  $|\tau| = k$ . Then, for all  $I \in \Gamma$ ,  $f_\tau[I]$  is a boolean function of arity 0, i.e., a boolean constant. Since  $r_k(|x|) = 1$ , **CULL** cannot select more than one interval from  $\Gamma$  to generate its output. **CULL** computes  $S = \{I \in \Gamma \mid f_\tau[I] = (\text{True})\}$ . If  $S$  is nonempty, **CULL** selects the largest element in  $S$  in lexicographic order; if  $S$  is empty, **CULL** outputs the empty set. We claim that the output of **CULL** is a refinement of  $\Gamma$  for  $x$  under  $\tau$ . To see why, suppose that  $\Gamma$  is nice for  $x$  and that the hypothesis list  $\tau$  is correct. Then, for every interval  $I = [u, v] \in \Gamma$ ,  $w_{\max}(x)$  is less than or equal to  $u$  if  $\beta_\tau[I] = \text{True}$  and is strictly greater than  $u$  otherwise. So, for every interval  $I \in \Gamma$ ,  $w_{\max}(x) \notin I$  if either  $\beta_\tau[I] = \text{False}$  or  $I$  is not the largest element in  $S$  in lexicographic order. Thus, it is safe to select the largest element in  $S$  in lexicographic order.

On the other hand, suppose that  $|\tau| < k$ . Then **CULL** executes two phases, Phases 1 and 2. In Phase 1, **CULL** eliminates intervals from  $\Gamma$  so that

$f_\tau[I] \neq f_\tau[J]$  for every pair of distinct intervals  $I, J \in \Gamma$ . In Phase 2, **CULL** selects from  $\Gamma$  disjoint subsets,  $\Gamma_1, \dots, \Gamma_l$ ,  $1 \leq l \leq 2(k - |\tau|)(2q(n) + 1)$ , where it is guaranteed that no intervals in the rest of  $\Gamma$  contain  $w_{\max}(x)$  if  $\tau$  is correct. For each of the subsets, **CULL** makes exactly two recursive calls. The output of **CULL**( $x, \Gamma, \tau$ ) is the union of the outputs of all the  $2l$  recursive calls.

Below we describe the two phases of **CULL**. Let  $(x, \Gamma, \tau)$  be an input to **CULL**. Let  $d = |\tau|$ . Suppose that  $0 \leq d \leq k - 1$ .

*Phase 1: Making  $f_\tau$  unambiguous* **CULL** executes the following:

- While there exist two intervals  $I, I' \in \Gamma$  such that  $I < I' \in \Gamma$  and  $f_\tau[I] = f_\tau[I']$ , find the smallest such pair  $(I, I')$  in lexicographic order and eliminate  $I$  from  $\Gamma$ .

Let  $\Gamma'$  be the set  $\Gamma$  when **CULL** quits the loop. We claim that  $\Gamma'$  is a refinement of  $\Gamma$  for  $x$  under  $\tau$ . To see why, assume that  $\tau$  is correct. Suppose that  $\Gamma$  contains two intervals  $I = [u, v]$  and  $I' = [u', v']$  such that  $I < I'$  and  $f_\tau[I] = f_\tau[I']$ . Then  $f_\tau[I]$  is satisfied by  $S$  if and only if  $f_\tau[I']$  is satisfied by  $S$ . Since  $\tau$  is correct,  $f_\tau[I]$  is satisfied by  $S$  if and only if  $f[I]$  is satisfied by  $S$ . Also,  $f_\tau[I']$  is satisfied by  $S$  if and only if  $f[I']$  is satisfied by  $S$ . So,  $f[I]$  is satisfied by  $S$  if and only if  $f[I']$  is satisfied by  $S$ . Then, by equation 1.4, we have

$$w_{\max}(x) \geq u \iff w_{\max}(x) \geq u'.$$

In particular,  $w_{\max}(x) \geq u \implies w_{\max}(x) \geq u'$ . This implies that either  $w_{\max}(x) < u$  or  $w_{\max}(x) \geq u'$ . Since  $u' > v \geq u$ , it holds that either  $w_{\max}(x) < u$  or  $w_{\max}(x) > v$ . Thus,  $w_{\max}(x) \notin I$ . So,  $\Gamma - \{I\}$  is a refinement of  $\Gamma$  for  $x$  under  $\tau$ . By part 2 of Fact 1.12, each time an interval is eliminated by executing the above algorithm, the resulting set is a refinement of  $\Gamma$  for  $x$  under  $\tau$ . Thus,  $\Gamma'$  is a refinement of  $\Gamma$  for  $x$  under  $\tau$ .

*Phase 2: Refining  $\Gamma'$*  **CULL** splits  $\Gamma'$  into two groups  $\Delta_0$  and  $\Delta_1$ , where for each  $b \in \{0, 1\}$ ,

$$\Delta_b = \{I \in \Gamma' \mid \beta_\tau[I](0, \dots, 0) = b\}.$$

**CULL** refines  $\Delta_0$  and  $\Delta_1$  separately.

*Refining  $\Delta_0$ :* Suppose  $\Delta_0$  is nonempty. **CULL** computes an integer  $m \geq 1$  and a sequence of intervals  $I_1, I_2, \dots, I_m \in \Delta_0$ ,  $I_1 < I_2 < \dots < I_m$ , as follows:

- $I_1$  is the lexicographic minimum of the intervals in  $\Delta_0$ .
- For each  $t \geq 1$  such that  $I_t$  is defined, let  $S_{t+1} = \{I \in \Delta_0 \mid (\forall j : 1 \leq j \leq t) [Q_\tau[I] \cap Q_\tau[I_j] = \emptyset]\}$ . If  $S_{t+1} = \emptyset$ , then  $I_{t+1}$  is undefined. If  $S_{t+1} \neq \emptyset$ , then  $I_{t+1}$  is the lexicographic minimum of the intervals in  $S_{t+1}$ .
- $m$  is the largest  $t$  such that  $I_t$  is defined.

Define

$$\Delta'_0 = \begin{cases} \Delta_0 & \text{if } m \leq q(|x|), \\ \{J \in \Delta_0 \mid J < I_{q(|x|)+1}\} & \text{otherwise.} \end{cases}$$

We claim that  $\Delta'_0$  is a refinement of  $\Delta_0$  for  $x$  under  $\tau$ . If  $m \leq q(|x|)$ , then  $\Delta'_0 = \Delta_0$ , so  $\Delta'_0$  clearly is a refinement of  $\Delta_0$  for  $x$  under  $\tau$ . So, suppose that  $m \geq q(n) + 1$ , and thus, that  $\Delta'_0 \subsetneq \Delta_0$ . Suppose that  $\tau$  is correct. For each  $j$ ,  $1 \leq j \leq m$ , let  $I_j = [u_j, v_j]$ . Assume  $u_{q(|x|)+1} \leq w_{\max}(x)$ . Then for all  $j$ ,  $1 \leq j \leq q(|x|) + 1$ ,  $u_j \leq w_{\max}(x)$ . Since  $\tau$  is correct, for all  $j$ ,  $1 \leq j \leq q(|x|) + 1$ ,  $f_\tau[I_j]$  is satisfied by  $S$ . For each  $j$ ,  $1 \leq j \leq q(|x|) + 1$ ,  $\beta_\tau[I_j](0, \dots, 0) = 0$ , and thus,  $Q_\tau[I_j] \cap S \neq \emptyset$ . The intervals  $I_1, \dots, I_m$  are chosen so that  $Q_\tau[I_1], \dots, Q_\tau[I_m]$  are pairwise disjoint. Thus,

$$\|S \cap \bigcup_{1 \leq j \leq q(|x|)+1} Q_\tau[I_j]\| \geq q(|x|) + 1.$$

This is a contradiction, because the number of strings in  $S$  that may appear as a query string in  $f(\langle x, y \rangle)$  for some  $y \in \Sigma^{p(|x|)}$  is at most  $q(|x|)$ . Thus,  $w_{\max}(x) < u_{q(|x|)+1}$ . So, all intervals  $I \in \Delta_0$  whose left end is greater than or equal to  $u_{q(|x|)+1}$  can be safely eliminated from  $\Delta_0$ . Hence,  $\Delta'_0$  is a refinement of  $\Delta_0$  for  $x$  under  $\tau$ .

Let  $m_0 = \min\{m, q(|x|)\}$  and

$$R = \bigcup_{1 \leq j \leq m_0} Q_\tau[I_j].$$

Let  $h = \|R\|$ . Then  $h \leq (k-d)m_0$ . For every  $I \in \Delta'_0$ , there exists some  $y \in R$  such that  $y \in Q_\tau[I]$ . Let  $y_1, \dots, y_h$  be the enumeration of the strings in  $R$  in lexicographic order. For each  $j$ ,  $1 \leq j \leq h$ , let

$$\Theta_j = \{I \mid (I \in \Delta_0) \wedge (\forall s : 1 \leq s \leq j-1) [I \notin \Theta_s] \wedge (y_j \in Q_\tau[I])\}.$$

By the choice of the intervals  $I_1, \dots, I_m$ ,  $\Theta_1, \dots, \Theta_h$  are all nonempty and  $\Delta'_0 = \Theta_1 \cup \dots \cup \Theta_h$ . For each  $j$ ,  $1 \leq j \leq h$ , and each  $b \in \{0, 1\}$ , let  $\Theta'_{j,b}$  be the set of intervals that **CULL** on input  $(x, \Theta_j, \tau)$  outputs as a refinement of  $\Theta_j$  for  $x$  under  $\tau'$ , where  $\tau'$  is  $\tau$  with the pair  $(y_j, b)$  appended at the end. Let  $\Upsilon_0 = \bigcup_{1 \leq j \leq h} \bigcup_{b \in \{0,1\}} \Theta'_{j,b}$ . By part 1 of Fact 1.12, if **CULL** correctly computes a refinement for all the recursive calls, then  $\Upsilon_0$  is a refinement of  $\Delta_0$  for  $x$  under  $\tau$ .

*Dividing  $\Delta_1$ :* Suppose that  $\Delta_1 \neq \emptyset$ . **CULL** computes an integer  $m \geq 1$  and a sequence of intervals  $I_1, I_2, \dots, I_m \in \Delta_0$ ,  $I_1 > I_2 > \dots > I_m$ , as follows:

- $I_1$  is the lexicographic maximum of the intervals in  $\Delta_1$ .
- For each  $t \geq 1$  such that  $I_t$  is defined, let  $S_{t+1} = \{J \in \Delta_1 \mid (\forall j : 1 \leq j \leq t) [Q_\tau[I] \cap Q_\tau[I_j] = \emptyset]\}$ . If  $S_{t+1} = \emptyset$ , then  $I_{t+1}$  is undefined. If  $S_{t+1} \neq \emptyset$ , then  $I_{t+1}$  is the lexicographic maximum of the intervals in  $\Delta_1$ .
- $m$  is the largest  $t$  such that  $I_t$  is defined.

Define

$$\Delta'_1 = \begin{cases} \Delta_1 & \text{if } m \leq q(|x|) + 1, \\ \{J \in \Delta'_1 \mid J \geq I_{q(|x|)+1}\} & \text{otherwise.} \end{cases}$$

We claim that  $\Delta'_1$  is a refinement of  $\Delta_1$  for  $x$  under  $\tau$ . If  $m \leq q(|x|) + 1$ , then  $\Delta'_1 = \Delta_1$ , so  $\Delta'_1$  clearly is a refinement of  $\Delta_1$  for  $x$  under  $\tau$ . So, suppose that  $m \geq q(n) + 2$ , and thus, that  $\Delta'_1 \subsetneq \Delta_1$ . Suppose that  $\tau$  is correct. For each  $j$ ,  $1 \leq j \leq m$ , let  $I_j = [u_j, v_j]$ . Assume  $u_{q(|x|)+1} > w_{\max}(x)$ . Then for all  $j$ ,  $1 \leq j \leq q(|x|) + 1$ ,  $u_j > w_{\max}(x)$ . Since  $\tau$  is correct, for all  $j$ ,  $1 \leq j \leq q(|x|) + 1$ ,  $f_\tau[I_j]$  is not satisfied by  $S$ . For every  $j$ ,  $1 \leq j \leq q(|x|) + 1$ ,  $\beta_\tau[I_j](0, \dots, 0) = 1$ . So, for every  $j$ ,  $1 \leq j \leq q(|x|) + 1$ ,  $Q_\tau[I_j] \cap S \neq \emptyset$ . The intervals  $I_1, \dots, I_m$  are chosen so that  $Q_\tau[I_1], \dots, Q_\tau[I_m]$  are pairwise disjoint. Thus,

$$||S \cap \bigcup_{1 \leq j \leq q(|x|)+1} Q_\tau[I_j]|| \geq q(|x|) + 1.$$

This is a contradiction. So,  $w_{\max}(x) \geq u_{q(|x|)+1}$ . This implies that if  $\tau$  is correct, then all intervals  $I \in \Delta_1$  whose right end is strictly less than  $u_{q(|x|)+1}$  can be eliminated from  $\Delta_1$ . Hence,  $\Delta'_1$  is a refinement of  $\Delta_1$  for  $x$  under  $\tau$ . The rest of the procedure is essentially the same as that of Case 1. The only difference is that the number of selected intervals is at most  $q(|x|) + 1$ , and thus, the total number of refinements that are combined to form a refinement of  $\Delta'_1$  is at most  $2(k-d)(q(|x|)+1)$ . Let  $\Upsilon_1$  denote the union of the refinements obtained by the recursive calls.

The output  $\Upsilon$  of **CULL** is  $\Upsilon_0 \cup \Upsilon_1$ . Suppose that for each  $b \in \{0, 1\}$ ,  $\Upsilon_b$  is a refinement of  $\Delta_b$  for  $x$  under  $\tau$ . Since  $\Delta = \Delta_0 \cup \Delta_1$ , by part 1 of Fact 1.12,  $\Upsilon$  is a refinement of  $\Delta$  for  $x$  under  $\tau$ . The total number of recursive calls that **CULL** makes is  $2(k - |\tau|)(2q(|x|) + 1)$ .

**1.1.2.5 Correctness and Analysis of the Culling Method.** Since the depth of recursion is bounded by  $k$ , the entire culling procedure runs in time polynomial in  $|x|$ . The correctness of **CULL** can be proven by induction on the length of the hypothesis list, going down from  $k$  to 0. For the base case, let the length be  $k$ . We already showed that if the length of hypothesis list is  $k$  then **CULL** works correctly. For the induction step, let  $0 \leq d \leq k - 1$  and suppose that **CULL** works correctly in the case when the length of the hypothesis list is greater than or equal to  $d + 1$ . Suppose that  $(x, \Gamma, \tau)$  is given to **CULL** such that  $|\tau| = d$ . In each of the recursive calls that **CULL** makes on input  $(x, \Gamma, \tau)$ , the length of the hypothesis list is  $d + 1$ , so by the induction hypothesis, the output of each of the recursive calls is correct. This implies that the output of **CULL** on  $(x, \Gamma, \tau)$  is a refinement of  $\Gamma$ .

We also claim that, for every  $d$ ,  $0 \leq d \leq k$ , the number of intervals in the output of **CULL** in the case when the hypothesis list has length  $d$  is at most  $r_d(|x|)$ . Again, this is proven by induction on  $d$ , going down from  $k$  to 0. The claim certainly holds for the base case, i.e., when  $d = k$ , since the output of

**CULL** contains at most one interval when the hypothesis list has length  $k$  and  $r_k$  is the constant 1 function. For the induction step, let  $d = d_0$  for some  $d_0$ ,  $0 \leq d_0 \leq k - 1$ , and suppose that the claim holds for all values of  $|\tau|$  between  $d_0 + 1$  and  $k$ . Let  $(x, \Gamma, \tau)$  be an input to **CULL** such that  $|\tau| = d$ . The number of recursive calls that **CULL** on input  $(x, \Gamma, \tau)$  is at most

$$2((k - d)q(|x|) + 2(k - d)(q(|x|) + 1)) = 2(k - d)(2q(|x|) + 1).$$

In each of the recursive calls that are made, the length of the hypothesis list is  $d + 1$ , so by the induction hypothesis, the output of each recursive call has at most  $r_{d+1}(|x|)$  elements. So, the number of intervals in the output of **CULL** on input  $(x, \Gamma, \tau)$  is at most

$$2(k - d)(2q(|x|) + 1)r_{d+1}(2q(|x|) + 1).$$

This is  $r_d(|x|)$ . Thus, the claim holds for  $d$ .

Since  $r_0 = r$ , the number of intervals in the output of the culling phase is at most  $r(|x|)$  as desired. Hence,  $L \in P$ .  $\square$

## 1.2 The Turing Case

In the previous section, we saw that if any sparse set is NP-hard or NP-complete with respect to many-one reductions or even bounded-truth-table reductions, then  $P = NP$ . In this section, we ask whether any sparse set can be NP-hard or NP-complete with respect to Turing reductions. Since Turing reductions are more flexible than many-one reductions, this is a potentially weaker assumption than many-one completeness or many-one hardness. In fact, it remains an open question whether these hypotheses imply that  $P = NP$ , though there is some relativized evidence suggesting that such a strong conclusion is unlikely. However, one can achieve a weaker collapse of the polynomial hierarchy, and we do so in this section.

Unlike the previous section, the results and proofs for the  $\leq_T^P$ -completeness and  $\leq_T^P$ -hardness cases are quite different. We start with the  $\leq_T^P$ -complete case, which uses what is informally known as a “census” approach. The hallmark of algorithms based on the census approach is that they first obtain for a set (usually a sparse set) the exact number of elements that the set contains up to some given length, and then they exploit that information.

The  $\Theta_2^P$  level of the polynomial hierarchy (see Sect. A.4) captures the power of parallel access to NP. In particular, is known to equal the downward closure under truth-table reductions (see Sect. B.1) of NP; this closure is denoted (see the notational shorthands of Sect. B.2)  $R_{tt}^P(NP)$ . Thus, Theorem 1.14 proves that if NP has Turing-complete sparse sets, then the entire polynomial hierarchy can be accepted via parallel access to NP. However, the following equality is known to hold.



**Proposition 1.13**  $R_{tt}^p(\text{NP}) = P^{\text{NP}[\mathcal{O}(\log n)]}$ .

We will use the latter form of  $\Theta_2^p$  in the proof below.

**Theorem 1.14** *If NP has sparse, for NP  $\leq_T^p$ -complete sets then  $\text{PH} = \Theta_2^p$ .*

**Proof** Let  $S$  be a sparse set that is  $\leq_T^p$ -complete for NP. For each  $\ell$ , let  $p_\ell(n)$  denote  $n^\ell + \ell$ . Let  $j$  be such that  $(\forall n)[|S^{\leq n}| \leq p_j(n)]$ . Let  $M$  be a deterministic polynomial-time Turing machine such that  $\text{SAT} = L(M^S)$ ; such a machine must exist, as  $S$  is Turing-hard for NP. Let  $k$  be such that  $p_k(n)$  bounds the runtime of  $M$  regardless of the oracle  $M$  has; without loss of generality, let  $M$  be chosen so that such a  $k$  exists.

**Pause to Ponder 1.15** *Show why this “without loss of generality claim” holds.*

(Answer sketch for Pause to Ponder 1.15: Given a machine  $M$ , let the machines  $M_1, M_2, \dots$ , be as follows.  $M_i^A(x)$  will simulate the action of exactly  $p_i(|x|)$  steps of the action of  $M^A(x)$ , and then will halt in an accepting state if  $M^A(x)$  halted and accepted within  $p_i(|x|)$  steps, and otherwise will reject. Note that since the overhead involved in simulating one step of machine is at most polynomial, for each  $i$ , there will exist an  $\hat{i}$  such that for every  $A$  it holds that  $M_i^A$  runs in time at most  $p_{\hat{i}}(n)$ . Furthermore, in each relativized world  $A$  in which  $M^A$  runs in time at most  $p_i$ , it will hold that  $L(M^A) = L(M_i^A)$ . Relatedly, in our proof, given the machine  $M$  such that  $\text{SAT} = L(M^S)$ , we will in light of whatever polynomial-time bound  $M^S$  obeys similarly replace  $M$  with an appropriate  $M_j$  from the list of machines just described.)

Let  $L$  be an arbitrary set in  $\Sigma_2^p$ . Note that, since SAT is NP-complete, it is clear that  $\Sigma_2^p = \text{NP}^{\text{SAT}}$ . So, in particular, there is some nondeterministic polynomial-time Turing machine  $N$  such that  $L = L(N^{\text{SAT}})$ . Let  $\ell$  be such that  $p_\ell(n)$  bounds the nondeterministic runtime of  $N$  for all oracles; without loss of generality, let  $N$  be chosen such that such an integer exists (see Pause to Ponder 1.15). Note that  $L = L(N^{L(M^S)})$ .

Define:

$$V = \{0\#1^n\#1^q \mid |S^{\leq n}| \geq q\} \cup \\ \{1\#x\#1^n\#1^q \mid (\exists Z \subseteq S^{\leq n})[|Z| = q \wedge x \in L(N^{L(M^Z)})]\}.$$

Note that, in light of the fact that  $S$  is an NP set,  $V \in \text{NP}$ .

We now give a  $\Theta_2^p$  algorithm that accepts  $L$ . In particular, we give an algorithm that makes  $\mathcal{O}(\log n)$  calls to the NP oracle  $V$ . Suppose the input to our algorithm is the string  $y$ .

**Step 1** In  $\mathcal{O}(\log |y|)$  sequential queries to  $V$  determine  $|S^{\leq p_k(p_\ell(|y|))}|$ . Our queries will be of the form “ $0\#1^{p_k(p_\ell(|y|))}\#1^z$ ,” where we will vary  $z$  in a binary search fashion until we home in on the exact value of  $|S^{\leq p_k(p_\ell(|y|))}|$ . Since  $|S^{\leq p_k(p_\ell(|y|))}|$  is bounded by a polynomial in  $y$ , namely,

by  $p_j(p_k(p_\ell(|y|)))$ , it holds that  $\mathcal{O}(\log |y|)$  queries indeed suffice for binary search to pinpoint the census value. Let the census value obtained be denoted  $r$ .

**Step 2** Ask to  $V$  the query  $1\#y\#1^{p_\ell(p_k(|y|))}\#1^r$ , and accept if and only if the answer is that  $1\#y\#1^{p_\ell(p_k(|y|))}\#1^r \in V$ .

That completes the statement of the algorithm. Note that the algorithm clearly is a  $\Theta_2^P$  algorithm. Furthermore, note that the algorithm indeed accepts  $L$ . This is simply because, given that Step 1 obtains the true census  $r$ , the Step 2 query to  $V$  can accept only if the actual strings in  $S^{\leq p_k(p_\ell(|y|))}$  are guessed (because there are only  $r$  strings at those lengths, so if  $r$  distinct strings in  $S$  have been guessed, then we have guessed all of  $S^{\leq p_k(p_\ell(|y|))}$ ) and, when used by  $M$  to generate a prefix of SAT (and note that this prefix is correct on all queries to SAT of length at most  $p_\ell(|y|)$ , since such queries generate queries to  $S$  of length at most  $p_k(p_\ell(|y|))$ ), causes  $N$  to accept.

So, since  $L$  was an arbitrary set from  $\Sigma_2^P$ , we have  $\Sigma_2^P = \Theta_2^P$ . Since  $\Theta_2^P$  is closed under complementation, this implies  $\Sigma_2^P = \Pi_2^P$ , which itself implies  $\text{PH} = \Sigma_2^P$ . So  $\text{PH} = \Sigma_2^P = \Theta_2^P$ , completing our proof.  $\square$

The proof for the case of  $\leq_T^P$ -hardness is more difficult than the case of  $\leq_T^P$ -completeness, since the census proof used above crucially uses the fact that the sparse set is in NP. The proof below rolls out a different trick. It extensively uses nondeterminism to guess a set of strings that, while perhaps not the exact elements of a prefix of the sparse NP- $\leq_T^P$ -hard set, function just as usefully as such a prefix. The following result is often referred to as the Karp–Lipton Theorem.

**Theorem 1.16 (The Karp–Lipton Theorem)** *If NP has sparse  $\leq_T^P$ -hard sets then  $\text{PH} = \text{NP}^{\text{NP}}$ .*

**Proof** Let  $S$  be a sparse set that is  $\leq_T^P$ -hard for NP. For each  $\ell$ , let  $p_\ell(n)$  denote  $n^\ell + \ell$ . Let  $j$  be such that  $(\forall n)[|S^{\leq n}| \leq p_j(n)]$ . Let  $M$  be a deterministic polynomial-time Turing machine such that  $\text{SAT} = L(M^S)$ ; such a machine must exist, as  $S$  is Turing-hard for NP. Let  $k$  be such that  $p_k(n)$  bounds the runtime of  $M$  for all oracles; without loss of generality, let  $M$  be such that such an integer exists (see Pause to Ponder 1.15).

Let  $L$  be an arbitrary set in  $\Sigma_3^P$ . We will give a  $\Sigma_2^P$  algorithm for  $L$ . This establishes that  $\Sigma_2^P = \Sigma_3^P$ , which itself implies that  $\text{PH} = \Sigma_2^P$ , thus proving the theorem.

Note that, since SAT is NP-complete, it is clear that  $\Sigma_3^P = \text{NP}^{\text{NP}^{\text{SAT}}}$ . So, in particular, there are two nondeterministic polynomial-time Turing machines  $N_1$  and  $N_2$  such that  $L(N_1^{L(N_2^{\text{SAT}})}) = L$ . Let  $\ell$  be such that  $p_\ell(n)$  bounds the nondeterministic runtime of  $N_1$  for all oracles, and such that  $p_\ell(n)$  also bounds the nondeterministic runtime of  $N_2$  for all oracles; without loss of generality, let  $N_1$  and  $N_2$  be such that such an integer exists (see Pause to Ponder 1.15).

Define

$V_0 = \{0\#1^n\#S' \mid (\exists z \in (\Sigma^*)^{\leq n})[(a) \text{ } z \text{ is not a well-formed formula and } M^{S'}(z) \text{ accepts; or (b) } z \text{ is a well-formed formula free variables and either (b1) } M^{S'}(z) \text{ accepts and } z \notin \text{SAT or (b2) } M^{S'}(z) \text{ rejects and } z \in \text{SAT; or (c) } z \text{ is a well-formed formula variables } z_1, z_2, \dots \text{ and it is not the case that: } M^{S'}(z) \text{ accepts if and only if}$

$$(M^{S'}(z[z_1 = \text{True}]) \text{ accepts} \vee M^{S'}(z[z_1 = \text{False}]) \text{ accepts})\},$$

where, as defined earlier in this chapter,  $z[\dots]$  denotes  $z$  with the indicated variables assigned as noted.

$$V_1 = \{1\#S'\#z \mid z \in L(N_2^{L(M^{S'})})\}.$$

$$V = V_0 \cup V_1.$$

Note that  $V \in \text{NP}$ . Informally,  $V$  functions as follows. The  $0\#1^n\#S'$  strings in  $V$  determine whether given sets of strings “work” as sparse oracles that (on all instances of length at most  $n$ ) allow  $M$  to correctly accept SAT. Or, more exactly, it checks if a given set *fails* to simulate SAT correctly. Of course, the fact that  $S$  is a sparse Turing-hard set for NP ensures that there are some such sets  $S'$  that *do* simulate SAT correctly in this sense; however, it is possible that sets  $S'$  other than prefixes of  $S$  may also happen to simulate SAT correctly in this sense. The  $1\#\dots$  part of  $V$  takes a set of strings that happens to simulate SAT as just described, and uses them, in concert with  $M$ , to simulate SAT.

We now give a  $\text{NP}^{\text{NP}}$  algorithm that accepts  $L$ . In particular, we give an  $\text{NP}^V$  algorithm. Suppose the input to our algorithm is the string  $y$ . Note that the longest possible query to SAT that  $N_2$  will make on queries  $N_1$  asks to its oracle during the run of  $N_1^{L(N_2^{\text{SAT}})}(y)$  is  $p_\ell(p_\ell(|y|))$ . Note also that  $M$ , on inputs of length  $p_\ell(p_\ell(|y|))$ , asks its oracle only questions of length at most  $p_k(p_\ell(p_\ell(|y|)))$ . And finally, note that there is some sparse oracle  $U$  such that  $L(M^{(U^{\leq p_k(p_\ell(p_\ell(|y|)))})}) = \text{SAT}^{\leq p_\ell(p_\ell(|y|))}$ ; for example, the set  $S$  is such an oracle.

**Step 1** Nondeterministically guess a set  $S' \subseteq (\Sigma^*)^{\leq p_k(p_\ell(p_\ell(|y|)))}$  satisfying  $\|S'\| \leq p_j(p_k(p_\ell(p_\ell(|y|))))$ . If  $0\#1^{p_k(p_\ell(p_\ell(|y|)))}\#S' \in V$  then reject. Otherwise, go to Step 2.

**Step 2** Simulate the action of  $N_1(y)$  except that, each time  $N_1(y)$  makes a query  $z$  to its  $L(N_2^{\text{SAT}})$  oracle, ask instead the query  $1\#S'\#z$  to  $V$ .

That completes the description of the algorithm. Note that the algorithm we have given is clearly a  $\Sigma_2^P$  algorithm. Furthermore, note that the algorithm indeed accepts  $L$ . This is simply because Step 1 obtains a valid set of strings  $S'$  that either are  $S^{\leq p_k(p_\ell(p_\ell(|y|)))}$ , or that, in the action of machine  $M$ , function just as well as  $S^{\leq p_k(p_\ell(p_\ell(|y|)))}$  in simulating SAT. That is, we obtain a set of strings  $S'$  such that

$$\text{SAT}^{\leq p_k(p_\ell(p_\ell(|y|)))} = \left( L(M^{S'}) \right)^{\leq p_k(p_\ell(p_\ell(|y|)))}.$$

This correct prefix of SAT is just long enough that it ensures that Step 2 of the algorithm will correctly simulate  $N_2^{\text{SAT}}$ .  $\square$

This result has been extended in various ways. One very useful strengthening that we will refer to later is that one can replace the base-level NP machine with an expected-polynomial-time probabilistic machine. (The parenthetical equivalence comment in the theorem is based on the well-known fact, which is an easy exercise that we commend to the reader, that  $R_T^p(\{S \mid S \text{ is sparse}\}) = P/\text{poly.}$ )

**Theorem 1.17** *If NP has sparse  $\leq_T^p$ -hard sets (equivalently, if  $\text{NP} \subseteq P/\text{poly}$ ), then  $\text{PH} = \text{ZPP}^{\text{NP}}$ .*

### 1.3 The Case of Merely Putting Sparse Sets in NP – P: The Hartmanis–Immerman–Sewelson Encoding

In the previous sections we studied whether classes such as NP had complete or hard sparse sets with respect to various reductions. We know from Theorem 1.7, for example, that there is no NP-complete sparse set unless  $P = \text{NP}$ .

In this section, we ask whether there is any sparse set in  $\text{NP} - P$ . Note in particular that we are not here asking whether there is any sparse set in  $\text{NP} - P$  that is NP-complete; by Theorem 1.7 the answer to that question is clearly “no.” We here are instead merely asking whether any sparse set in NP can be so complex as to lack deterministic polynomial-time algorithms.

Before approaching this question, let us motivate it from a quite different direction. One central goal of computational complexity theory is to understand the relative power of different complexity classes. For example, is deterministic polynomial-time computation a strictly weaker notion than nondeterministic polynomial-time computation, that is  $P \neq \text{NP}$ ? The ideal results along such lines are results collapsing complexity classes or separating complexity classes.

In fact, complexity theorists have achieved a number of just such results—outright, nontrivial complexity class collapses and separations. For example, the strong exponential hierarchy—an exponential-time analog of the polynomial hierarchy—is known to collapse, and for very small space bounds a space analog of the polynomial hierarchy is known to truly separate. The famous time and space hierarchy theorems also provide unconditional separation results. Unfortunately, not one such result is known to be useful in the realm between P and PSPACE. It remains plausible that  $P = \text{PSPACE}$  and it remains plausible that  $P \neq \text{PSPACE}$ .

Given this disturbingly critical gap in our knowledge of the power of complexity classes between P and PSPACE—exactly the computational realm in

which most interesting real-world problems fall—what can be done? One approach is, instead of directly trying to separate or collapse the classes, to link the many open questions that exist within this range. The philosophy behind this is very similar to the philosophy behind NP-completeness theory. There, we still do not know whether NP-complete problems have polynomial-time algorithms. However, we do know that, since all NP-complete problems are  $\leq_m^P$ -interreducible, they stand or fall together; either all have polynomial-time algorithms or none do.

In the context of complexity classes between P and PSPACE, the goal along these lines would be to link together as many open questions as possible, ideally with “if and only if” links. It turns out that it often is easy to “upward translate” collapses, that is, to show that if small classes collapse then (seemingly) larger classes collapse. The truly difficult challenge is to “downward translate” equalities: to show that if larger classes collapse then (seemingly) smaller classes collapse.

In this section we study a famous downward translation that partially links the  $P = NP$  question to the collapse of exponential-time classes. In particular, we will ask whether the collapse of deterministic and nondeterministic exponential time implies any collapse in the classes between P and PSPACE. The really blockbuster result to seek would be a theorem establishing that  $E = NE \implies P = NP$ . However, it is an open question whether this can be established. What is known, and what we will here prove, is the following theorem, which says that the collapse of NE to E is equivalent to putting into P all sparse sets in NP.

**Theorem 1.18** *The following are equivalent:*

1.  $E = NE$ .
2.  $NP - P$  contains no sparse sets.
3.  $NP - P$  contains no tally sets.

**Proof** Part 2 clearly implies part 3, as every tally set is sparse. The theorem follows immediately from this fact, and from Lemmas 1.19 and Lemma 1.21.  $\square$

The following easy lemma shows that if no tally sets exist in  $NP - P$ , then NE collapses to E.

**Lemma 1.19** *If  $NP - P$  contains no tally sets then  $E = NE$ .*

**Proof** Let  $L$  be some set in NE, and assume that  $NP - P$  contains no tally sets. Let  $N$  be a nondeterministic exponential-time machine such that  $L(N) = L$ . Define  $L' = \{1^k \mid (\exists x \in L)[k = (1x)_{\text{bin}}]\}$ , where for any string (over  $\{0,1\}$ )  $z$  the expression  $(z)_{\text{bin}}$  denotes the integer the string represents when viewed as a binary integer, e.g.,  $(1000)_{\text{bin}} = 8$ .

Note that  $L' \in NP$ , since the following algorithm accepts  $L'$ . On input  $y$ , reject if  $y$  is not of the form  $1^k$  for some  $k > 0$ . Otherwise  $y = 1^k$  for some

$k > 0$ . Write  $k$  in binary, and let  $s$  be the binary of representation of  $k$  to the right of, and not including, its leftmost one, viewed as a binary string. Call this string  $w$ . (If  $k = 1$ , then  $w = \epsilon$ .) Simulate  $N(w)$  (thus accepting if and only if  $N(w)$  accepts). Though  $N$  is an exponential-time machine, the length of  $w$  is logarithmic in the length of  $y$ , and thus the overall nondeterministic runtime of this algorithm is, for some constant  $c$ , at most  $O(2^{c \log n})$ .

Thus,  $L' \in \text{NP}$ . However, by hypothesis this implies that  $L'$  is in  $\text{P}$ . So, let  $M$  be a deterministic polynomial-time machine such that  $L(M) = L'$ . We now describe a deterministic exponential-time algorithm for  $L$ . On input  $a$ , compute the string  $b = 1^{(1^a)_{\text{bin}}}$ , and then simulate  $M(b)$ , accepting if and only if  $M(b)$  accepts. Since  $M$  is a polynomial-time machine and  $|b| \leq 2^{|a|}$ , the number of steps that  $M(b)$  runs is  $(2^n)^c = 2^{cn}$ . As the overhead of doing the simulation and the cost of obtaining  $b$  from  $a$  are also at most exponential in the input's length, clearly our algorithm for  $L$  is a deterministic exponential-time algorithm. Thus,  $L \in \text{E}$ , which completes our proof.  $\square$

Finally, we must prove that if  $\text{E} = \text{NE}$  then all sparse NP sets in fact are in  $\text{P}$ .

**Pause to Ponder 1.20** *As an easy warmup exercise, try to prove the simpler claim: If  $\text{E} = \text{NE}$  then all tally sets in NP are in P.*

A sketch of the solution to Pause to Ponder 1.20 is as follows. If  $L$  is a tally set in NP, then let  $L' = \{x \mid (x \text{ is } 0 \text{ or } x \text{ is a binary string of nonzero length with no leading zeros}) \text{ and } 1^{(x)_{\text{bin}}} \in L\}$ . It is not hard to see that  $L' \in \text{NE}$ . Thus by assumption  $L' \in \text{E}$ , and thus there is a natural  $\text{P}$  algorithm for  $L$ , namely, the algorithm that on input  $a$  rejects if  $a \notin 1^*$  and that if  $a = 1^k$  writes  $k$  as 0 if  $k = 0$  and otherwise as  $k$  in binary with no leading zeros, and then simulates the  $\text{E}$  algorithm for  $L'$  on this string. This concludes the proof sketch for Pause to Ponder 1.20.

However, recall that we must prove the stronger result that if  $\text{E} = \text{NE}$  then all sparse NP sets are in  $\text{P}$ . Historically, the result in Pause to Ponder 1.20 was established many years before this stronger result. If one looks carefully at the proof just sketched for Pause to Ponder 1.20, it is clear that the proof, even though it works well for the stated case (tally sets), breaks down catastrophically for sparse sets. The reason it fails to apply to sparse sets is that the proof is crucially using the fact that the length of a string in a tally set fully determines the string. In a sparse set there may be a polynomial number of strings at a given length. Thus the very, very simple encoding used in the proof sketch of Pause to Ponder 1.20, namely, representing tally strings by their length, is not powerful enough to distinguish same-length strings in a sparse set.

To do so, we will define a special “Hartmanis–Immerman–Sewelson encoding set” that crushes the information of any sparse NP set into extremely bite-sized morsels from which the membership of the set can be easily reconstructed. In fact, the encoding manages to put all useful information about a

sparse NP set's length  $n$  strings into length- $\mathcal{O}(\log n)$  instances of the encoding set—and yet maintain the easy decodability that is required to establish the following lemma.

**Lemma 1.21** *If  $E = NE$  then  $NP - P$  contains no sparse sets.*

**Proof** Let  $L$  be some sparse NP set, and assume that  $E = NE$ . Given that  $L$  is sparse, there is some polynomial, call it  $q$ , such that  $(\forall n)[|L^n| \leq q(n)]$ . Define the following encoding set:

$$L' = \{0\#n\#k \mid |L^n| \geq k\} \cup \\ \{1\#n\#c\#i\#j \mid (\exists z_1, z_2, \dots, z_c \in L^n)[z_1 <_{\text{lex}} z_2 <_{\text{lex}} \dots <_{\text{lex}} z_c \wedge \text{the } j\text{th bit of } z_i \text{ is } 1]\}.$$

Since  $L \in NP$ , it is clear that  $L' \in NE$ . So by our assumption,  $L' \in E$ .

We will now use the fact that  $L' \in E$  to give a P algorithm for  $L$ . Our P algorithm for  $L$  works as follows. On input  $x$ , let  $n = |x|$ . Query  $L'$  to determine which of the following list of polynomially many strings belong to  $L'$ :  $0\#n\#0, 0\#n\#1, 0\#n\#2, \dots, 0\#n\#q(n)$ , where here and later in the proof the actual calls to  $L'$  will for the numerical arguments (the  $n$ 's,  $c$ ,  $i$ ,  $j$ , and  $k$  of the definition of  $L'$ ) be coded as (and within  $L'$  will be decoded back from) binary strings. Given these answers, set

$$c = \max\{k \mid 0 \leq k \leq q(n) \wedge 0\#n\#k \in L'\}.$$

Note that  $c = |L^n|$ . Now ask the following questions to  $L'$ :

$$\begin{aligned} &1\#n\#c\#1\#1, 1\#n\#c\#1\#2, \dots, 1\#n\#c\#1\#n, \\ &1\#n\#c\#2\#1, 1\#n\#c\#2\#2, \dots, 1\#n\#c\#2\#n, \\ &\quad \dots, \\ &1\#n\#c\#c\#1, 1\#n\#c\#c\#2, \dots, 1\#n\#c\#c\#n. \end{aligned}$$

The answers to this list of polynomially many questions to  $L'$  give, bit by bit, the entire set of length  $n$  strings in  $L$ . If our input,  $x$ , belongs to this set then accept, and otherwise reject. Though  $L' \in E$ , each of the polynomially many queries asked to  $L'$  (during the execution of the algorithm just described) is of length  $\mathcal{O}(\log n)$ . Thus, it is clear that the algorithm is indeed a polynomial-time algorithm.  $\square$

Theorem 1.18 was but the start of a long line of research into downward translations. Though the full line of research is beyond the scope of this book, and is still a subject of active research and advances, it is now known that the query hierarchy to NP itself shows certain downward translations of equality. In particular, the following result says that if one and two questions to  $\Sigma_k^P$  yield the same power, then the polynomial hierarchy collapses not just to  $P^{\Sigma_k^{P[1]}}$  but in fact even to  $\Sigma_k^P$  itself.

**Theorem 1.22** *Let  $k > 1$ .  $\Sigma_k^P = \Pi_k^P$  if and only if  $P^{\Sigma_k^{P[1]}} = P^{\Sigma_k^{P[2]}}$ .*

## 1.4 OPEN ISSUE: Does the Disjunctive Case Hold?

Theorem 1.7 shows that NP lacks sparse  $\leq_m^P$ -complete sets unless  $P = NP$ . Does this result generalize to bounded-truth-table, conjunctive-truth-table, and disjunctive-truth-table reductions:  $\leq_{btt}^P$ ,  $\leq_{ctt}^P$ , and  $\leq_{dtt}^P$ ?

Theorem 1.10 already generalizes Theorem 1.7 to the case of  $\leq_{btt}^P$ -hardness. Using the left set technique it is also easy to generalize the result to the case of  $\leq_{ctt}^P$ -hardness: If NP has  $\leq_{ctt}^P$ -hard sparse sets then  $P = NP$ .

The case of  $\leq_{dtt}^P$ -hardness remains very much open.

**Open Question 1.23** *Can one prove: If NP has  $\leq_{dtt}^P$ -hard sparse sets, then  $P = NP$ ?*

However, it is known that proving the statement would be quite strong. In particular, the following somewhat surprising relationship is known.

**Proposition 1.24** *Every set that  $\leq_{btt}^P$ -reduces to a sparse set in fact  $\leq_{dtt}^P$ -reduces to some sparse set.*

Thus, if one could prove the result of Open Question 1.23, that result would immediately imply Theorem 1.10.

## 1.5 Bibliographic Notes

Theorem 1.2 (which is often referred to as “Berman’s Theorem”) and Corollary 1.3 are due to Berman [Ber78], and the proof approach yields the analog of these results not just for the tally sets but also for the P-capturable [CGH<sup>+</sup>89] sets, i.e., the sets having sparse P supersets. Theorem 1.4 and Corollary 1.5 are due to Fortune [For79]. Among the results that followed soon after the work of Fortune were advances by Ukkonen [Ukk83], Yap [Yap83], and Yesha [Yes83].

Theorems 1.7 (which is known as “Mahaney’s Theorem”) and Theorem 1.9 are due to Mahaney [Mah82]. The historical motivation for his work is sometimes forgotten, but is quite interesting. The famous Berman–Hartmanis Isomorphism Conjecture [BH77], which conjectures that all NP-complete sets are polynomial-time isomorphic, was relatively new at the time. Since no dense set (such as the NP-complete set SAT) can be polynomial-time isomorphic to any sparse set, the existence of a sparse NP-complete set would immediately show the conjecture to be false. Thus, Mahaney’s work was a way of showing the pointlessness of that line of attack on the Berman–Hartmanis Isomorphism Conjecture (see [HM80]): if such a set exists, then  $P = NP$ , in which case the Berman–Hartmanis Isomorphism Conjecture fails trivially anyway.

Theorem 1.10 (which is often referred to as “the Ogiwara–Watanabe Theorem”) is due to Ogiwara and Watanabe ([OW91], see also [HL94]),



who introduced the left set technique in the study of sparse complete sets. Somewhat more general results than Theorem 1.10 are now known to hold, due to work of Homer and Longpré [HL94], Arvind et al. [AHH<sup>+</sup>93], and Glaßer ([Gla00], see also [GH00]). Our presentation is based on the work of Homer and Longpré [HL94].

These results are part of a rich exploration of sparse completeness results, with respect to many complexity classes and many types of reductions, that started with Berman's work and that continues to this day. Numerous surveys of general or specialized work on sparse complete sets exist [HM80, Mah86, Mah89, You92, HOW92, vMO97, CO97, GH00].

Regarding the relativized evidence mentioned on page 18, Immerman and Mahaney [IM89] have shown that there are relativized worlds in which NP has sparse Turing-hard sets yet  $P \neq NP$ . Arvind et al. [AHH<sup>+</sup>93] extended this to show that there are relativized worlds in which NP has sparse Turing-complete sets yet the boolean hierarchy [CGH<sup>+</sup>88] does not collapse, and Kadin [Kad89] showed that there are relativized worlds in which NP has sparse Turing-complete sets yet some  $\Theta_2^P$  languages cannot be accepted via P machines making  $o(\log n)$  sequential queries to NP.

Proposition 1.13 is due to Hemachandra [Hem89]. Theorem 1.14 is due to Kadin [Kad89]. Theorem 1.16 is due to Karp and Lipton [KL80], and we prove it here using a nice, intuitive, alternate proof line first suggested by Hopcroft ([Hop81], see also [BBS86]). The fact that  $R_T^P(\{S \mid S \text{ is sparse}\}) = P/\text{poly}$  appears in a paper by Berman and Hartmanis [BH77], where it is attributed to Meyer. Theorem 1.17 is due to Köbler and Watanabe ([KW98], see also [KS97]) based on work of Bshouty et al. [BCKT94, BCG<sup>+</sup>96].

Cai [Cai01] has proven that the “symmetric alternation” version of  $NP^{NP}$ , a class known as  $S_2^P$  [Can96, RS98], satisfies  $S_2^P \subseteq ZPP^{NP}$ . In light of Sengupta's observation (see the discussion in [Cai01]) that a Hopcroft-approach proof of Theorem 1.16 in fact can be used to conclude that  $S_2^P = PH$ , Cai's result says that Sengupta's collapse to  $S_2^P$  is at least as strong as, and potentially is even stronger than, that of Theorem 1.16.

The collapse of the strong exponential-time hierarchy referred to near the start of Sect. 1.3 is due to Hemachandra [Hem89], and the separation of small-space alternation hierarchies referred to in Sect. 1.3 is due, independently (see [Wag93]), to Liśkiewicz and Reischuk [LR96, LR97], von Braunmühl, Gengler, and Rettinger [vBGR93, vBGR94], and Geffert [Gef94]. The study of time and space hierarchy theorems is a rich one, and dates back to the pathbreaking work of Hartmanis, Lewis, and Stearns [HS65, LSH65, SHL65].

Lemma 1.19 and the result stated in Pause to Ponder 1.20—and thus the equivalence of parts 1 and 3 of Theorem 1.18—are due to Book [Boo74b].

The Hartmanis–Immerman–Sewelson Encoding, and in particular Lemma 1.21 (and thus in effect the equivalence of parts 1 and 2 of Theorem 1.18), was first employed by Hartmanis [Har83]. The technique was further explored by Hartmanis, Immerman, and Sewelson ([HIS85], see

also [All91,AW90]). Even the Hartmanis–Immerman–Sewelson Encoding has its limitations. Though it does prove that  $E = NE$  if and only if  $NP - P$  has sparse sets, it does not seem to suffice if we shift our attention from  $NP$  (and its exponential analog,  $NE$ ) to  $UP$ ,  $FewP$ ,  $\oplus P$ ,  $ZPP$ ,  $RP$ , and  $BPP$  (and their respective exponential analogs). In fact, the Buhrman–Hemaspaandra–Longpré Encoding [BHL95], a different, later encoding encoding based on some elegant combinatorics [EFF82,EFF85,NW94], has been used by Rao, Rothe, and Watanabe [RRW94] to show that the  $\oplus P$  and “ $FewP$ ” analogs of Theorem 1.18 do hold. That is, they for example prove that  $E$  equals, “ $\oplus E$ ,” the exponential-time analog of  $\oplus P$ , if and only if  $\oplus P - P$  contains sparse sets. In contrast with this, Hartmanis, Immerman, and Sewelson showed that there are oracles relative to which the  $coNP$  analog of Theorem 1.18 fails. Hemaspaandra and Jha [HJ95a] showed that there are oracles relative to which the  $ZPP$ ,  $R$ , and  $BPP$  analogs of Theorem 1.18 fail, and they also showed that even for the  $NP$  case the “immunity” analog of Theorem 1.18 fails. Allender and Wilson [All91,AW90] have shown that one claimed “supersparse” analog of Theorem 1.18 fails, but that in fact certain analogs can be obtained. For some classes, for example  $UP$ , it remains an open question whether an analog of Theorem 1.18 can be obtained.

The proof of Lemma 1.21 proves something a bit stronger than what the lemma itself asserts. In particular, the proof makes it clear that: If  $E = NE$  then every sparse  $NP$  set is  $P$ -printable (i.e., there is an algorithm that on input  $1^n$  prints *all* length  $n$  strings in the given sparse  $NP$  set). This stronger claim is due to Hartmanis and Yesha [HY84].

Regarding downward translations of equality relating exponential-time classes to smaller classes, we mention that a truly striking result of Babai, Fortnow, Nisan, and Wigderson [BFNW93] shows: If a certain exponential-time analog of the polynomial hierarchy collapses to  $E$ , then  $P = BPP$ . This is not quite a “downward” translation of equality, as it is not clear in general whether  $BPP \subseteq E$  (though that does hold under the hypothesis of their theorem, due to the conclusion of their theorem), but this result nonetheless represents a remarkable connection between exponential-time classes and polynomial-time classes.

A  $\Sigma_k^P = \Pi_k^P$  conclusion, and thus a downward translation of equality for classes in the  $NP$  query hierarchy, was reached by Hemaspaandra, Hemaspaandra, and Hempel [HHH99a] for the case  $k > 2$ . Buhrman and Fortnow [BF99] extended their result to the  $k = 2$  case. These appear as Theorem 1.22. Downward translations of equality are known not just for the 1-vs-2 query case but also for the  $j$ -vs- $(j+1)$  query case ([HHH99a,HHH99b], see also [HHH98]), but they involve equality translations within the bounded-access-to- $\Sigma_k^P$  hierarchies, rather than equality translations to  $\Sigma_k^P = \Pi_k^P$ .

In contrast with the difficulty of proving downward translations of equality, upward translations of equality are so routine that they are considered by most people to be “normal behavior.” For example, it is well-known for

almost all pairs of levels of the polynomial hierarchy that if the levels are equal then the polynomial hierarchy collapses. This result dates back to the seminal work of Meyer and Stockmeyer, who defined the polynomial hierarchy [MS72, Sto76]. The fascinating exception is whether  $\Theta_k^p = \Delta_k^p$  implies that the polynomial hierarchy collapses. Despite intense study, this issue remains open—see the discussion in [Hem94, HRZ95].

Nonetheless, it is far from clear that the view that upward translation of equality is a “normal” behavior of complexity classes is in itself a correct view. It does tend to hold within the polynomial hierarchy, which is where the intuition of most complexity theorists has been forged, but the polynomial hierarchy has many peculiar properties that even its close cousins lack (stemming from such features as the fact that the set of all polynomials happens to be closed under composition—in contrast to the set of logarithmic functions or the set of exponential functions), and thus is far from an ideal basis for predictions. In fact, Hartmanis, Immerman, and Sewelson [HIS85] and Impagliazzo and Tardos [IT89] have shown that there is an oracle relative to which upward translation of equality fails in an exponential-time analog of the polynomial hierarchy, and Hemaspaandra and Jha ([HJ95a], see also [BG98]) have shown the same for the limited-nondeterminism hierarchy of NP—the so-called  $\beta$  hierarchy of Kintala and Fischer [KF80] and Díaz and Torán [DT90].

Proposition 1.24 is due to Allender et al. [AHOW92]. Though Open Question 1.23, with its  $P = NP$  conjectured conclusion from the assumption of there being sparse  $\leq_{dt}^p$ -hard sets for NP, indeed remains open, some consequences—though not as strong as  $P = NP$ —are known to follow from the existence of sparse  $\leq_{dt}^p$ -hard sets for NP. In particular, Cai, Naik, and Sivakumar have shown that if NP has sparse  $\leq_{dt}^p$ -hard sets then  $RP = NP$  [CNS96]. It is also known that if NP has sparse  $\leq_{dt}^p$ -hard sets, then there is a polynomial-time set  $A$  such that every unsatisfiable boolean formula belongs to  $\bar{A}$  and every boolean formula that has exactly one satisfying assignment belongs to  $A$  (implicit in [CNS95], as noted by van Melkebeek [vM97] and Sivakumar [Siv00]). That is,  $A$  correctly solves SAT on all inputs having at most one solution, but might not accept some satisfiable formulas having more than one satisfying assignment.



## 2. The One-Way Function Technique

No proven one-way functions are known. Not even one. Nonetheless, one-way functions play a central role in complexity theory and cryptography. In complexity theory, one-way functions have been used in the (to date unsuccessful) attempts to show that there exist two NP-complete sets that are not essentially the same set in disguise (i.e., that are not polynomial-time isomorphic). In average-case and worst-case cryptography, one-way functions have been used as key components in the construction of cryptographic protocols.

Fortunately, the comment made above about not even one one-way function being known is, though true, a bit deceptive. Good candidates for being one-way functions are known. In fact, complete characterizations now exist regarding the existence of one-way functions. In particular, it is now known that the type of one-way function used in average-case cryptography exists if and only if pseudorandom generators exist. It is also known that the type of one-way function used in both computational complexity theory and worst-case cryptography exists if and only if two well-known complexity classes differ.

This chapter's GEM section proves the latter result. We will see in that section that one way in which one-way functions are classified is in terms of their level of many-to-one-ness—what types of collision are allowed. Sect. 2.2 proves the rather remarkable result that one-to-one one-way functions exist if and only if constant-to-one one-way functions exist. Though this does not say that low-collision-intensity one-way-function classes all collapse to mutual equality, it does say that their existence stands or falls together. Section 2.3 looks at two-argument one-way functions and in particular at the extremely powerful (associative, commutative, total, and strongly noninvertible) types of one-way functions that have been supposed-to-exist and used as hypothetical building blocks for protocols in worst-case cryptography. We prove that they are just as likely to exist as are standard one-way functions. So, one might as well feel free to use these “killer” building blocks, as their existence stands or falls together with the existence of standard building blocks.

## 2.1 GEM: Characterizing the Existence of One-Way Functions

Informally put, a one-way function is a function that is easy to compute and hard to invert. However, to be able to rigorously characterize whether one-way functions exist, we will have to formally pin down each of these notions. In addition, we will require a technical condition known as “honesty,” which is needed to keep the entire discussion from being trivialized. Also, the functions discussed in the definitions and theorems of this section and Sect. 2.2 are one-argument functions, that is, their type is  $f : \Sigma^* \rightarrow \Sigma^*$ ; in Sect. 2.3, we will extend the notion of one-way function to the case of two-argument functions.

**Definition 2.1** *We say a (possibly nontotal) function  $f$ , is honest if*

$$(\exists \text{ polynomial } q)(\forall y \in \text{range}(f))(\exists x)[|x| \leq q(|y|) \wedge f(x) = y].$$

**Definition 2.2** *We say a (possibly nontotal) function  $f$  is polynomial-time invertible if there is a (possibly nontotal) polynomial-time computable function  $g$  such that<sup>1</sup>*

$$(\forall y \in \text{range}(f))[y \in \text{domain}(g) \wedge g(y) \in \text{domain}(f) \wedge f(g(y)) = y].$$

**Definition 2.3** *We say a (possibly nontotal) function  $f$  is one-way if*

1.  $f$  is polynomial-time computable,
2.  $f$  is not polynomial-time invertible, and
3.  $f$  is honest.

Let us see why the honesty condition is natural and needed. Consider the function  $f(x) = 1^{\lceil \log \log \log(\max\{|x|, 4\}) \rceil}$ , that is, a string of  $\lceil \log \log \log(\max\{|x|, 4\}) \rceil$  ones. This function’s outputs are so short relative to its inputs that, simply to have enough time to write down an inverse, any machine inverting  $f$  must take triple exponential time. Thus,  $f$  is a polynomial-time computable function that is not polynomial-time invertible. However, this noninvertibility is simply an artifact of the dramatically length-decreasing nature of  $f$ . As this type of length-trick noninvertibility is of no help at all in cryptography or complexity theory, we preclude it by putting the honesty condition into our definition of a one-way function.

The honesty condition says that each element  $y$  of the range of  $f$  has some inverse whose length is at most polynomially longer than the length of  $y$ . So, for honest functions  $f$ ,  $f$  does have short inverses, and if  $f$  is not

<sup>1</sup> The “ $\wedge$ ” here is a bit subtle, since if “ $y \in \text{domain}(g)$ ” does not hold, the expression “ $g(y) \in \text{domain}(f) \wedge f(g(y)) = y$ ” isn’t even meaningful. In fact, the “ $\wedge$ ” is really a “cand” (a “conditional and”)—an “ $\wedge$ ” such that the right-hand side is evaluated only if the left-hand side evaluates to “true.” However, since this type of thing, here and elsewhere, is clear from context, we use “ $\wedge$ ” as our notation.

polynomial-time invertible the reason is not a length trick, but rather reflects our intuitive notion of what noninvertibility should mean.

Of course, having defined one-way functions, the natural question that immediately arises is whether one-way functions exist. This question is one of the major open issues in complexity theory and worst-case cryptography. However, it is not a new open issue, but rather is a familiar issue in disguise. In particular, Theorem 2.5 proves that this question in fact is a restatement of the famous “ $P \neq NP$ ?” question, and for one-to-one functions it is a restatement of the question “ $P \neq UP$ ?”

The reason the one-to-one case is often studied is that it corresponds to the case where each encrypted message has at most one possible decoding—an obviously desirable situation.

**Definition 2.4** *We say a (possibly nontotal) function  $f : \Sigma^* \rightarrow \Sigma^*$  is one-to-one if  $(\forall y \in \Sigma^*)[|\{x \mid f(x) = y\}| \leq 1]$ .*

### Theorem 2.5

1. *One-way functions exist if and only if  $P \neq NP$ .*
2. *One-to-one one-way functions exist if and only if  $P \neq UP$ .*

**Proof** We first prove part 1.

Let us start with the “if” direction. Assume that  $P \neq NP$ . Let  $A$  be a language in  $NP - P$ . So, it certainly holds that there will exist an NPTM (nondeterministic polynomial-time Turing machine)  $N$  such that  $A = L(N)$ . We assume  $\langle \cdot, \cdot \rangle$  is some standard pairing function (i.e., a bijection between  $\Sigma^* \times \Sigma^*$  and  $\Sigma^*$  that is polynomial-time computable and polynomial-time invertible). Consider the function  $f(\langle x, w \rangle)$  that outputs  $0x$  if  $w$  is an accepting path of  $N(x)$  and that outputs  $1x$  otherwise. This function is clearly polynomial-time computable and honest (the polynomial-time computability and invertibility of the pairing function block it from so severely distorting lengths as to destroy honesty). Suppose that  $f$  were polynomial-time invertible, via function  $g$ . Then we have that  $A \in P$ , as shown by the following algorithm. On any input  $y$ , if  $0y \notin \text{domain}(g)$  then reject  $y$ . Otherwise, interpret  $g(0y)$  as a pair and test whether its second component is an accepting path of  $N(y)$ . If so then accept and otherwise reject. However, as we assumed that  $A \notin P$ ,  $A \in P$  is a contradiction, and so our supposition that  $f$  is invertible must be incorrect. So,  $f$  is a polynomial-time computable, honest function that is not polynomial-time invertible. That is, it is a one-way function.

We now turn to the “only if” direction of part 1. Assume that one-way functions exist and that  $f$  is a one-way function. Let  $p$  be the honesty polynomial for  $f$ , in the sense of Definition 2.1. Consider the following language.

$$L = \{\langle z, pre \rangle \mid (\exists y)[|y| + |pre| \leq p(|z|) \wedge f(pre \cdot y) = z]\},$$

where “ $\cdot$ ” denotes string concatenation. Clearly,  $L \in NP$ . However, if  $L \in P$  we can invert  $f$  by prefix search. That is, to invert  $z$  we ask first “ $\langle z, \epsilon \rangle \in L$ ?”

If the answer is no, then  $z$  is not in the range of  $f$ . If the answer is yes, we check whether  $f(\epsilon) = z$  and if so we have found a preimage under  $f$  of  $z$  as desired, and otherwise we ask each of the questions “ $\langle z, 0 \rangle \in L$ ?” and “ $\langle z, 1 \rangle \in L$ ?” At least one must receive the answer yes. We now proceed as above to check whether that one is the inverse. If so we are done, and if not we expand one bit further (if both receive the answer yes, we choose either one, but not both, to expand further) and continue similarly. For strings  $z$  that are in the range of  $f$ , we will find a certificate in polynomial time, as with each pair of questions we get one additional bit of a certificate, and the certificates are of length at most  $p(|z|)$ . We conclude that  $L \in \text{NP} - \text{P}$ . Thus we have shown the “only if” direction of part 1.

The proof of part 2 is almost the same as the above, except we require unambiguity of the NP machines and we add unambiguity to the inverses of the functions. To make this work requires some minor proof tweaking; for completeness, we include a proof of the changed part. However, we urge the reader to see the proof not just by reading it below but rather by first him- or herself checking that the previous proof with minor modification does cover this case.

We prove part 2’s “if” direction. Assume that  $\text{P} \neq \text{UP}$ . Let  $A$  be a language in  $\text{UP} - \text{P}$ . So, it certainly holds that there will exist an NPTM  $N$  such that  $A = L(N)$  and such that on each input  $x$ ,  $N(x)$  has at most one accepting computation path. Let  $p$  be a polynomial bounding the runtime of  $N$ . Consider the function  $f(\langle x, w \rangle)$  that outputs  $0x$  if  $w$  is an accepting path of  $N(x)$  and that outputs  $1\langle x, w \rangle$  otherwise. This function is clearly polynomial-time computable and honest. Since  $N$  has at most one accepting path per input, and since we have modified  $f$  to now send the nonaccepting witnesses to distinct locations,  $f$  is a one-to-one function. However, if  $f$  were polynomial-time invertible then by the same algorithm given in the case of part 1, we would conclude that  $A \in \text{P}$ , yielding a contradiction. So,  $f$  is a polynomial-time computable, honest, one-to-one function that is not polynomial-time invertible. That is, it is a one-to-one one-way function.

We now turn to the “only if” direction of part 2. Assume that one-to-one one-way functions exist and that  $f$  is such a function. The language  $L$  constructed in the proof of part 1 will in fact be in UP due to  $f$ ’s one-to-oneness, and the rest of the “only if” direction of part 1 holds without change, thus completing our proof.  $\square$

The proof of Theorem 2.5 is a model of the one-way function technique, namely, *proving results via the connection between certificates for machine acceptance and the invertibility of functions*. We will draw on this approach both in Sect. 2.2, in the form of Fact 2.9, and in Sect. 2.3, where the main proof is also an example of the one-way function technique.



## 2.2 Unambiguous One-Way Functions Exist If and Only If Bounded-Ambiguity One-Way Functions Exist

In the GEM section, we saw that the existence of one-to-one one-way functions is characterized by  $P \neq UP$ , but the existence of one-way functions is characterized by the less demanding condition  $P \neq NP$ . Though  $P \neq UP$  implies  $P \neq NP$ , the converse has never been established. That is, it is at least plausible that one-way functions exist but that no one-to-one one-way functions exist.

In contrast, we will now prove that for a certain narrower gap in allowed amount of many-to-one-ness, the existence of one-way function at the two levels of many-to-one-ness stands or falls together. In particular, we prove that one-to-one one-way functions exist if and only if constant-to-one one-way functions exist.

**Definition 2.6** 1. For each  $k \geq 1$ , we say that a (possibly nontotal) function  $f$  is  $k$ -to-one if  $(\forall y \in \text{range}(f)) |\{x \mid f(x) = y\}| \leq k$ .  
 2. We say that a (possibly nontotal) function  $f$  is of bounded-ambiguity if there is a  $k \geq 1$  such that  $f$  is  $k$ -to-one.

Note that “1-to-one” is synonymous with “one-to-one” (Definition 2.4), and so we will use the terms interchangeably. Note that such functions  $f$  are completely *unambiguous* in terms of inversion; each element of  $\text{range}(f)$  has exactly one inverse. In the literature, bounded-ambiguity functions are often referred to as “constant-to-one” or “ $O(1)$ -to-one” functions.

**Theorem 2.7** *Unambiguous (i.e., one-to-one) one-way functions exist if and only if bounded-ambiguity one-way functions exist.*

**Proof** All one-to-one functions are bounded-ambiguity functions, so the “only if” direction holds.

We will prove the “if” direction somewhat indirectly. Recall that part 2 of Theorem 2.5 shows that one-to-one one-way functions exist if and only if  $P \neq UP$ . By an exactly analogous proof, we have Fact 2.9.

**Definition 2.8** A language  $L$  is in  $UP_{\leq k}$ ,  $k \geq 1$ , if there is an NPTM  $N$  such that

1.  $(\forall x \in L)[N(x) \text{ has at least one and at most } k \text{ accepting paths}]$ , and
2.  $(\forall x \in \bar{L})[N(x) \text{ has no accepting paths}]$ .

**Fact 2.9** *For each  $k \geq 2$ ,  $k$ -to-one one-way functions exist if and only if  $P \neq UP_{\leq k}$ .*

Our attack on the “if” direction of Theorem 2.7 will be to prove by induction that, for all  $k \in \{1, 2, 3, \dots\}$ ,

$$P = UP \implies P = UP_{\leq k}.$$

This clearly holds for  $k = 1$ , as  $\text{UP} = \text{UP}_{\leq 1}$ .

Assume, inductively, that we have proven  $\text{P} = \text{UP} \implies \text{P} = \text{UP}_{\leq k'}$ . We will now show that  $\text{P} = \text{UP} \implies \text{P} = \text{UP}_{\leq k'+1}$ . So, assume  $\text{P} = \text{UP}$ . Let  $L$  be an arbitrary member of  $\text{UP}_{\leq k'+1}$ . Let  $N$  be an NPTM—having at most  $k' + 1$  accepting paths on each input—that accepts  $L$  (in the sense of Definition 2.8). Consider the set

$$B = \{x \mid N(x) \text{ has exactly } k' + 1 \text{ accepting paths}\}.$$

Clearly,  $B \in \text{UP}$ , via the machine that on each input  $x$  guesses each lexicographically ordered  $(k' + 1)$ -tuple of distinct computation paths and that accepts on such a path exactly if each of the  $k' + 1$  guessed paths is an accepting path on input  $x$ . So by our  $\text{P} = \text{UP}$  assumption,  $B \in \text{P}$ .

However, since  $B \in \text{P}$ , the set

$$D = \{x \mid x \notin B \wedge x \in L(N)\}$$

is in  $\text{UP}_{\leq k'}$ . Namely, we first *deterministically* check—using some  $\text{P}$  algorithm for  $B$ , and we just argued that  $B \in \text{P}$  so some such algorithm exists under our current assumptions—whether  $x$  is in  $B$ . If  $x \in B$  we reject, and if  $x \notin B$  we directly simulate  $N(x)$ . This latter simulation will have at most  $k'$  accepting paths, as  $x \notin B$  precludes there being exactly  $k' + 1$  paths, and  $N$ 's choice precludes there being more than  $k' + 1$  paths. Since  $D \in \text{UP}_{\leq k'}$ , we conclude from our inductive hypothesis (which was  $\text{P} = \text{UP} \implies \text{P} = \text{UP}_{\leq k'}$ ), and our assumption that  $\text{P} = \text{UP}$ , that  $D \in \text{P}$ . Since  $\text{P}$  is closed under union,  $B \cup D \in \text{P}$ . However,  $L = B \cup D$ , and since  $L$  was an arbitrary member of  $\text{UP}_{\leq k'+1}$ , we have now established our inductive step, namely, that  $\text{P} = \text{UP} \implies \text{P} = \text{UP}_{\leq k'+1}$ .  $\square$

It remains an open research issue whether Theorem 2.7 can be extended to a nonconstant level of many-to-one-ness. Certainly, the proof technique used above does not seem valuable beyond the constant-to-one case.

## 2.3 Strong, Total, Commutative, Associative One-Way Functions Exist If and Only If One-Way Functions Exist

In this chapter, we have until now focused on the theory of one-argument one-way functions. The present section studies two-argument (henceforth denoted 2-ary) one-way functions. Such functions arise naturally in the study of cryptographic protocols. In fact, their study and the most interesting new issues they pose were directly motivated by proposed cryptographic protocols.

In particular, Rabi, Rivest, and Sherman have proposed interesting protocols for digital signatures and multiparty secret-key agreement that used as building blocks (hypothetical) 2-ary one-way functions having also such properties as being total, commutative, associative, and “strongly noninvertible.”

This immediately raises the question of how likely it is that such functions exist. As we will see in this section, a satisfying answer can be given to this question: One-way functions exist if and only if strongly noninvertible, total, commutative, associative, 2-ary one-way functions exist.

Before we prove this theorem, we first formally define what we mean for 2-ary functions by each of these properties.

**Definition 2.10** *We say a (possibly nontotal) 2-ary function  $f : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$  is honest if*

$$(\exists \text{ polynomial } q)(\forall y \in \text{range}(f))(\exists x, x')[|x| + |x'| \leq q(|y|) \wedge f(x, x') = y].$$

Note that Definition 2.10 does not require that  $|x| + |x'| \leq q(|y|)$  hold for every  $x$  and  $x'$  for which  $f(x, x') = y$ ; the definition merely requires that each element of  $\text{range}(f)$  have at least one appropriate pair  $(x, x')$ .

**Definition 2.11** *We say a (possibly nontotal) 2-ary function  $f : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$  is (polynomial-time) invertible if there is a (possibly nontotal) polynomial-time computable function  $g$  such that, for each  $y \in \text{range}(f)$ ,*

$$\begin{aligned} & y \in \text{domain}(g) \wedge \\ & (\text{first}(g(y)), \text{second}(g(y))) \in \text{domain}(f) \wedge \\ & f(\text{first}(g(y)), \text{second}(g(y))) = y, \end{aligned}$$

where the projection functions  $\text{first}(z)$  and  $\text{second}(z)$  denote, respectively, the first and second components of the unique ordered pair of strings that when paired give  $z$ .

**Definition 2.12** *We say a (possibly nontotal) 2-ary function  $f : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$  is one-way if*

1.  $f$  is polynomial-time computable,
2.  $f$  is not polynomial-time invertible, and
3.  $f$  is honest.

Next, we turn towards defining strong noninvertibility. Informally put, strong noninvertibility means that even given one of the inputs as well as the output, the other input cannot in general be computed in polynomial time. We capture this formally as Definition 2.14 below, being careful to avoid the analog for strongness of a length-based “honesty” trick that might artificially block inversion via shrinking lengths in one argument.<sup>2</sup>

---

<sup>2</sup> Note that our definition does confer s-honesty on functions that wildly shrink their arguments, but do so in parallel for both their arguments. For example, consider the function  $f(a, b)$  that equals  $\lceil \log \log \log(\max(|a|, 4)) \rceil$  when  $|a| = |b|$ , and that is undefined otherwise. This obviously dishonest function is in fact “s-honest.”

**Definition 2.13** We say a (possibly nontotal) 2-ary function  $f : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$  is *s-honest* if

1.  $(\exists \text{ polynomial } q) (\forall y, a : (\exists b)[f(a, b) = y] \implies (\exists b')[|b'| \leq q(|y| + |a|) \wedge f(a, b') = y].$
2.  $(\exists \text{ polynomial } q) (\forall y, b : (\exists a)[f(a, b) = y] \implies (\exists a')[|a'| \leq q(|y| + |b|) \wedge f(a', b) = y].$

**Definition 2.14** We say a (possibly nontotal) 2-ary function  $f : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$  is *strongly (polynomial-time) noninvertible* if it is *s-honest* and yet neither of the following conditions holds.

1. There is a (possibly nontotal) polynomial-time computable function  $g : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$  such that  $(\forall y \in \text{range}(f))(\forall x_1, x_2 : (x_1, x_2) \in \text{domain}(f) \wedge f(x_1, x_2) = y) \implies [(y, x_1) \in \text{domain}(g) \wedge f(x_1, g(y, x_1)) = y]$ .
2. There is a (possibly nontotal) polynomial-time computable function  $g : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$  such that  $(\forall y \in \text{range}(f))(\forall x_1, x_2 : (x_1, x_2) \in \text{domain}(f) \wedge f(x_1, x_2) = y) \implies [(y, x_2) \in \text{domain}(g) \wedge f(g(y, x_2), x_1) = y]$ .

We define associativity and commutativity only for the case of total functions, as Theorem 2.16 uses these notions only for that case. However, we mention in passing (and the Bibliographic Notes discuss in more detail) that if one tries to apply analogs of these notions to partial functions, one must be very careful. There are two different ways one can do this, and confusing them has led to serious problems in the literature.

**Definition 2.15**

1. We say a total, 2-ary function  $f : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$  is *associative* if

$$(\forall x, y, z)[f(f(x, y), z) = f(x, f(y, z))].$$

2. We say a total, 2-ary function  $f : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$  is *commutative* if

$$(\forall x, y)[f(x, y) = f(y, x)].$$

For example, applying the notion in the natural way to functions that may interpret their input strings as integers, the 2-ary function “multiplication (of integers)” is associative and commutative, the 2-ary function “concatenation (of strings)” is associative but not commutative, and the 2-ary function “subtraction (of integers)” is neither associative nor commutative.

We can now state the main theorem of this section.

**Theorem 2.16** *One-way functions exist if and only if strongly noninvertible, total, commutative, associative, 2-ary one-way functions exist.*

Before we prove Theorem 2.16, we prove the following easy proposition, from which it is clear that Theorem 2.16 can be alternatively interpreted as saying that for 2-ary functions, the existence of one-way functions stands or falls together with the existence of strongly noninvertible, total, commutative, associative one-way functions, and also stands or falls together with  $P \neq NP$ .

**Proposition 2.17** *The following are equivalent.*

1. One-way functions exist.
2. 2-ary one-way functions exist.
3.  $P \neq NP$ .

**Proof** (1)  $\equiv$  (3) by Theorem 2.5.

To show (2)  $\implies$  (1), let  $\langle \cdot, \cdot \rangle$  be a pairing function with the standard nice properties (see the proof of Theorem 2.5), and that in addition is nondecreasing in each argument when the other argument is fixed. Let  $f : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$  be any 2-ary one-way function. Then it is easy to see that  $g : \Sigma^* \rightarrow \Sigma^*$  defined by

$$g(z) = f(\text{first}(z), \text{second}(z))$$

is a one-way function, where  $\text{first}(z)$  denotes the first component of the (unique) pair mapped to  $z$  by the pairing function, and  $\text{second}(z)$  denotes the second component of the (unique) pair mapped to  $z$  by the pairing function. So (2)  $\implies$  (1).

If  $h : \Sigma^* \rightarrow \Sigma^*$  is a one-way function, then  $h' : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$  defined by  $h'(x, y) = \langle h(x), y \rangle$  is easily seen to be a 2-ary one-way function, as is  $h''(x, y) = \langle h(x), h(y) \rangle$  (in fact, the latter even throws in for free strong noninvertibility). So (1)  $\implies$  (2).  $\square$

We now turn to the proof of Theorem 2.16.

**Proof of Theorem 2.16** It follows from Proposition 2.17 that one-way functions exist if 2-ary one-way functions exist, and thus one-way functions certainly exist if strongly noninvertible, total, commutative, associative, 2-ary one-way functions exist. So we need only show that if one-way functions exist then strongly noninvertible, total, commutative, associative, 2-ary one-way functions exist. In fact, by Proposition 2.17 it suffices to show that if  $P \neq NP$  then there exist functions.

So, assume  $P \neq NP$ . Then there will exist an NPTM  $N'$  such that  $L(N') \in NP - P$ . By standard machine manipulation it follows that there exists a polynomial  $p$ , satisfying  $(\forall n)[p(n) > n]$ , and an NPTM  $N$  such that  $L(N') = L(N)$  (thus,  $L(N) \in NP - P$ ) and on each input  $x$  each computation path of  $N(x)$  has exactly  $p(|x|)$  bits. We will view these paths (the sequences of nondeterministic guesses) as potential witnesses for  $x \in L(N)$ , and we will call such a path a *witness* (for  $x \in L(N)$ ) exactly if it is an accepting path of  $N(x)$ . To formalize this, we define  $W(x)$  to be the set of all witnesses for  $x \in L(N)$ . Note that  $x \in L(N) \iff W(x) \neq \emptyset$ , and that each witness has length polynomially related to  $|x|$ . Note also that, due to the “by standard

machine manipulation” sentence above, a string can never be a witness for its own membership.

Let our pairing function  $\langle \cdot, \cdot \rangle$  be as in the proof of Proposition 2.17.

Let  $t$  be any fixed string such that  $t \notin L(N)$ . By  $t1$  we denote  $t$  with the bit 1 appended.

We now define our strongly noninvertible, total, commutative, associative, 2-ary one-way function,  $f$ . Let  $f$  be as follows.

$$f(u, v) = \begin{cases} \langle x, \text{lexmin}(w_1, w_2) \rangle & \text{if } u = \langle x, w_1 \rangle \wedge v = \langle x, w_2 \rangle \wedge \{w_1, w_2\} \subseteq W(x) \\ \langle x, x \rangle & \text{if } (\exists w \in W(x))\{u, v\} = \{\langle x, x \rangle, \langle x, w \rangle\} \\ \langle t, t1 \rangle & \text{otherwise,} \end{cases}$$

where  $\text{lexmin}(z, z')$  denotes the lexicographically lesser of  $z$  and  $z'$ . We mention immediately that due to our having required that  $p(n) > n$ , in the  $\{u, v\} = \{\langle x, x \rangle, \langle x, w \rangle\}$  case above, there is no chance of  $x$  being a witness string for  $x \in L(N)$ , as all witnesses for  $x$  are longer than  $x$ .

Intuitively, the action of the above function is as follows. It has two inputs, and it expects them each to be a pairing of the same string,  $x$ , with itself or with a witness for  $x \in L(N)$ . (Note that

$$\{\langle x, w \rangle \mid w \in W(x)\}$$

is in  $P$ , i.e., witness testing is easy.) If the input is of the wrong form—two different first components, or the same first components but some second component that is neither  $x$  nor a member of  $W(x)$ —then  $f$  outputs a distinguished string that will function as a garbage dump. Otherwise  $f$  *will reduce by one the number of witness instances*. That is, if both its inputs are  $x$  paired with elements of  $W(x)$ , then  $f$  reduces the number of witness instances from two to one by outputting  $x$  paired with the lexicographically smaller<sup>3</sup> witness; it is legal for the second components to both hold the same witness, which is why we said “witness instance” above, rather than “witness.” Similarly, if  $f$  has just one witness instance among the second components of the (first-component-matching) inputs, then it will reduce to zero the number of witness instances by outputting  $\langle x, x \rangle$ . If  $f$  has no witness instances among its two inputs, it maps to the garbage-dump string.

Let us verify that  $f$  is a strongly noninvertible, total, commutative, associative, 2-ary one-way function.

From its definition, it is clear that 2-ary function  $f$  is total and polynomial-time computable.

<sup>3</sup> As we will see, choosing the lexicographically smaller witness will help us obtain the algebraic properties we seek.

Also,  $f$  is commutative. Why? Consider  $f(u, v)$ . If either of the last two cases of the definition of  $f$  hold we trivially have  $f(u, v) = f(v, u)$ , and if the first case holds, we also have  $f(u, v) = f(v, u)$ , since  $\text{lexmin}$  itself is commutative.

$f$  also is strongly noninvertible. It is not hard to see that  $f$  is  $s$ -honest. Interestingly, the only nonobvious case regarding  $s$ -honesty is inverting the garbage output string given one of the inputs that mapped to it. However, due to our having been so specific about fixing the witness of a string  $x$  to be of length exactly  $p(|x|)$ , even with either argument fixed we can find a string for the other argument that is of appropriate length. So,  $f$  is  $s$ -honest. Continuing, let us suppose that  $f$  is not strongly noninvertible. Then, given its  $s$ -honesty, this must be because it can be inverted with respect to (at least) one of its two arguments given the other argument and the output. Let us consider the first case, i.e., that condition 1 of the definition of strong noninvertibility (Definition 2.14) holds. So, there is a polynomial-time function  $g$  such that for each  $x \in L(N)$  it holds that  $g(\langle x, x \rangle, \langle x, x \rangle)$  must output a string of the form  $\langle x, w \rangle$  with  $w \in W(x)$ . (If  $x \notin L(N)$ , then  $g(\langle x, x \rangle, \langle x, x \rangle)$  may output anything it likes, but as testing membership in  $W(x)$  is easy it cannot possibly fool us into thinking that it has output a witness for  $x \in L(N)$ .) This gives a polynomial-time algorithm for testing membership in  $L(N)$ : On input  $x$ , compute  $g(\langle x, x \rangle, \langle x, x \rangle)$  and accept exactly if  $g(\langle x, x \rangle, \langle x, x \rangle)$  is of the form  $\langle x, w \rangle$  for some  $w \in W(x)$ . However, we know that  $L(N) \notin P$ , so our supposition that condition 1 of Definition 2.14 holds must itself be wrong. By the symmetric, analogous argument, condition 2 of Definition 2.14 cannot hold. So,  $f$  is indeed strongly noninvertible.

It would be tempting to claim that strong noninvertibility immediately implies that the “noninvertibility” component of Definition 2.12 is satisfied. However, for subtle reasons, this is not so. (Indeed, it is known that unless  $P = NP$  there exist honest, strongly noninvertible functions that are invertible.) Nonetheless, the flavor of the preceding argument about strong noninvertibility still can be used to show noninvertibility in the current setting; simply inverting based on the output  $\langle x, x \rangle$  will put into our hands strings at least one of which is a pair having as its second component a witness for  $x \in L(N)$ , if any such witness exists.

Furthermore,  $f$  is honest. Due to the fact that (by our choice of  $N$ ) witnesses are of polynomial length relative to the strings whose membership in  $L(N)$  they certify—and that our pairing function is polynomial-time computable and polynomial-time invertible and so it cannot distort lengths by more than a polynomial amount, i.e.,  $|\langle a, b \rangle|$  and  $|a| + |b|$  each are bounded by a polynomial in the other—the first two cases of the definition of  $f$  pose no problem. The only case that remains is when our range element is the special string  $\langle t, t1 \rangle$ . However, a single range point can never on its own preclude honesty. That is, consider the shortest string mapping to  $\langle t, t1 \rangle$  and choose our honesty polynomial large enough to stretch from  $|\langle t, t1 \rangle|$  up to the length

of that one string and to handle the honesty of the first two cases of the definition of  $f$ . This is legal and handles this case, as honesty requires that range elements have at least one relatively short inverse, not that they have all their inverses be relatively short.

So, we have left only the issue of whether  $f$  is associative. Our goal is to show that, for each  $z, z', z'' \in \Sigma^*$ ,

$$f(f(z, z'), z'') = f(z, f(z', z'')).$$

Let *first* and *second* be as in the proof of Proposition 2.17. We say a string  $a$  is *legal* if

$$(\exists x)(\exists w)[w \in W(x) \wedge a = \langle x, w \rangle].$$

It follows from the definition of  $f$  that if at least two of  $z$ ,  $z'$ , and  $z''$  are not legal then  $f(f(z, z'), z'') = f(z, f(z', z'')) = \langle t, t1 \rangle$ . (Recall that by the choice, on page 40, of  $t$ , and by the definition of  $f$ ,  $\langle t, t1 \rangle$  will function here as an “absorbing” element.) Similarly,  $f(f(z, z'), z'') = f(z, f(z', z'')) = \langle t, t1 \rangle$  unless  $\text{first}(z) = \text{first}(z') = \text{first}(z'')$ . And if  $\text{first}(z) = \text{first}(z') = \text{first}(z'')$  and exactly one  $z$ ,  $z'$ , and  $z''$  is not legal, then we still will have  $f(f(z, z'), z'') = f(z, f(z', z'')) = \langle t, t1 \rangle$  unless the one that is not legal is the string  $\langle \text{first}(z), \text{first}(z) \rangle$ .

The only remaining case is that  $\text{first}(z) = \text{first}(z') = \text{first}(z'')$ , and either two or three of  $z$ ,  $z'$ , and  $z''$  have second components belonging to  $W(\text{first}(z))$ , and in the case that exactly two of  $z$ ,  $z'$ , and  $z''$  have second components belonging to  $W(\text{first}(z))$  it also holds that the remaining string is  $\langle \text{first}(z), \text{first}(z) \rangle$ .

If  $\text{first}(z) = \text{first}(z') = \text{first}(z'')$  and exactly two of  $z$ ,  $z'$ , and  $z''$  have second components belonging to  $W(\text{first}(z))$  and the remaining string is  $\langle \text{first}(z), \text{first}(z) \rangle$ , then applying the definition of  $f$  we have

$$f(f(z, z'), z'') = f(z, f(z', z'')) = \langle \text{first}(z), \text{first}(z) \rangle.$$

If  $\text{first}(z) = \text{first}(z') = \text{first}(z'')$  and exactly three of  $z$ ,  $z'$ , and  $z''$  have second components belonging to  $W(\text{first}(z))$ , then applying the definition of  $f$  we see that

$$f(f(z, z'), z'') = f(z, f(z', z'')) = \langle \text{first}(z), q \rangle,$$

where  $q$  is the lexicographically least of  $\text{second}(z)$ ,  $\text{second}(z')$ , and  $\text{second}(z'')$ .

Thus,  $f$  is associative.

So, we have shown that  $f$  is a strongly noninvertible, total, commutative, associative, 2-ary one-way function.  $\square$

## 2.4 OPEN ISSUE: Low-Ambiguity, Commutative, Associative One-Way Functions?

We ideally would like our one-way functions to be one-to-one or, if that cannot be achieved, to have as few preimages as possible. Exactly what can



be said about upper and lower bounds on the degree of many-to-one-ness of total, associative, 2-ary one-way functions—under various assumptions or unconditionally? Some preliminary work has been done, but matching upper and lower bounds have so far proven elusive.

A major open issue regarding one-way functions is whether one-way functions exist if the polynomial hierarchy does not collapse. That is, can one prove that  $P = UP \implies PH$  collapses? For that matter, can one prove that  $UP = NP \implies PH$  collapses?

## 2.5 Bibliographic Notes

Section 2.1 covers two related notions: one-way functions and one-to-one one-way functions. The earliest citation we know of for the one-way functions part of Theorem 2.5 is a paper by Watanabe ([Wat88], see also [BDG95, Sel92, BFH78, Bra79]). The one-to-one one-way functions part of Theorem 2.5, and the related definitions, are due to Grollmann and Selman [GS88]. This result was also obtained independently by Berman [Ber77] and Ko [Ko85].

Of course, there are a range of “how many”-to-one-ness levels between one-to-one and many-to-one, and they similarly and very naturally have been shown to be characterized by the collapse of complexity classes. For example, Allender and Rubinfeld [AR88] show that polynomial-to-one one-way functions exist if and only if  $P \neq \text{FewP}$ .

Similarly, the constant-to-one cases are relevant to Sect. 2.2. This section is based on the work of Watanabe [Wat88], who in particular established Theorem 2.7. Definition 2.8 is due to Beigel [Bei89] and Fact 2.9 is stated explicitly in Hemaspaandra and Zimand [HZ93], who study the structure and potential collapses of bounded-ambiguity classes such as  $UP_{\leq k}$  and  $\text{co}UP_{\leq k}$ .

Regarding Sect. 2.3, Rabi, Sherman, and Rivest raised the issue of how algebraic properties interacted with 2-ary one-way functions ([RS93, RS97, She86], see the discussion and literature pointers in those papers), and developed secret-key agreement and digital signature protocols that interestingly use such functions as building blocks. Rabi and Sherman proved [RS93, RS97] that  $P = NP$  if and only if commutative, associative, 2-ary one-way functions exist. However, their functions are nontotal and are not strongly noninvertible. The main theorem of Sect. 2.3, Theorem 2.16, is due to Hemaspaandra and Rothe [HR99]. The counterintuitive result mentioned in passing in the proof of Theorem 2.16—that if  $P \neq NP$  then there exist honest, strongly noninvertible, polynomial-time computable functions that are polynomial-time invertible—is due to Hemaspaandra, Paskanen, and Rothe [HPR01].

As mentioned on page 38, some tricky issues arise in the study of associativity of partial functions. In fact, two distinct notions of associativity can be studied, and they are inspired by ideas dating back to the early work of Kleene [Kle52], namely, “complete equality” versus “weak equality.” Simply

put, the issue is whether  $f(a) = x$  should be considered true when  $x \in \Sigma^*$  and  $a \notin \text{domain}(f)$ . These issues are discussed in detail by Hemaspaandra and Rothe [HR99], who in particular prove that, due to a subtle confusion between the two notions, a claim of Rabi and Sherman [RS97, She86] is invalid if  $\text{UP} \neq \text{NP}$ .

Regarding Sect. 2.4, Rabi and Sherman ([RS97], see the discussion in [HR99]) show that no total, “weakly associative,” 2-ary one-way functions exist, which implies that no total, associative, 2-ary one-way functions exist. Hemaspaandra and Rothe [HR99] note that lack of one-to-one-ness holds *per force* for any commutative 2-ary function having some distinct elements in its domain, and they propose a more general notion to study that they call unordered injectivity. Homan [Hom00] provides a deep and direct study of what upper and lower bounds hold on the degree of many-to-one-ness of 2-ary functions, and of the interactions between that, algebraic properties, invertibility properties, and complexity-theoretic hypotheses. For example, he proves that if  $\text{P} \neq \text{UP}$  then there exists a  $\mathcal{O}(n)$ -to-one, strongly noninvertible, total, associative, 2-ary one-way function.

Selman [Sel92] has written a general survey of one-way functions, and a later survey by Beygelzimer et al. [BHHR99] focuses on the study of associative, 2-ary one-way functions. These references all study worst-case cryptocomplexity. References on the related but distinct study of average-case cryptocomplexity and that area’s notion of one-way functions include the books of Luby [Lub96] and Goldreich [Gol01]. The result, mentioned in this chapter’s introduction, that the type of one-way function used in average-case cryptography exists if and only if pseudorandom generators exist is due to Håstad, Impagliazzo, Levin, and Luby [HILL99].

### 3. The Tournament Divide and Conquer Technique

If computer science were a country rather than a field, one can well imagine that its motto would be “Divide and Conquer,” which might have edged out “In Polynomial Time We Trust.” Indeed, divide and conquer techniques accompany the computer scientist from the introduction to binary search through the mastery of cutting-edge algorithmic techniques. However, computer scientists do not own the franchise. Divide and conquer techniques are useful in many fields.

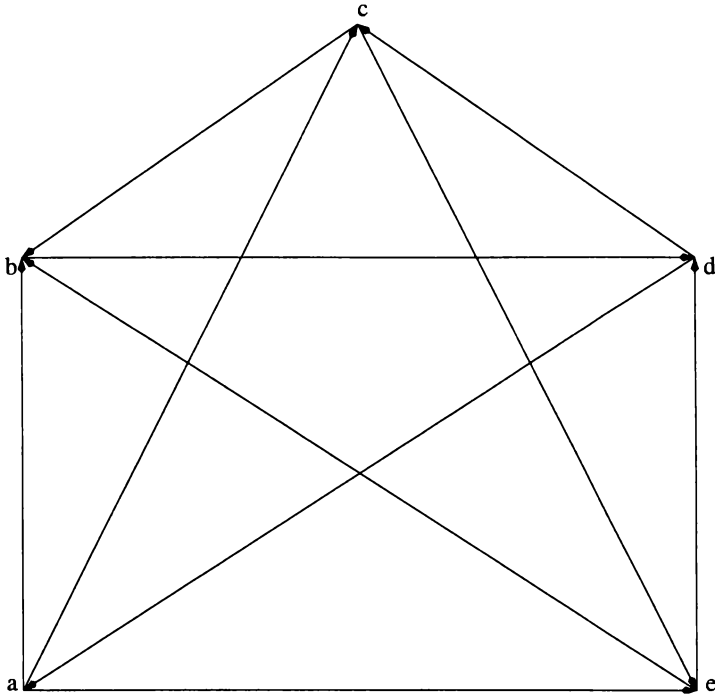
In this chapter’s GEM, we will prove via divide and conquer a result from tournament theory. Perhaps surprisingly, we will see that this tournament theory result immediately yields an upper bound on the nonuniform complexity of semi-feasible sets, i.e., how much advice various classes need to accept them (Sect. 3.1). The result—with a bit of work—also proves that NP machines cannot find unique satisfying assignments to satisfiable formulas unless the polynomial hierarchy collapses (Sect. 3.3).

We also will exactly pinpoint the maximum nonuniform complexity of semi-feasible sets (Sect. 3.2).

#### 3.1 GEM: The Semi-feasible Sets Have Small Circuits

Consider a  $k$ -node graph, having no self-loops, such that for each pair,  $\{a, b\}$ , of distinct nodes we have a directed edge from  $a$  to  $b$  or a directed edge from  $b$  to  $a$ , but not both such edges. Such a structure is known as a  $k$ -tournament (see Fig. 3.1). One may think of this as a round-robin tournament in which each node represents a player, and the edge between two players is directed towards whichever of the two won the match they played (no ties are allowed). Tournaments have been extensively studied. However, for our purposes at the moment, we need only the following simple claim (Theorem 3.1): In any  $k$ -tournament, there exists some small collection of players such that every player in the tournament defeats at least one member of that small collection (we consider each member, by convention, to defeat him- or herself).

For a graph  $G$ , let  $V_G$  denote the vertex set of  $G$  and let  $E_G$  denote the edge set of  $G$ . Let  $(a, b)$  denote an edge pointing from  $a$  to  $b$ . In our sports tournament analog, we draw between players  $a$  and  $b$  the edge  $(a, b)$  if  $b$  defeats  $a$ , and  $(b, a)$  if  $a$  defeats  $b$ ; that is, edges point towards winners.



**Fig. 3.1** A 5-Tournament

**Theorem 3.1** *If  $G$  is a  $k$ -tournament on nodes  $V_G = \{1, 2, \dots, k\}$  then there exists a set  $H \subseteq \{1, 2, \dots, k\}$  such that*

1.  $|H| \leq \lfloor \log(k+1) \rfloor$ , and
2. for each  $v \in V_G - H$ , there is some  $g \in H$  such that  $(g, v) \in E_G$ .

**Proof** Consider a tournament in which there are  $k$  players. Thus, each player plays  $k - 1$  games. Note that some player must lose at least half the time, i.e., must lose at least  $\lceil \frac{k-1}{2} \rceil$  games. This is because in each game someone wins and someone loses, so the total number of wins equals the total number of losses. However, if no player were to lose at least half the time, then each player individually has strictly more wins than losses, and thus overall there are strictly more wins than losses, which is impossible.

Select some player who loses at least half the games he plays and place him in the set  $H$ . Remove from consideration that player and all the players that defeat that player. Note that the new player set has at most  $k - (1 + \lceil \frac{k-1}{2} \rceil) = \lceil \frac{k}{2} \rceil - 1$  players. Consider the tournament induced by restricting our attention to only the edges (games) in the old tournament played between players of this reduced set. Note that, in this reduced tournament, our previous argument

will again apply, and so there will be a player who loses to at least half the players in the reduced tournament. Add that player to  $H$  and again reduce the tournament. Continuing this process, we eventually arrive at a set  $H$  having property 2 from the statement of the theorem, since there will eventually be no vertices under consideration. Also, the number of elements in  $H$  is bounded by the recurrence relation:  $S(0) = 0$  and, for each  $k \geq 1$ ,  $S(k) \leq 1 + S(\lceil \frac{k}{2} \rceil - 1)$ . As is standard, this recurrence relation implies  $S(k) \leq \lfloor \log(k+1) \rfloor$ .  $\square$

This innocuous theorem in fact proves that the semi-feasible sets have small circuits. (Readers who are not familiar with the definitions of small circuits and semi-feasible sets should at this point quickly read the introductions to these topics contained in, respectively, Sects. A.6 and A.14.)

**Theorem 3.2**  $P\text{-sel} \subseteq P/\text{poly}$ .

Informally, the proof goes as follows: At each length  $n$  in any semi-feasible (equivalently, P-selective) set  $L$ , there will always exist, by Theorem 3.1, a small set of nodes in  $L$  such that every element in  $L^n$  defeats one of these nodes, and this small set of nodes itself will function for us as a “small advice set” for  $L$  (since each string in  $L$  will defeat one of these nodes, and by the definition of semi-feasibility any string that defeats one of these nodes must be in  $L$ ).

**Proof** Consider a semi-feasible set  $L$ . Recall that to prove that  $L$  has small circuits (i.e.,  $L \in P/\text{poly}$ ), it suffices (see Sect. A.6) to provide a function  $g$  and a set  $A \in P$  such that

$$(\forall x) [x \in L \iff \langle x, g(|x|) \rangle \in A] \quad (3.1)$$

and

$$(\exists \text{ polynomial } q) (\forall n) [|g(n)| \leq q(n)]. \quad (3.2)$$

Consider the length  $n$  strings in  $L$ ,  $L^n$ . Let  $f : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$  be a polynomial-time computable function that is a P-selector for semi-feasible set  $L$ . Without loss of generality, we assume that  $f(a, b) = f(b, a)$  for all  $a$  and  $b$ , as if  $f$  does not satisfy this condition, it can be replaced by the function

$$f'(a, b) = f(\min\{a, b\}, \max\{a, b\}),$$

which does satisfy this condition and, as follows clearly from the fact that  $f$  is a P-selector for  $L$ ,  $f'$  is also a P-selector for  $L$ . Consider the tournament on  $L^n$  induced by  $f$ . That is, consider a simple (i.e., having no self-loops) graph  $G$  whose nodes are the elements of  $L^n$ , and such that for any  $a, b \in L^n$ ,  $a \neq b$ , it holds that  $(a, b) \in E_G \iff f(a, b) = b$ . Note that this indeed is a tournament. Theorem 3.1, applied to this tournament, states that there exists a small set  $H_n$  (in particular,  $|H_n| \leq \lfloor \log(1 + |L^n|) \rfloor \leq n + 1$ ) such that this set  $H_n$  contains only members of  $L^n$  and for every element of  $L^n$  there is some element  $h \in H_n$  satisfying  $f(h, x) = x$ . Note that if  $x \in H_n$  the

test is whether  $f(x, x) = x$ , which is always true. Note also that if  $L$  contains no length  $n$  elements, then  $H_n = \emptyset$ .

We now state the advice function and advice interpreter for semi-feasible set  $L$ . The advice function,  $g(n)$ , outputs  $H_n$ , coded in any fixed natural way. Clearly, equation 3.2 holds as, for each  $n$ ,  $H_n$  has at most  $n + 1$  elements, each  $n$  bits long. Our advice interpreter set for  $L$  is

$$A = \{\langle x, y \rangle \mid y \text{ is a (possibly empty) list of length } n \text{ elements } v_1, \dots, v_z, \text{ and for some } j \text{ it holds that } f(v_j, x) = x\}.$$

Clearly,  $A \in P$ , as  $f$  is a polynomial-time computable function. Does equation 3.1 hold? Let  $x \in L$ . Then  $\langle x, g(|x|) \rangle \in A$  by our choice of  $g$  and  $A$ . Let  $x \notin L$ . Let  $n = |x|$ . Suppose  $\langle x, g(|x|) \rangle \in A$ . Then for some  $h_i \in H_n$  it must hold that  $f(h_i, x) = x$ . However, all elements of  $H_n$  are in  $L^n$ , so, since  $f$  is a P-selector function, the fact that  $x$  has defeated an element of  $L$  implies that  $x \in L$ . So if  $x \notin L$  then  $\langle x, g(|x|) \rangle \notin A$ .  $\square$

**Pause to Ponder 3.3** *In this section, we saw that  $P\text{-sel} \subseteq P/\text{poly}$ . Can one, based on the proof presented in this section, say a bit more about the number of advice bits than merely that a polynomial number (the “poly” of  $P/\text{poly}$ ) of advice bits suffice? For example, do  $\mathcal{O}(n^2)$  bits suffice? That is, does it hold that  $P\text{-sel} \subseteq P/\text{quadratic}$ ? Hint: Consider the cardinality of the sets  $H_n$  in the proof of Theorem 3.2, and the number of bits in each element of  $H_n$ .*

*Looking towards the topic of the next section, one can also ask: Is there a class  $C$  such that  $P\text{-sel} \subseteq C/\text{linear}$ ?*

## 3.2 Optimal Advice for the Semi-feasible Sets

In some sense, computer science is the study of the efficient handling of information. In this section we ask: How much information do semi-feasible sets contain?

In the previous section, we saw that  $P\text{-sel} \subseteq P/\text{poly}$ . However, we proved a bit more. The actual advice function used at length  $n$  was a coding of at most  $n + 1$  strings of length  $n$ . Thus,  $\mathcal{O}(n^2)$  bits are easily sufficient. So we have the following theorem.

### Definition 3.4

1. Let *linear* denote the class of all functions  $f$  such that  $f(n) = \mathcal{O}(n)$ .<sup>1</sup>
2. Let *quadratic* denote the class of all functions  $f$  such that  $f(n) = \mathcal{O}(n^2)$ .

---

<sup>1</sup> We could also have defined  $\widehat{\text{linear}}$  to be all functions  $f$  such that for some  $c$  and all  $n$  we have  $f(n) = cn$ . Though  $\text{linear} \neq \widehat{\text{linear}}$ , it is not hard to see that  $P/\text{linear} = P/\widehat{\text{linear}}$ , as the length of a linear function itself holds little information—at most  $\mathcal{O}(\log n)$  bits. Analogous comments hold for the quadratic case.

**Theorem 3.5**  $P\text{-sel} \subseteq P/\text{quadratic}$ .

Can we get by with a subquadratic number of advice bits? In this section, we will see that linear-sized advice suffices, if we are allowed to use powerful advice interpreters. In particular, rather than use advice interpreters running in deterministic polynomial time, we will use advice interpreters running in probabilistic polynomial time (Theorem 3.7) and nondeterministic polynomial time (Theorem 3.10). We will eventually show that linear advice is the best one can do; no strength of advice interpreter can always work successfully on the semi-feasible sets using sublinear advice.

Recall from Sect. A.6 that to prove that  $L \in \mathcal{C}/\text{linear}$  we must provide a function  $g$  and a set  $A \in \mathcal{C}$  such that

$$(\forall x) [x \in L \iff \langle x, g(|x|) \rangle \in A] \quad (3.3)$$

and

$$(\exists q \in \text{linear}) (\forall n) [|g(n)| = q(n)]. \quad (3.4)$$

We now prove that  $P\text{-sel} \subseteq PP/\text{linear}$ . That is, linear advice suffices to accept semi-feasible sets, given advice interpreters that are probabilistic polynomial-time machines (see Sect. A.12 for an introduction to PP). Later in this section, we will extend this result by showing the even stronger claim that  $P\text{-sel} \subseteq NP/\text{linear}$ .

**Pause to Ponder 3.6** *Prove that  $P\text{-sel} \subseteq PP/\text{linear}$ . [Hint: Count!]*

**Theorem 3.7**  $P\text{-sel} \subseteq PP/\text{linear}$ .

**Proof** Let us be given a set  $L \in P\text{-sel}$  and a P-selector function  $f$  for  $L$ . As before, without loss of generality, we may assume that for all  $a$  and  $b$  it holds that  $f(a, b) = f(b, a)$ . Our advice function will be the census function of  $L$  at the given length, i.e.,  $g(n) = ||L^=n||$ , padded if needed with leading zeros so as to be exactly  $n + 1$  bits long. (This is enough bits since  $0 \leq ||L^=n|| \leq 2^n$ , so there are at most  $1 + 2^n$  possible census values.) Consider a string  $y$  of length  $n$ . If  $y \in L$ , then

$$||\{z \mid n = |z| \wedge f(y, z) = z\}|| \leq ||L^=n||,$$

as only elements in  $L$  can defeat elements in  $L$  according to a P-selector function. On the other hand, if  $y \notin L$ , then

$$||\{z \mid n = |z| \wedge f(y, z) = z\}|| > ||L^=n||,$$

as each element in  $||L^=n||$  defeats  $y$ , and also  $y$  defeats  $y$ . Our advice interpretation set—the  $A$  of equation 3.3—is defined by the following, where  $m$  is interpreted as the binary representation of an integer.

$$A = \{\langle x, m \rangle \mid m \geq ||\{z \mid f(x, z) = z \wedge |z| = |x|\}||\}.$$

It is not hard to see that  $A \in \text{PP}$ . (We leave the proof as an easy exercise for the reader. Hint: Construct a PP machine that has two types of paths. For each  $z$  of length  $|x|$ , we have a path that accepts if and only if  $f(x, z) \neq z$ . Also, there will be other paths that accept or reject so as to ensure that the machine has the right acceptance/rejection threshold.) Thus,  $L \in \text{PP/linear}$ , as  $\|L^n\| \in \{0, 1, \dots, 2^n\}$ , so as noted above  $n + 1$  bits suffice for the advice function.  $\square$

Unfortunately, PP is a very powerful class.  $\text{PP} \supseteq \text{NP}$  and, within the flexibility of Turing reductions, PP contains the entire polynomial hierarchy (see Sect. A.12). It turns out that such powerful interpreters are not needed. NP interpreters suffice. To see this, we will have to return briefly to tournament theory in order to obtain an easy but useful lemma.

Given a directed graph  $G$ , and a node  $v \in V_G$ , let

$$R_{0,G}(v) = \{v\}$$

and, for each  $i > 0$ , let

$$R_{i,G}(v) = R_{i-1,G} \cup \{z \in V_G \mid (\exists w \in R_{i-1,G}(v)) [(w, z) \in E_G]\}.$$

That is,  $R_{i,G}(v)$  denotes the set of nodes that can be reached from  $v$  via directed paths of length at most  $i$ . For any  $i$ ,  $G$ , and  $S \subseteq V_G$ , define

$$R_{i,G}(S) = \{w \in V_G \mid (\exists v \in S) [w \in R_{i,G}(v)]\}.$$

Theorem 3.1 says that if  $G$  is a  $k$ -tournament, then there is a relatively small set  $H$  such that  $V_G = R_{1,G}(H)$ . That is, there exists a small collection of nodes from which all nodes can be reached via paths of length at most one. We now claim that in any  $k$  tournament, there is some node from which all nodes can be reached via remarkably short paths (just how short they are will be the topic of Pause to Ponder 3.8).

Note that it is clear, by induction, that in a  $k$ -tournament there is a node  $v$  such that  $V_G = R_{k-1,G}(v)$ . Why? When one adds a node, either it points to the node that (inductively) reached all other nodes, or it is pointed to by that node. In the former case, the new node reaches all nodes, and in the latter case, the node that inductively reached all nodes other than the new node also reaches the new node, in both cases via sufficiently short paths.

Similarly, it is clear that in a  $k$ -tournament, there is a node  $v$  such that  $V_G = R_{\lceil \log k \rceil, G}(v)$ . We quickly sketch a proof. In the proof of Theorem 3.1 we defined a sequence of nodes  $v_1, \dots, v_m$  and a sequence of sets  $T_1, \dots, T_m$ ,  $m \leq \lfloor \log(k+1) \rfloor$ , such that for every  $i$ , each element in  $T_i$  defeats  $v_i$  and  $v_i$  defeats  $v_{i+1}, \dots, v_m$ . So, for every  $i$ ,  $v_i$  is reachable from  $v_m$  by the path  $[v_m, v_{m-1}, \dots, v_i]$ , and every element  $u$  in  $T_i$  is reachable via this path with extra edge  $(v_i, u)$ . Thus, every node is reachable from  $v_m$  by a path of length  $m \leq 1 + (\lfloor \log(k+1) \rfloor - 1)$ . Also, in the special case  $k = 1$  it is clear that



length zero paths suffice. So in all cases each node is reachable via paths of length at most  $\lceil \log k \rceil$ .

$V_G = R_{\lceil \log k \rceil, G}(v)$  is a result that is strong enough to allow us to prove the forthcoming Theorem 3.10. However, we prove as Theorem 3.9 a stronger result that gives more insight into the shortness of paths in tournaments, and thus into the nonuniform complexity of the semi-feasible sets.

**Pause to Ponder 3.8** *As just discussed, in each  $k$ -tournament there is a node from which all nodes can be reached via paths of at most logarithmic length. Improve this to the claim that in each  $k$ -tournament there is a node from which all nodes can be reached via paths of length about most  $\mathcal{O}(\log^* k)$ . Beyond that, improve this further to the claim that in each  $k$ -tournament there is a node from which all nodes can be reached via paths of constant-bounded length. [Note: Don't first peek at Theorem 3.9 below, as it will bias your thoughts on this by showing you the value of the constant.]*

**Theorem 3.9** *If  $G$  is a  $k$ -tournament, then there is a  $v \in V_G$  such that  $V_G = R_{2, G}(v)$ .*

**Proof** The result obviously holds for 1-tournaments and 2-tournaments. Inductively, assume it holds for all  $\hat{k}$  tournaments. Consider an arbitrary  $\hat{k} + 1$ -tournament,  $G$ . Let  $a$  be a node of that tournament, and let  $G'$  be the  $\hat{k}$ -tournament induced by the nodes of  $G$  other than  $a$ . By induction, there is a node  $b$  in  $G'$  such that  $R_{2, G'}(b) = V_{G'}$ .

If the edge between  $a$  and  $b$  points to  $a$  we are done, as in that case  $R_{2, G}(b) = V_G$ . So let us henceforth assume that the edge between  $a$  and  $b$  points to  $b$ . If  $a \in R_{2, G}(b)$  we also are done, as in that case  $R_{2, G}(b) = V_G$ . So let us henceforth also assume that  $a \notin R_{2, G}(b)$ .

However, if  $a \notin R_{2, G}(b)$  that implies that, for each node  $c \in R_{1, G}(b)$ , the edge between  $a$  and  $c$  points from  $a$  to  $c$ . This in turn implies that  $R_{2, G}(a) = V_G$ . Why? We already know (in light of our “henceforth” assumptions)  $R_{1, G}(b) \subseteq R_{1, G}(a)$ . We also have  $R_{2, G'}(b) - R_{1, G'}(b) \subseteq R_{2, G}(a)$ , namely, as any length two path from  $b$  has as its second node a node from  $R_{1, G'}(b)$ , but all such nodes are also pointed to by  $a$ . So, since  $V_G = R_{2, G'}(b) \cup \{a\}$ , we have  $V_G \subseteq R_{2, G}(a)$ .  $\square$

Theorem 3.1, a tournament theory result, yielded a consequence about the semi-feasible sets, Theorem 3.2. Analogously, the tournament theory result proven above, Theorem 3.9, also yields our promised result about the semi-feasible sets:  $\text{P-sel} \subseteq \text{NP/linear}$ .

The linear advice at each length  $n$  is simply the element (whose existence is ensured by Theorem 3.9) that reaches all elements of  $L^n$  via extremely short paths. The nondeterministic interpreter merely guesses the short paths.

**Theorem 3.10**  $\text{P-sel} \subseteq \text{NP/linear}$ .

**Proof** Let  $L \in \text{P-sel}$ , via P-selector function  $f$ . As usual, we assume without loss of generality that  $(\forall a, b) [f(a, b) = f(b, a)]$ . Assume also that the special

case  $|x| = 0$  is hard-coded into the advice interpreter (i.e., whether  $\epsilon \in L$ ). We give the advice function,  $g$ , and the advice interpreter,  $A \in \text{NP}$ , that prove  $L \in \text{NP/linear}$ . For each  $n \geq 1$ ,  $g(n)$  will be  $1^{n+1}$  if  $L^n = \emptyset$  and otherwise equals  $0w_n$ , where  $w_n$  is the length  $n$  string in  $L^n$  such that, by Theorem 3.9, each node in the tournament induced on  $L^n$  by  $f$  (i.e., the tournament in which there is a node for each member of  $L^n$  and for  $a, b \in L^n$ ,  $a \neq b$ , directed edge  $(a, b)$  is in the graph if and only if  $f(a, b) = b$ ) can be reached from  $w_n$  via paths of length at most two. The advice interpreter set  $A$  is as follows.

$A = \{ \langle x, 0w \rangle \mid \text{there is a path of length at most two, in the tournament induced on } L^n \text{ by } f, \text{ from } w \text{ to } x \}.$

Clearly,  $g$  is of linear length and  $A \in \text{NP}$ . If  $x \in L$ , then by construction  $\langle x, f(|x|) \rangle \in A$ . If  $x \notin L$ , then  $\langle x, f(|x|) \rangle \notin A$ , as if  $\langle x, f(|x|) \rangle \in A$ , then we have a  $z \in \{0, 1, 2\}$  and a directed path

$$a_0 \rightarrow \cdots \rightarrow a_z$$

from  $w_n$  to  $x$ , i.e., one satisfying

$$a_0 = w_n \wedge a_z = x \wedge (\forall i : 0 \leq i \leq z-1) [f(a_i, a_{i+1}) = a_{i+1}].$$

So, since  $w_n \in L$ , by the definition of semi-feasibility we also have that each  $a_i$  must be in  $L$ . Thus,  $a_z = x$  must be in  $L$ . This yields a contradiction.  $\square$

Since P-sel is closed under complementation, we have the following corollary.

**Corollary 3.11**  $\text{P-sel} \subseteq \text{NP/linear} \cap \text{coNP/linear}$ .

In fact, note that Theorem 3.9 ensures that the guessed paths from  $w_n$  will be very short—of length at most two. So our NP interpreter, when resolving the question “Is  $x$  in  $L$ ?,  $|x| = n$ ” in fact need only guess a path of length at most two, with each element on the path being itself an  $n$ -bit string. So, since we can deterministically check for paths of length at most one,  $n$  nondeterministic guess bits in fact suffice. Thus, we certainly have the following corollary to the proof.

**Corollary (to the proof) 3.12** *Each semi-feasible set can be accepted with linear advice via an NP machine using linear nondeterminism.*

Though Theorem 3.10 speaks merely of linear advice, one can say a bit more about the amount of advice. In fact, the proof of Theorem 3.10 shows that

$$\text{P-sel} \subseteq \text{NP}/n+1,$$

(this is a shorthand for  $\text{P-sel} \subseteq \text{NP}/h(n)$ , where  $h(n) = n+1$ , and recall that this means that the advice string must be of length *exactly*  $h(n)$ ). Is this optimal? We now show that, though  $n+1$  bits suffice,  $n$  bits do not suffice.

**Theorem 3.13**  $\text{P-sel} \not\subseteq \text{NP}/n$ .

The proof here is less about computation than about information content. We'll see that  $n$  bits simply cannot hold enough information to disambiguate a certain family of semi-feasible sets.

**Proof** We will construct a set,  $L$ , consisting mostly of holes. That is, at widely spaced lengths, our set will include exactly some (possibly empty) left cut of the strings of that length, and at all other lengths it will be empty (see Fig. 3.2). Yet, we will ensure that the set is of limited complexity. This will allow us to conduct a brute-force search, at each shorter length that might contain strings, of exactly which strings are at those lengths.

In particular, let  $\ell_0 = 2$ , and for each  $i \geq 1$ , let  $\ell_i = 2^{2^{i-1}}$ . Let  $Q = \{\ell_0, \ell_1, \ell_2, \dots\}$ . We will construct  $L$  to ensure that the following three conditions hold.

$$L \subseteq \Sigma^{\ell_0} \cup \Sigma^{\ell_1} \cup \Sigma^{\ell_2} \cup \dots. \text{ That is, all strings in } L \text{ have} \quad (3.5)$$

lengths from the set  $Q$ .

$$\text{For each } x \text{ and } y, \text{ if } |x| = |y| \text{ and } x \leq_{lex} y \text{ and } y \in L, \text{ then} \quad (3.6)$$

$x \in L$ . That is, at each length  $L$  is a (perhaps empty) left cut of the strings at that length.

$$L \in \text{DTIME}[2^{2^n}]. \quad (3.7)$$

Our proof concludes with the following two claims, and their proofs.

**Claim 3.14** *Any set  $L$  satisfying equations 3.5, 3.6, and 3.7 is semi-feasible.*

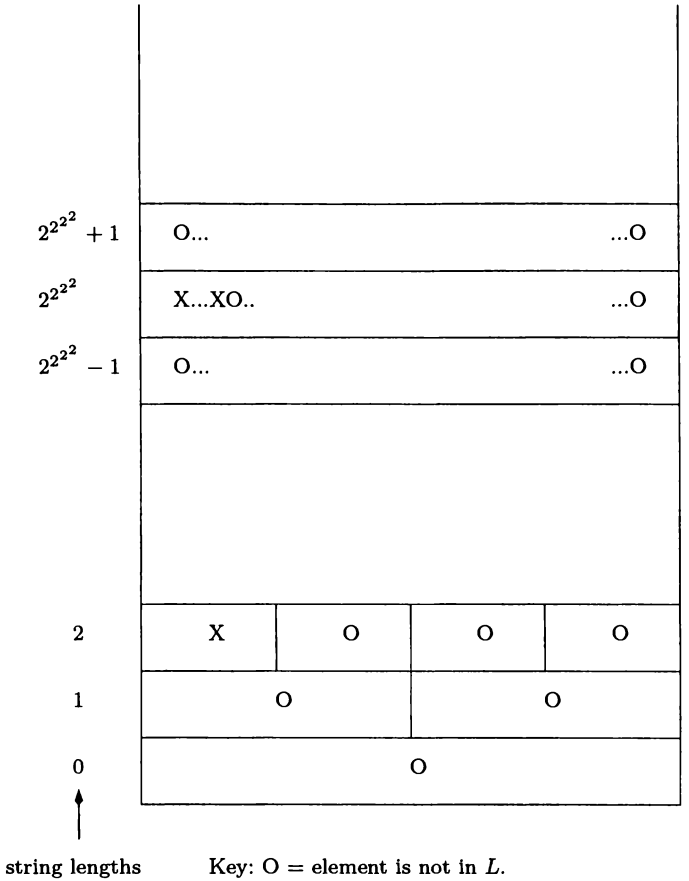
**Claim 3.15** *There is a set  $L \notin \text{NP}/n$  satisfying equations 3.5, 3.6, and 3.7.*

**Proof of Claim 3.14** Let  $L$  satisfy equations 3.5, 3.6, and 3.7. Consider the following function  $f$ .

$$f(x, y) = \begin{cases} x & \text{if } |y| \notin Q, \\ y & \text{if } |x| \notin Q \wedge |y| \in Q, \\ \min\{x, y\} & \text{if } |x| \in Q \wedge |y| \in Q \wedge |x| = |y|, \\ \min\{x, y\} & \text{if } |x| \in Q \wedge |y| \in Q \wedge |x| \neq |y| \wedge \min\{x, y\} \in L, \\ \max\{x, y\} & \text{if } |x| \in Q \wedge |y| \in Q \wedge |x| \neq |y| \wedge \min\{x, y\} \notin L. \end{cases}$$

It is clear that  $f$  is a selector function for  $L$ , as no case above outputs an element not in  $L$  if at least one of the arguments is in  $L$ . Keeping in mind the widely spaced left cut structure of  $L$ ,  $f$  can be computed in polynomial time as follows. The first three cases clearly are in polynomial time. As to

the last two cases, recall that  $Q$  is of the form  $Q = \{2, 2^{2^{2^2}}, 2^{2^{2^{2^{2^2}}}}, \dots\}$ . So if  $|x| \in Q$ ,  $|y| \in Q$ , and  $|x| \neq |y|$ , it must hold that  $\max\{|x|, |y|\} \geq 2^{2^{\min\{|x|, |y|\}}}$ . Since  $L$  is computable in time  $2^{2^n}$ , that means that a machine running in time polynomial in  $|x| + |y|$  can easily (in the last two cases) compute by



For example, the “length 2” row in this example says that  $00 \in L$ ,  $01 \notin L$ ,  $10 \notin L$ , and  $11 \notin L$ .

**Fig. 3.2** Typical Structure of  $L$

brute-force search whether  $\min\{x, y\} \in L$ . Thus,  $f$  is a P-selector for  $L$ , so  $L$  is semi-feasible. □ Claim 3.14

**Proof of Claim 3.15** Let  $N_1, N_2, N_3, \dots$  be a standard, easily computable list of NPTMs (nondeterministic polynomial-time Turing machines) such that

$$\text{NP} = \{B \mid (\exists i) [L(N_i) = B]\}.$$

Let  $\langle \cdot, \cdot \rangle$  be a fixed, nice, 2-ary pairing function. Let the enumeration  $\hat{N}_1, \hat{N}_2, \hat{N}_3, \dots$  be such that, for each  $i$  and  $k$ ,  $\hat{N}_{\langle i, k \rangle}$  is simply  $N_i$ . So in

the enumeration  $\widehat{N}_1, \widehat{N}_2, \widehat{N}_3, \dots$ , each machine  $N_i$  will appear infinitely often. Our proof proceeds by diagonalization, in stages.

*Stage  $\ell$ :* At this stage we define the contents of  $L^\ell$ . If  $\ell \notin Q$ , then set  $L^\ell = \emptyset$  and move on to stage  $\ell + 1$ .

If  $\ell \in Q$ , then do the following suite of simulations for at most  $2^{2^\ell}$  steps (total). If this number of steps is reached and we have not completed the following, then let  $L^\ell = \emptyset$ . Say  $\ell$  is the  $\langle i, k \rangle$ th element of  $Q$ ; that is, define  $i$  and  $k$  (which both will depend on  $\ell$ ) to be the unique integers satisfying  $\langle i, k \rangle = ||\{j \mid j \in Q \wedge j \leq \ell\}||$ .

Consider  $\widehat{N}_{\langle i, k \rangle}$ . For each of the  $2^\ell$  potential advice strings  $y$  of length  $\ell$ , do the following.

For each of the  $2^\ell$  strings  $x$  of length  $\ell$  run  $\widehat{N}_{\langle i, k \rangle}(\langle x, y \rangle)$ . If none of these  $2^\ell$  runs accept, then let  $rightmost_y = 1^{\ell-1}$ . Otherwise, let

$$rightmost_y = \max\{x \mid |x| = \ell \wedge \widehat{N}_{\langle i, k \rangle}(\langle x, y \rangle) \text{ accepts}\}.$$

Since for each of the  $2^\ell$  potential advice strings of length  $\ell$  we chose at most one “rightmost” string, the total number of rightmost strings is at most  $2^\ell$ . So the set

$$J_\ell = \left( \Sigma^\ell \bigcup \{1^{\ell-1}\} \right) - \{z \mid (\exists y \in \Sigma^\ell) [rightmost_y = z]\}$$

is not empty. Let  $j_\ell$  be an element of  $J_\ell$ , for example, the lexicographically smallest element of  $J_\ell$ . Set

$$L^\ell = \{x \mid \ell = |x| \wedge x \leq_{lex} j_\ell\}. \quad (3.8)$$

*End of Stage  $\ell$*

By construction, equation 3.5 holds, and by equation 3.8, equation 3.6 holds. By the fact that we allow at most  $2^{2^\ell}$  steps in the simulation, equation 3.7 holds. It remains to show that  $L \notin \text{NP}/n$ . Suppose  $L \in \text{NP}/n$  via NP language  $L'$ . Let  $i'$  be such that  $L(N_{i'}) = L'$ . Note that, for each  $k'$ ,  $\widehat{N}_{\langle i', k' \rangle} = N_{i'}$ . For all sufficiently large  $k'$ , in stage  $\langle i', k' \rangle$  the construction completes without being cut off by the  $2^{2^\ell}$  step bound. This is because the stage  $\ell$  that satisfies  $\langle i', k' \rangle = ||\{j \mid j \in Q \wedge j \leq \ell\}||$  requires about

$$2^\ell 2^\ell \left( 2^{b\ell^c} \right)^2$$

steps (for some positive constants  $b$  and  $c$  that are dependent only on  $i$ ), the three terms respectively being due to the loop over advice strings, the loop over input strings, and the simulation of an NP machine (converted to an EXP brute-force machine, and allowing quadratic overhead for the simulation of that machine on a universal EXP machine). For all sufficiently large  $\ell$ , this is less than  $2^{2^\ell}$ . So for a fixed  $i'$ , it holds that, for all but finitely many  $k'$ , the stage  $\ell$  where  $\ell$  is the  $\langle i', k' \rangle$ th element of  $Q$  will use no more than  $2^{2^\ell}$

steps. Let  $k''$  (implicitly,  $k'' = k''(i')$ ) be one  $k'$  for which completion does occur. Let  $\ell''$  denote the  $\langle i', k'' \rangle$ th element of  $Q$ . Notice that by construction (namely, the construction's choice of  $j_{\ell''}$ ) none of the  $2^{\ell''}$  advice strings of length  $\ell''$  when given as advice to  $\widehat{N}_{\langle i', k'' \rangle}$ —equivalently to  $N_{i'}$ —yields, at length  $\ell''$ ,  $L^{\ell''}$ . In particular, we have the following two claims.

1. If  $|j_{\ell''}| = \ell''$ , then for each length  $\ell''$  advice string,  $y$ , either  $N_{i'}(\langle j_{\ell''}, y \rangle)$  rejects (yet  $j_{\ell''} \in L$ ) or for some length  $\ell''$  string  $z$  that is, lexicographically, strictly greater than  $j_{\ell''}$ , it holds that  $N_{i'}(\langle z, y \rangle)$  accepts (yet  $z \notin L$ ).
2. If  $|j_{\ell''}| = \ell'' - 1$ , then  $L^{\ell''}$  is empty but for each advice string  $y \in \Sigma^{\ell''}$  there is an  $x \in \Sigma^{\ell''}$  such that  $N_{i'}(\langle x, y \rangle)$  accepts.

Thus,  $L \not\subseteq L'/n$ . This contradicts our supposition that  $L \in \text{NP}/n$  via NP language  $L'$ . □ Claim 3.15

□ Theorem 3.13

In fact, it is not hard to modify the proof of Theorem 3.13 to yield the following more general claim. We leave the proof to the reader. [Hint: Use the same “large gaps, and ‘brute force’ short strings” technique used in the proof of Theorem 3.13, but modify the gaps and the time complexity to be sensitive to the time bound  $h(n)$ .]

**Theorem 3.16** *Let  $h(n)$  be any recursive function.  $\text{P-sel} \not\subseteq \text{DTIME}[h(n)]/n$ .*

### 3.3 Unique Solutions Collapse the Polynomial Hierarchy

A central focus of computational complexity theory is whether large collections of computational objects can be thinned down. The Isolation Technique, the focus of Chap. 4, provides a probabilistic approach to this thinning process. In this section, we approach the issue of thinning from a different perspective. We ask whether, given a satisfiable boolean formula, a nondeterministic polynomial-time function can cull a single satisfying assignment from its set of satisfying assignments. We'll soon define exactly what is meant by a nondeterministic function, and will see that if this thinning could be done, then the polynomial hierarchy would collapse.

Nondeterministic polynomial-time functions can easily find *all* satisfying assignments of satisfiable formulas. Thus, we have the curious case—which at first may even seem paradoxical—that it is easier to find all solutions than to cull out one solution. This is not a paradox. We are dealing with *nondeterministic* functions. When they are multivalued their various values appear on distinct paths in a potentially very bushy tree (formal definitions follow soon). So “simply taking the smallest output and discarding the rest” cannot be done in any obvious way within the power of nondeterministic functions, as an individual path has no obvious way of telling whether the

satisfying assignment it obtained and is about to output is lexicographically smaller than whatever satisfying assignments its many sibling paths may have obtained.

To put into context the strength one needs to cull satisfying assignments, note that machines seemingly a bit stronger than NPTMs can cull satisfying assignments unconditionally, and deterministic polynomial-time machines cannot cull satisfying assignments unless  $P = NP$ . Both these claims, as formalized in the following proposition, are immediate. In particular, part 2 holds since an  $FP^{NP}$  function can even, by prefix search, find the lexicographically smallest satisfying assignment of a given formula.

**Proposition 3.17**

1.  $P = NP \iff$  *there is a polynomial-time computable function  $f$  such that*  

$$(\forall F \in SAT) [f(F) \text{ is a satisfying assignment of } F].$$
2. *There is a function  $f$  computable by a polynomial-time machine with an NP oracle (i.e.,  $f \in FP^{NP}$ ) such that*

$$(\forall F \in SAT) [f(F) \text{ is a satisfying assignment of } F].$$

Our case, whether NP functions can cull a single satisfying assignment, is intermediate (perhaps not strictly) in logical likelihood between the case of FP functions and the case of  $FP^{NP}$  functions.

Curiously enough, though the main result of this section does not seem on its face to be about semi-feasible sets, a nondeterministic analog of the semi-feasible sets plays a central role in the proof. In particular, the tournament divide and conquer approach of Sect. 3.1 will be central here, though in the context of nondeterministic selector functions. First, though, we state some definitions.

The class NPMV captures the notion of multivalued nondeterministic function computation. The class NPSV captures the notion of single-valued nondeterministic function computation.

**Definition 3.18**

1. *Let  $f$  be a multivalued function.  $\text{set-}f(x)$  denotes the set of all values that are an output of  $f(x)$ . Note that if  $f(x)$  has no output then  $\text{set-}f(x) = \emptyset$ .*
2. *We consider any given nondeterministic polynomial-time machine  $N$  to implicitly compute a (potentially partial) multivalued function, namely, the function  $f_N$  defined by  $\text{set-}f_N(x) = \{y \mid \text{some computation path of } N(x) \text{ outputs } y\}$ . NPMV denotes the class of functions computed in this sense by nondeterministic polynomial-time machines.*
3. *A (potentially partial) multivalued function  $f$  is said to be single-valued if  $(\forall x) [|\text{set-}f(x)| \leq 1]$ . NPSV denotes the class of all single-valued NPMV functions.*

Note that there is a function  $f \in \text{NPMV}$  such that, for all  $F \in \text{SAT}$ ,  $\text{set-}f(F)$  consists of all  $F$ 's satisfying assignments.

Our notion of “thinning” a multivalued function is the standard one: refinement. A refinement of multivalued function  $f$  is a function  $g$  with the same domain and containing a subset (possibly nonproper) of  $f$ 's values.

**Definition 3.19** *Given multivalued functions  $f$  and  $g$ , we say  $g$  is a refinement of  $f$  if*

1.  $(\forall x) [\text{set-}g(x) = \emptyset \iff \text{set-}f(x) = \emptyset]$ , and
2.  $(\forall x) [\text{set-}g(x) \subseteq \text{set-}f(x)]$ .

We now can state the main theorem of this section. If for each NPMV function there exists an NPSV function that is a refinement of the NPMV function, then the polynomial hierarchy collapses quite dramatically.

**Theorem 3.20** *If all NPMV functions have NPSV refinements, then  $\text{PH} = \text{ZPP}^{\text{NP}}$ .*

Since  $\text{ZPP}^A \subseteq \text{NP}^A$  for all  $A$ , we have the following corollary.

**Corollary 3.21** *If all NPMV functions have NPSV refinements, then  $\text{PH} = \text{NP}^{\text{NP}}$ .*

Lemma 3.23 connects the hypothesis of Theorem 3.20 to the equivalent hypothesis that nondeterministic polynomial-time machines can cull a single satisfying assignment for any input satisfiable formula.

We will need the following famous result, which we state without proof.

**Theorem 3.22 (Cook's Theorem)** *Let  $N_i$  be a standard enumeration of NPTMs. There is a function  $f_{\text{COOK}} \in \text{FP}$ , mapping from strings to boolean formulas, such that*

1.  $(\forall i) (\forall x) [N_i(x) \text{ accepts} \iff f_{\text{COOK}}(N_i, x) \text{ is satisfiable}]$ ,
2.  $(\exists g_{\text{COOK}} \in \text{FP}) (\forall i) (\forall x) [g_{\text{COOK}}(f_{\text{COOK}}(N_i, x)) = \langle N_i, x \rangle]$ , and
3.  $(\exists h_{\text{COOK}} \in \text{FP}) (\forall i) (\forall x) (\forall a) [\text{if } a \text{ is a satisfying assignment of } f_{\text{COOK}}(N_i, x), \text{ then } h_{\text{COOK}}(N_i, x, a) \text{ outputs an accepting computation path of } N_i(x)]$ .

**Lemma 3.23** *The following are equivalent:*

1. Every NPMV function has an NPSV refinement.
2.  $(\exists f \in \text{NPSV}) (\forall F \in \text{SAT}) [f(F) \text{ is a satisfying assignment of } F]$ .<sup>2</sup>

**Proof** Consider the NPMV function  $h$  defined by

$$\text{set-}h(F) = \{a \mid a \text{ is a satisfying assignment of } F\}.$$

<sup>2</sup> We use  $f(F)$  here as a shorthand, for those inputs on which  $F \in \text{SAT}$ , for the single element in the one-element set that comprises  $\text{set-}f(F)$ .



If every NPMV function has an NPSV refinement, then  $h$  does. This refinement satisfies part 2 of the lemma.

Suppose  $(\exists f \in \text{NPSV}) (\forall F \in \text{SAT}) [f(F) \text{ is a satisfying assignment of } F]$ . Let  $\hat{f}$  be one such  $f$ .

Let  $g \in \text{NPMV}$ , via function-computing NPTM  $N_i$ . Without loss of generality let  $N_i$  be such that paths that output values are (always) accepting paths (i.e., their final state is an accepting state), and paths that do not output values are (always) rejecting paths (i.e., their final state is not an accepting state); since  $N_i$  is a function-computing machine, this addition of accepting final states may seem pointless, but in fact it will be useful in letting us employ the machinery of Cook's Theorem, which links accepting paths to satisfying assignments. Let  $f_{\text{COOK}}$  and  $h_{\text{COOK}}$  be as in Theorem 3.22. We now give an NPTM  $N$  computing an NPSV refinement of  $g$ . On input  $x$ ,  $N$  deterministically computes  $f_{\text{COOK}}(N_i, x)$  and then nondeterministically guesses a path of NPSV function  $\hat{f}$ . If along our guessed path  $\hat{f}$  has no output then we will make no output along the current path. If along our guessed path  $\hat{f}$  does have an output, call it  $\alpha$ , then we along the current path first check whether  $\alpha$  is a satisfying assignment of  $f_{\text{COOK}}(N_i, x)$ . If  $\alpha$  is not a satisfying assignment of  $f_{\text{COOK}}(N_i, x)$  (a strange situation that might in fact occur, due to Lemma 3.23's part 2 being silent on the issue of what  $f$  does, aside from having at most one output value, when  $F \notin \text{SAT}$ ), then we make no output on the current path. If  $\alpha$  is a satisfying assignment of  $f_{\text{COOK}}(N_i, x)$ , then on our current path we deterministically compute  $h_{\text{COOK}}(N_i, x, \alpha)$ —call this value *path*—and then we deterministically compute what value is output along computation path *path* of the computation of  $N_i(x)$  and output that value along our current path.  $\square$

We will use two key lemmas—Lemmas 3.25 and 3.27—and one new definition in the proof of Theorem 3.20. The new definition extends to partial nondeterministic functions the notion of semi-feasible computation.

**Definition 3.24** *Let  $\mathcal{F}$  be any (possibly partial, possibly multivalued) function class. We say a set  $L$  is  $\mathcal{F}$ -selective if there is a multivalued function  $f \in \mathcal{F}$  such that*

1.  $(\forall x, y) [\text{set-}f(x, y) \subseteq \{x, y\}]$ , and
2.  $(\forall x, y) [(x \in L \vee y \in L) \implies \emptyset \neq \text{set-}f(x, y) \subseteq L]$ .

That is, selector functions, in this generalized sense, (1) never choose strings that are not among their arguments, and (2) if at least one of the function's arguments is in the set then they must choose at least one argument and they must choose no argument that is not in the set. Note that what is typically referred to in the literature by the notation “P-selective” (i.e., the semi-feasible) would be referred to as “ $\text{FP}_{\text{total}}$ -selective” were one to rigidly follow the notation of Definition 3.24.

**Lemma 3.25**  $\text{NPSV-sel} \cap \text{NP} \subseteq (\text{NP} \cap \text{coNP})/\text{poly}$ .

We defer the proof of the above lemma until after the proof of the main theorem, Theorem 3.20.

Theorem 1.16, the Karp–Lipton Theorem, states that if NP has sparse  $\leq_T^p$ -hard sets (equivalently and more to the point here, if  $\text{NP} \subseteq \text{P/poly}$ ), then the polynomial hierarchy collapses to  $\text{NP}^{\text{NP}}$ . Can one establish the same strong conclusion from the weaker hypothesis that  $\text{NP} \subseteq (\text{NP} \cap \text{coNP})/\text{poly}$ ? Not only is the answer yes, but this stronger result is in fact essentially implicit in the earlier result, via relativization. Such a situation is referred to, only half-jokingly, as “proof by relativization.” We now give such a proof for the stronger result.

**Lemma 3.26**  $\text{NP} \subseteq (\text{NP} \cap \text{coNP})/\text{poly} \implies \text{PH} = \text{NP}^{\text{NP}}$ .

**Proof** By Theorem 1.16,

$$\text{NP} \subseteq \text{P/poly} \implies \text{PH} = \text{NP}^{\text{NP}}.$$

Also, it is true that this result relativizes, i.e.,<sup>3</sup>

$$(\forall A) [\text{NP}^A \subseteq \text{P}^A/\text{poly} \implies \text{PH}^A = \text{NP}^{\text{NP}^A}]. \quad (3.9)$$

Assume  $\text{NP} \subseteq (\text{NP} \cap \text{coNP})/\text{poly}$ . So  $\text{SAT} \in (\text{NP} \cap \text{coNP})/\text{poly}$ , say via  $\text{NP} \cap \text{coNP}$  set  $B$ . By equation 3.9, taking  $A = B$ , we have

$$\text{NP}^B \subseteq \text{P}^B/\text{poly} \implies \text{PH}^B = \text{NP}^{\text{NP}^B}. \quad (3.10)$$

However,  $\text{NP} \subseteq \text{NP}^B \subseteq \text{NP}^{\text{NP}} \cap \text{coNP} = \text{NP}$  (Sect. A.4) and  $\text{P}^B \subseteq \text{P}^{\text{NP}} \cap \text{coNP} = \text{NP} \cap \text{coNP}$  (Sect. A.4). So by our  $\text{NP} \subseteq (\text{NP} \cap \text{coNP})/\text{poly}$  assumption, the hypothesis of equation 3.10 holds for  $B$ . (One can alternatively see that, under our assumption, the hypothesis of equation 3.10 holds via noting that if  $\text{SAT} \in B/\text{poly}$  then  $\text{NP} \subseteq R_m^p(B/\text{poly}) \subseteq \text{P}^B/\text{poly}$ .) Thus, by equation 3.10,  $\text{PH}^B = \text{NP}^{\text{NP}^B}$ . However, since  $B \in \text{NP} \cap \text{coNP}$ ,  $\text{PH}^B = \text{PH}$  and  $\text{NP}^{\text{NP}^B} = \text{NP}^{\text{NP}}$ . Thus,  $\text{NP} \subseteq (\text{NP} \cap \text{coNP})/\text{poly} \implies \text{PH} = \text{NP}^{\text{NP}}$ .  $\square$

<sup>3</sup> We leave it as an exercise to the reader to verify this. The *proof* given in this book for the Karp–Lipton Theorem actually does not relativize, as it is based on the concrete set SAT. To relativize the proof cleanly one, however, may employ the same idea refocused onto an NP-complete set that, like SAT, is self-reducible, but that is also nicely relativizable. A good example of such a set is the set  $L_A$  that we will now define. Let  $\hat{N}_1, \hat{N}_2, \dots$  be a standard enumeration of NP (oracle) Turing machines index such that each machine  $\hat{N}_i$  is  $i+n^i$  time-bounded on every oracle. Such enumerations do to exist. Define  $L_A = \{\hat{N}_i \# x \# \text{pre} \# \text{pad} \# \text{pad} \mid (\hat{N}_i^A(x) \text{ has an accepting path such that } \text{pre} \text{ is a prefix of the guess bits along that accepting path}) \text{ and } (|\text{prefix}| + |\text{pad}| = i + n^i)\}$ . Note that  $L_A$  is not just  $\leq_m^p$ -complete for  $\text{NP}^A$  but even is  $\leq_m^p$ -complete for  $\text{NP}^A$ , for each  $A$ .  $L_A$  is also (2-disjunctively) self-reducible; the doubling of *pad* ensures that we decrease in length within the self-reduction.

We'll employ the following slightly stronger form, based on Theorem 1.17, of this result.

**Lemma 3.27**  $\text{NP} \subseteq (\text{NP} \cap \text{coNP})/\text{poly} \implies \text{PH} = \text{ZPP}^{\text{NP}}$ .

We now can prove the main theorem.

**Proof of Theorem 3.20** Assume that all NPMV functions have NPSV refinements. Consider the multivalued function  $f_{\text{SAT}}$  such that, for each  $x$  and  $y$ ,

$$\text{set-}f_{\text{SAT}}(x, y) = \{x, y\} \cap \text{SAT}.$$

This function is in NPMV, as it is computed by a nondeterministic polynomial-time machine that nondeterministically chooses  $x$  or  $y$ , and then nondeterministically chooses a variable assignment to the chosen formula, and then outputs the chosen formula if that assignment satisfies the formula.

By hypothesis, all NPMV functions have NPSV refinements. Let  $g_{\text{SAT}}$  be an NPSV refinement of  $f_{\text{SAT}}$ . Note that  $g_{\text{SAT}}$  is defined exactly if at least one of its arguments is in SAT,  $g_{\text{SAT}}$  is an NPSV function, and if at least one of  $g_{\text{SAT}}$ 's arguments is in SAT then  $g_{\text{SAT}}$  outputs an argument that is in SAT. Indeed,  $g_{\text{SAT}}$  is an NPSV-selector function for SAT. Thus,  $\text{SAT} \in \text{NPSV-sel}$ . Note that

$$\{L \mid (\exists B \in \text{NPSV-sel}) [L \leq_m^p B]\}$$

clearly equals NPSV-sel, as such an  $L$  has the NPSV-selector function  $f_L$  defined, for all  $x$  and  $y$ , by:

$$\text{set-}f_L(x, y) = \begin{cases} \{x\} & \text{if } f_B(g(x), g(y)) = g(x) \\ \{y\} & \text{if } f_B(g(x), g(y)) = g(y) \\ \emptyset & \text{otherwise,} \end{cases}$$

where  $f_B$  is the NPSV-selector function for  $B$  and  $g$  is the many-one reduction from  $L$  to  $B$ . Since each NP set  $\leq_m^p$ -reduces to SAT, it follows that  $\text{NP} \subseteq \text{NPSV-sel} \cap \text{NP}$ . So by Lemma 3.25  $\text{NP} \subseteq (\text{NP} \cap \text{coNP})/\text{poly}$ . Thus, by Lemma 3.27,  $\text{PH} = \text{ZPP}^{\text{NP}}$ .  $\square$  Theorem 3.20

We now turn to proving the key lemma, Lemma 3.25. There is something a wee bit surprising about this lemma. In particular, one might expect only the potentially weaker<sup>4</sup> (and also true) claim

$$\text{NPSV-sel} \cap \text{NP} \subseteq (\text{NP}/\text{poly}) \cap (\text{coNP}/\text{poly}).$$

This is because sets in NPSV-sel have selector functions that are merely partial, and partial functions usually confound  $(\text{NP} \cap \text{coNP})/\text{poly}$ -type proofs because to prove that a set is in  $(\text{NP} \cap \text{coNP})/\text{poly}$  requires a set that is  $\text{NP} \cap \text{coNP}$ -like for all advice strings—not just the correct advice string.

<sup>4</sup> Though clearly  $(\text{NP} \cap \text{coNP})/\text{poly} \subseteq (\text{NP}/\text{poly}) \cap (\text{coNP}/\text{poly})$ , it remains an open question whether  $(\text{NP} \cap \text{coNP})/\text{poly} = (\text{NP}/\text{poly}) \cap (\text{coNP}/\text{poly})$ .

The proof of Lemma 3.25 finesses this by combining the divide and conquer idea behind this chapter's GEM section with a trick: requiring membership proofs to be part of the advice. This, plus the fact that NPSV-selector functions are only partially partial—the definition of selectivity requires them to be defined whenever at least one of their arguments is in the set for which they are selectors—suffices to prove the result.

**Proof of Lemma 3.25** Let  $L$  be an arbitrary set in  $\text{NPSV-sel} \cap \text{NP}$ . Let  $N_L$  be an NPTM accepting  $L$ . Let  $f \in \text{NPSV}$  be a selector function for  $L$ . Without loss of generality, assume

$$(\forall x, y) [\text{set-}f(x, y) = \text{set-}f(y, x)].$$

We specify an advice interpreter  $A \in \text{NP} \cap \text{coNP}$  and an advice function  $g$  showing that  $L \in (\text{NP} \cap \text{coNP})/\text{poly}$ .

$$\begin{aligned} A = \{ \langle x, \langle \langle a_1, a_2, \dots, a_z \rangle, \langle w_1, w_2, \dots, w_z \rangle \rangle \rangle \mid \\ z = z' \text{ and} \\ (\forall i : 1 \leq i \leq z) [w_i \text{ is an accepting path of } N_L(a_i)] \text{ and} \\ (\exists i : 1 \leq i \leq z) [x \in \text{set-}f(x, a_i)] \}. \end{aligned}$$

Clearly,  $A \in \text{NP}$ .  $\bar{A}$  is also in NP, as the following NPTM  $N$  accepts  $\bar{A}$ .  $N$  accepts immediately if the input is syntactically ill-formed or if  $z \neq z'$ . Otherwise,  $N$  deterministically checks whether  $(\forall i : 1 \leq i \leq z) [w_i \text{ is an accepting path of } N_L(a_i)]$  and  $N$  immediately accepts if this check fails. Otherwise,  $N$  rejects immediately if  $x \in \{a_1, a_2, \dots, a_z\}$ . Otherwise, note that  $\{a_1, a_2, \dots, a_z\} \subseteq L$ , as  $N$  has seen and checked membership certificates for each  $a_i$ . So, by the definition of an NPSV-selector function, for each  $i$  it holds that  $|\text{set-}f(a_i, x)| = 1$ .  $N$  now nondeterministically guesses and checks the unique value of  $\text{set-}f(a_i, x)$  for all  $i$ . The nondeterministic path(s) that correctly guess and check for all  $i$  which of  $x$  and  $a_i$  is the unique element in  $\text{set-}f(a_i, x)$  accept if and only if for all  $i$  it holds that  $\text{set-}f(a_i, x) = \{a_i\}$ . This completes our NP algorithm for  $\bar{A}$ .

Our advice function is as follows. At each length  $n$ , consider  $L^n$ . NPSV-selector function  $f$  induces a tournament on  $L^n$  as follows. By the definition of an NPSV-selector function, for each  $a, b \in L^n$ ,  $a \neq b$ , exactly one of  $a \in \text{set-}f(a, b)$  and  $b \in \text{set-}f(a, b)$  holds, so  $f$  induces a tournament in the same fashion as in Sect. 3.1: for  $a, b \in L^n$ ,  $a \neq b$ , edge  $(a, b)$  will be in our tournament if and only if  $\text{set-}f(a, b) = \{b\}$ . By Theorem 3.1, there is a set  $H_n \subseteq \Sigma^n$ ,  $|H_n| \leq n$ , such that each element  $y$  of  $L^n$  either is in  $H_n$  or for some  $h \in H_n$  satisfies  $\text{set-}f(y, h) = y$ . Our advice string for length  $n$  will be  $\langle \langle h_1, h_2, \dots, h_z \rangle, \langle w_1, w_2, \dots, w_z \rangle \rangle$ , where  $\langle h_1, h_2, \dots, h_z \rangle = H$  and each  $w_i$  is an accepting path of  $N_L(h_i)$ . This advice function is polynomially length-bounded.

The set  $A$  and this advice function indeed do prove that  $L \in (\text{NP} \cap \text{coNP})/\text{poly}$ , as the interpreter  $A$ —with this advice function—will at each length  $n$  accept exactly those strings that are in  $H_n$  or that defeat

a string in  $H_n$ , and by Theorem 3.1 and the properties of NPSV-selector functions, this describes exactly  $L^n$ .  $\square$  Lemma 3.25

One must distinguish between Theorem 3.20 and the following seemingly similar question, which remains open.

**Open Question 3.28** *Does  $UP = NP$  imply that the polynomial hierarchy collapses?*

The distinction between this question and Theorem 3.20 is that NPSV machines output at most one value, but this value may appear on many paths. Indeed, it is not known either that  $UP = NP$  if all NPMV functions have NPSV refinements, or that  $UP = NP$  only if all NPMV functions have NPSV refinements.

### 3.4 OPEN ISSUE: Are the Semi-feasible Sets in P/linear?

Are the semi-feasible sets contained in P/linear? Note that from Sects. 3.1 and 3.2 we know

$$P\text{-sel} \subseteq NP/\text{linear} \cap P/\text{quadratic}.$$

Seeking the best of both worlds, we might wonder whether  $P\text{-sel} \subseteq P/\text{linear}$  can be established.

We suspect that  $P\text{-sel} \not\subseteq P/\text{linear}$ . However, to prove this might be challenging, as proving this implies  $P \neq NP$ . This is so since  $P\text{-sel} \subseteq NP/\text{linear}$ , so if  $P = NP$  then  $P\text{-sel} \subseteq P/\text{linear}$ . On the other hand, it remains plausible that, via some clever algorithm, one can unconditionally prove that  $P\text{-sel} \subseteq P/\text{linear}$ .

**Open Question 3.29**  $P\text{-sel} \subseteq P/\text{linear}?$

### 3.5 Bibliographic Notes

Section 3.1 is based on the work of Ko [Ko83]. In that paper, Ko establishes not just that  $P\text{-sel} \subseteq P/\text{poly}$ , but even that a slightly more general class, known as the weakly P-selective sets, is also contained in P/poly. Later work by Amir, Beigel, and Gasarch [ABG00], Hemaspaandra et al. [HJRW97], and Ogiwara [Ogi95b] shows that P/poly contains other broad generalizations of the semi-feasible sets.

Section 3.2 is based on the work of Hemaspaandra and Torenvliet [HT96] and Hemaspaandra, Nasipak, and Parkins [HNP98], except Theorem 3.7, which is implicit in a proof of Hemaspaandra et al. [HNOS96a], and Theorem 3.9, a standard fact from graph theory first noted in the

1950s ([Lan53], see also [Wes96]). An incomparable but related result has been proven by Burtchick and Lindner [BL97]. They prove that  $R_{\mathcal{O}(n)-T}^P(P\text{-sel}) \subseteq E/\text{linear}$ .

Section 3.3 is based on the work of Hemaspaandra et al. [HNOS96b]. Book, Long, and Selman ([BLS84], see also [BLS85, Sel94]) first introduced the function classes NPSV and NPMV (Definition 3.18). Refinements (Definition 3.19) have been carefully studied by Selman [Sel94]. Lemma 3.23 is also due to Selman [Sel94]. Semi-feasibility (selectivity) was extended to nondeterministic total functions by Hemaspaandra et al. [HHN<sup>+</sup>95], and was extended to nondeterministic partial functions (Definition 3.24) by Hemaspaandra et al. [HNOS96b]. Wang [Wan95] has studied extending semi-feasibility to counting classes. Though in this chapter we use “semi-feasible” as a synonym for “P-selective,” we mention that in the literature “semi-feasible” is often used in a broader sense that encompasses such nondeterministic and other analogs of P-selectivity. Lemma 3.26 was first stated, with somewhat complex direct proofs, by Abadi, Feigenbaum, and Kilian [AFK89] and Kämper [Käm91]; the fact that it is implicit in the original Karp–Lipton result [KL80] was noted by Hemaspaandra et al. [HHN<sup>+</sup>95], whose proof we follow here.

Lemma 3.27 is due to Köbler and Watanabe [KW98]. Related to the work of Cai mentioned in the Bibliographic Notes of Chap. 1, from the hypothesis of Lemma 3.27 one can even conclude that  $(S_2^P)^{NP} \cap \text{coNP} = \text{PH}$  [CCHO01], which in light of the fact that  $(S_2^P)^{NP} \cap \text{coNP} \subseteq \text{ZPP}^{NP}$  ([Cai01], see also [CCHO01]), is at least as strong as Lemma 3.27.

Theorem 3.22, Cook’s Theorem, is due to Cook [Coo71], though it is stated here in a relatively strong form. Levin [Lev75] independently discovered Cook’s Theorem, and thus it sometimes is referred to in the literature as the Cook–Levin Theorem or, in light of the contributions of Karp [Kar72], as the Cook–Karp–Levin Theorem. The interesting issue of whether a Theorem 3.20-like result holds for  $\text{FP}_{tt}^{NP}$  remains open. That is, it is not known whether: If every NPMV function has a refinement computable via polynomial-time truth-table access to NP, then the polynomial hierarchy collapses. It is known that the statement “every NPMV function has a refinement computable via polynomial-time truth-table access to NP” fails relative to some oracles ([IT89], see [BT96a]) yet holds relative to a random oracle ([WT93] and the proof is based on the Isolation Technique of Chap. 4). Ogiwara [Ogi96a] has shown that if every NPMV function has a refinement computable via polynomial-time *sublinear*-truth-table access to NP, then the polynomial hierarchy collapses. The issue of footnote 4, i.e., whether  $(\text{NP} \cap \text{coNP})/\text{poly}$  equals  $(\text{NP}/\text{poly}) \cap (\text{coNP}/\text{poly})$ , has been studied by Gavaldà and Balcázar ([GB91], see also [CHW99]). Though the question remains open, Gavaldà and Balcázar do give a structural consequence that would follow from this equality.

Theorem 3.20, which is due to Hemaspaandra et al. [HNOS96b], states that the polynomial hierarchy collapses if each NPMV function has an NPSV refinement. Can more be said? In fact, Hemaspaandra et al. [HNOS96b] prove—as implicitly does the proof in this chapter—the stronger result that the polynomial hierarchy collapses if each NP $k$ V function has an NPSV refinement (where the  $k$  in NP $k$ V means “at most  $k$  distinct values on any input”). Other papers have continued to explore what refinement assumptions imply polynomial hierarchy collapses. In particular, Ogihara [Ogi96a] proved that if NPMV functions have NPFewV refinements then the polynomial hierarchy collapses, where NPFewV (a class first studied, under different names, by Book, Long, and Selman [BLS84], see also [BLS85,Sel94]) indicates the class of NPMV functions such that for some polynomial  $q$  it holds that  $(\forall x) [|\text{set-}f(x)| \leq q(|x|)]$ . Naik et al. [NRRS98] proved, for each  $k$ , that if all NP $(k+1)$ V functions have NP $k$ V refinements then the polynomial hierarchy collapses. Taking an even broader view, Hemaspaandra, Ogihara, and Wechsung [HOW00] prove a sufficient condition for when numbers of solutions of NP functions can be reduced, and Kosub [Kos00] has shown that, for finite “solution types,” their condition in fact describes every type of solution reduction that holds in all relativized worlds. Hemaspaandra, Ogihara, and Wechsung [HOW00] also put these collapse results into an interesting perspective via proving general lowness results implying the collapses.





## 4. The Isolation Technique

Brother: *And the Lord spake, saying, "First shalt thou take out the Holy Pin. Then, shalt thou count to three, no more, no less. Three shalt be the number thou shalt count, and the number of the counting shalt be three. Four shalt thou not count, nor either count thou two, excepting that thou then proceed to three. Five is right out. Once the number three, being the third number, be reached, then lobbest thou thy Holy Hand Grenade of Antioch towards thy foe, who being naughty in my sight, shall snuff it."*

Maynard: *Amen.*

All: *Amen.*

Arthur: *Right! One... two... five!*

—*Monty Python and the Holy Grail*

Counting is cumbersome and sometimes painful. Studying NP would indeed be far simpler if all NP languages were recognized by NP machines having at most one accepting computation path, that is, if  $\text{NP} = \text{UP}$ . The question of whether  $\text{NP} = \text{UP}$  is a nagging open issue in complexity theory. There is evidence that standard proof techniques can settle this question neither affirmatively nor negatively. However, surprisingly, with the aid of randomness we will relate NP to the problem of detecting unique solutions. In particular, we can reduce, with high probability, the entire collection of accepting computation paths of an NP machine to a single path, provided that initially there is at least one accepting computation path. We call such a reduction method an isolation technique.

In this chapter we present one such technique. Based on this technique, we prove two surprising results relating NP and NL to counting complexity classes: PP is polynomial-time Turing hard for the polynomial hierarchy, and NL and UL are equal in the presence of polynomially length-bounded advice functions.

The organization of this chapter is as follows. In Sect. 4.1, we present the isolation technique, and show that NP is “randomized reducible” to the problem of detecting unique solutions. More precisely, for each language  $L$  in NP, there exist a randomized polynomial-time algorithm  $\mathcal{F}$  and an NP-decision problem  $A$ , such that for every string  $x$ , if  $x$  is a member of  $L$ , then with high probability the output of  $\mathcal{F}$  on input  $x$  is an instance of  $A$  with a

unique solution, and if  $x$  is not a member of  $L$ , then with probability  $\frac{1}{|x|^{\Theta(1)}}$  the output of  $\mathcal{F}$  on  $x$  is an instance of  $A$  with zero solutions or more than one solution.

In Sect. 4.2, we apply the isolation technique for NP to prove Toda's Theorem,  $\text{PH} \subseteq \text{P}^{\text{PP}}$ , and we also establish a well-known extension of Toda's Theorem. In Sect. 4.3, we prove that  $\text{NL/poly} = \text{UL/poly}$ .

## 4.1 GEM: Isolating a Unique Solution

The isolation technique we use in this chapter is often called the Isolation Lemma.

### 4.1.1 The Isolation Lemma

A *weight function* over a finite set  $\mathcal{U}$  is a mapping from  $\mathcal{U}$  to the set of positive integers. We naturally extend any weight function over  $\mathcal{U}$  to one on the power set  $2^{\mathcal{U}}$  as follows. For each  $S \subseteq \mathcal{U}$ , the weight of  $S$  with respect to a weight function  $W$ , denoted by  $W(S)$ , is  $\sum_{x \in S} W(x)$ . Let  $\mathcal{F}$  be a nonempty family of nonempty subsets of  $\mathcal{U}$ . Call a weight function  $W$  *good for  $\mathcal{F}$*  if there is exactly one minimum-weight set in  $\mathcal{F}$  with respect to  $W$ . Call  $W$  *bad for  $\mathcal{F}$*  otherwise.

**Lemma 4.1 (The Isolation Lemma)** *Let  $\mathcal{U}$  be a finite set. Let  $\mathcal{F}_1, \dots, \mathcal{F}_m$  be families of nonempty subsets over  $\mathcal{U}$ , let  $D = \|\mathcal{U}\|$ , let  $R > mD$ , and let  $\mathcal{Z}$  be the set of all weight functions whose weights are at most  $R$ . Let  $\alpha$ ,  $0 < \alpha < 1$ , be such that  $\alpha > \frac{mD}{R}$ . Then more than  $(1 - \alpha)\|\mathcal{Z}\|$  functions in  $\mathcal{Z}$  are good for all of  $\mathcal{F}_1, \dots, \mathcal{F}_m$ .*

**Proof** Let  $\mathcal{F}$  be one family. For a weight function  $W \in \mathcal{Z}$ , let  $\text{MinWeight}_W$  denote the minimum weight of  $\mathcal{F}$  with respect to  $W$ , i.e.,  $\text{MinWeight}_W = \min\{W(S) \mid S \in \mathcal{F}\}$ , and let  $\text{MinWeightSet}_W$  denote the set of all minimum-weight sets of  $\mathcal{F}$  with respect to  $W$ , i.e.,  $\text{MinWeightSet}_W = \{S \in \mathcal{F} \mid W(S) = \text{MinWeight}_W\}$ . For  $x \in \mathcal{U}$ , we say that the minimum-weight sets of  $\mathcal{F}$  with respect to  $W$  are unambiguous about inclusion of  $x$  if there exist some  $S, S' \in \text{MinWeightSet}_W$  such that  $x \in (S \setminus S') \cup (S' \setminus S)$ .

Recall that a weight function  $W \in \mathcal{Z}$  is bad for  $\mathcal{F}$  if  $\|\text{MinWeightSet}_W\| \geq 2$ . Suppose that  $W$  is bad for  $\mathcal{F}$ . Let  $S$  and  $S'$  be two distinct members of  $\text{MinWeightSet}_W$ . Since  $S \neq S'$  there exists some  $x \in \mathcal{U}$  such that  $x$  belongs to the symmetric difference of  $S$  and  $S'$ , i.e.,  $(S \setminus S') \cup (S' \setminus S)$ . Thus, the minimum-weight sets of  $\mathcal{F}$  with respect to  $W$  are ambiguous about some  $x \in \mathcal{U}$ . Conversely, if the minimum-weight sets of  $\mathcal{F}$  with respect to  $W$  are ambiguous about some  $x \in \mathcal{U}$ , then there is more than one minimum-weight sets of  $\mathcal{F}$  with respect to  $W$ , so  $W$  is bad. Thus,  $W$  is bad if and only if there

is some  $x \in \mathcal{U}$  such that the minimum-weight sets of  $\mathcal{F}$  with respect to  $W$  are ambiguous about inclusion of  $x$ .

Let  $x \in \mathcal{U}$  be fixed. We count the number of weight functions  $W \in \mathcal{Z}$  such that the minimum-weight sets of  $\mathcal{F}$  with respect to  $W$  are ambiguous about inclusion of  $x$ . Let  $y_1, \dots, y_{D-1}$  be an enumeration of  $\mathcal{U} - \{x\}$  and  $v_1, \dots, v_{D-1} \in \{1, \dots, R\}$ . Let  $\mathcal{A}$  be the set of all weight functions  $W$  such that for all  $i$ ,  $1 \leq i \leq D-1$ ,  $W(y_i) = v_i$ . Suppose that there is a weight function  $W$  in  $\mathcal{A}$  such that the minimum-weight sets of  $\mathcal{F}$  with respect to  $W$  are ambiguous about inclusion of  $x$ . Let  $W'$  be an arbitrary element in  $\mathcal{A} \setminus \{W\}$  and  $\delta = W'(x) - W(x)$ . We claim that the minimum-weight sets of  $\mathcal{F}$  with respect to  $W'$  are unambiguous about inclusion of  $x$ . To see why, first suppose that  $\delta > 0$ . Then, for all  $S \in \mathcal{F}$ ,  $W'(S) = W(S) + \delta$  if  $x \in S$  and  $W'(S) = W(S)$  otherwise. In particular, for all  $S \in \text{MinWeightSet}_W$ ,  $W'(S) = W(S) + \delta$  if  $x \in S$  and  $W'(S) = W(S)$  otherwise. This implies that  $\text{MinWeight}_{W'} = \text{MinWeight}_W$  and  $\text{MinWeightSet}_{W'} = \{S \in \text{MinWeightSet}_W \mid x \notin S\}$ . Next suppose that  $\delta < 0$ . Then, for all  $S \in \mathcal{F}$ ,  $W'(S) = W(S) - |\delta|$  if  $x \in S$  and  $W'(S) = W(S)$  otherwise. In particular, for all  $S \in \text{MinWeightSet}_W$ ,  $W'(S) = W(S) - |\delta|$  if  $x \in S$  and  $W'(S) = W(S)$  otherwise. This implies that  $\text{MinWeight}_{W'} = \text{MinWeight}_W - |\delta|$  and  $\text{MinWeightSet}_{W'} = \{S \in \text{MinWeightSet}_W \mid x \in S\}$ . Thus, if  $\delta > 0$  then all minimum-weight sets of  $\mathcal{F}$  with respect to  $W'$  contain  $x$ , and if  $\delta < 0$  then no minimum-weight sets of  $\mathcal{F}$  with respect to  $W'$  contain  $x$ . Hence, for all  $W' \in \mathcal{A} \setminus \{W\}$  the minimum-weight sets of  $\mathcal{F}$  with respect to  $W'$  are unambiguous about inclusion of  $x$ . This implies that there is at most one weight function  $W \in \mathcal{A}$  such that the minimum-weight sets of  $\mathcal{F}$  with respect to  $W$  are ambiguous about inclusion of  $x$ . For each  $i$ ,  $1 \leq i \leq D-1$ , there are  $R$  choices for  $v_i$ . So, there are at most  $R^{D-1}$  weight functions  $W \in \mathcal{Z}$  such that the minimum-weight sets of  $\mathcal{F}$  with respect to  $W$  are ambiguous about inclusion of  $x$ . There are  $R^D$  weight functions in  $\mathcal{Z}$ , there are  $m$  choices for  $\mathcal{F}$ , and there are  $D$  choices for  $x$ . Thus, the proportion of  $\{W \in \mathcal{Z} \mid \text{for some } i, 1 \leq i \leq m, W \text{ is bad for } \mathcal{F}_i\}$  is at most  $\frac{mDR^{D-1}}{R^D} = \frac{mD}{R} < \alpha$ . So, the proportion of  $\{W \in \mathcal{Z} \mid \text{for all } i, 1 \leq i \leq m, W \text{ is good for } \mathcal{F}_i\}$  is more than  $1 - \alpha$ .  $\square$

#### 4.1.2 NP Is Randomized Reducible to US

US is the class of languages  $L$  for which there exists a nondeterministic polynomial-time Turing machine  $N$  such that, for every  $x \in \Sigma^*$ ,  $x \in L$  if and only if  $N$  on input  $x$  has exactly one accepting computation path (see Sect. A.9). USAT is the set of all boolean formulas having exactly one satisfying assignment and that USAT is complete for US under polynomial-time many-one reductions (see Sect. A.9).

We say that a language  $A$  is randomized reducible to a language  $B$ , denoted by  $A \leq_{\text{randomized}} B$ , if there exist a probabilistic polynomial-time Turing machine  $M$  and a polynomial  $p$  such that, for every  $x \in \Sigma^*$ ,

- if  $x \in A$ , then the probability that  $M$  on input  $x$  outputs a member of  $B$  is at least  $\frac{1}{p(|x|)}$ , and
- if  $x \notin A$ , then the probability that  $M$  on input  $x$  outputs a member of  $B$  is zero.

Using the Isolation Lemma, one can show that every language in NP is randomized reducible to USAT.

**Theorem 4.2**  $(\forall L \in \text{NP})[L \leq_{\text{randomized}} \text{USAT}]$ .

To prove this theorem and other results in this chapter, we will use a pairing function with a special property regarding the encoding length. For binary strings  $x, y, \dots, z$ , we write  $x\#y\#\dots\#z$  to denote the string constructed from this expression by replacing each occurrence of 0, 1, and  $\#$  by 00, 01, and 11, respectively. More precisely, the encoding is the binary string of the form

$$0x_1 \dots 0x_{|x|} 110y_1 \dots 0y_{|y|} 11 \dots 0z_1 \dots 0z_{|z|}.$$

Note that this pairing function satisfies the following conditions:

- If  $x_1, x_2, \dots, x_k$  and  $y_1, y_2, \dots, y_k$  satisfy  $|x_1| + |x_2| + \dots + |x_k| = |y_1| + |y_2| + \dots + |y_k|$ , then  $|x_1\#x_2\#\dots\#x_k| = |y_1\#y_2\#\dots\#y_k|$ .
- For every  $x, y, \dots, z \in \Sigma^*$ , we can recover  $x, y, \dots, z$  from  $x\#y\#\dots\#z$  in time polynomial in  $|x\#y\#\dots\#z|$ .

**Proof of Theorem 4.2** Let  $L$  be a language in NP. Let  $p$  be a polynomial and  $A$  a language in P such that, for all  $x \in \Sigma^*$ ,

$$x \in L \iff (\exists y \in \Sigma^{p(|x|)}) [\langle x, y \rangle \in A].$$

We may assume that for all  $n \geq 0$ ,  $p(n) \geq 1$ , and that for all  $x$ ,  $\langle x, 0^{p(|x|)} \rangle \notin A$ . For each  $n \geq 1$ , let  $\mu(n)$  be the smallest power of 2 that is greater than or equal to  $p(n)$ . Define

$$A' = \{ \langle x, y \rangle \mid |y| = \mu(|x|) \wedge (\exists u, v)[|u| = p(|x|) \wedge uv = y \wedge \langle x, y \rangle \in A] \}.$$

Then  $A' \in \text{P}$  and for every  $x \in \Sigma^*$ ,

$$x \in L \iff (\exists y \in \Sigma^{\mu(|x|)}) [\langle x, y \rangle \in A'].$$

Since for all  $x$ ,  $\langle x, 0^{p(|x|)} \rangle \notin A$ , for all  $x$ ,  $\langle x, 0^{\mu(|x|)} \rangle \notin A'$ .

For each  $n \geq 1$ , we can specify each string  $y \in \Sigma^{\mu(n)}$  by bit positions at which  $y$  has a 1; i.e.,  $y$  can be specified via the set  $\{i \mid 1 \leq i \leq p(n) \wedge y_i = 1\}$ .

For each  $n \geq 1$ , let  $\mathcal{U}(n) = \{1, \dots, \mu(n)\}$ . Then for every  $n \geq 1$  the power set of  $\mathcal{U}(n)$  represents  $\Sigma^{\mu(n)}$ ; i.e., each element in  $\Sigma^{\mu(n)}$  can be viewed as a subset of  $\mathcal{U}(n)$ . For each  $x \in \Sigma^*$ , let  $\mathcal{F}(x)$  be the set of all  $y \subseteq \mathcal{U}(|x|)$  such that  $\langle x, y \rangle \in A'$ . By our assumption, for every  $x \in \Sigma^*$ , the empty set (which corresponds to the string  $0^{\mu(|x|)}$ ) is not in  $\mathcal{F}(x)$ . For each  $n \geq 1$ , let  $\mathcal{Z}(n)$  be the family of all the weight functions that assign to each number  $i$ ,  $1 \leq i \leq p(n)$ , a positive weight of at most  $4\mu(|x|)$ .

Let  $x$  be an arbitrary string. Apply Lemma 4.1 with  $m = 1$ ,  $\mathcal{U} = \mathcal{U}(|x|)$ ,  $\mathcal{Z} = \mathcal{Z}(|x|)$ ,  $\mathcal{F}_1 = \mathcal{F}(x)$ , and  $R = 4\mu(|x|)$ . Then,

- if  $x \in L$ , then the fraction of the weight functions in  $\mathcal{Z}(|x|)$  with respect to which  $\mathcal{F}(x)$  has exactly one minimum-weight element is more than  $\frac{3}{4}$ , and
- if  $x \notin L$ , then the fraction of the weight functions in  $\mathcal{Z}(|x|)$  with respect to which  $\mathcal{F}(x)$  has exactly one minimum-weight element is 0.

For every  $x \in \Sigma^*$ , every  $W \in \mathcal{Z}(|x|)$ , and every  $i$ ,  $1 \leq i \leq \mu(|x|)$ ,  $W(i) \leq 4\mu(|x|)$ . Thus, for every  $x \in \Sigma^*$ , every  $W \in \mathcal{Z}(|x|)$ , and every  $y \subseteq \mathcal{U}(|x|)$ ,  $W(y) \leq 4\mu^2(|x|)$ . Define

$$B = \{ \langle x, W, j \rangle \mid W \in \mathcal{Z}(|x|) \wedge 1 \leq j \leq 4\mu^2(|x|) \wedge \\ ||\{y \in \mathcal{F}(x) \mid W(y) = j \wedge \langle x, y \rangle \in A'\}|| = 1 \},$$

where  $W$  is encoded as  $W(1)\# \dots \# W(\mu(|x|))$ . Then  $B \in \text{US}$ , which can be witnessed by the nondeterministic Turing machine  $M$  that on input  $u$  behaves as follows: First,  $M$  checks whether  $u$  is of the form  $\langle x, w_1\#w_2\# \dots \#w_{\mu(|x|)}, j \rangle$  for some  $j$ ,  $1 \leq j \leq 4\mu^2(|x|)$ , and some  $w_1, \dots, w_{\mu(|x|)}$ ,  $1 \leq w_1, \dots, w_{\mu(|x|)} \leq 4\mu(|x|)$ . If the check fails,  $M$  immediately rejects  $u$ . Otherwise, using precisely  $\mu(|x|)$  nondeterministic moves,  $M$  selects  $y \in \Sigma^{\mu(|x|)}$ ; then  $M$  accepts  $u$  if and only if  $W(y) = j$  and  $\langle x, y \rangle \in A$ , where  $W$  is the weight function expressed by the string  $w_1\#w_2\# \dots \#w_{\mu(|x|)}$ , i.e., for all  $i$ ,  $1 \leq i \leq \mu(|x|)$ ,  $W(i) = w_i$ . Since  $B \in \text{US}$ , there is a polynomial-time many-one reduction  $g$  from  $B$  to  $\text{USAT}$ .

By the above probability analysis, for every  $x \in \Sigma^*$ ,

- if  $x \in L$ , the proportion of  $W \in \mathcal{Z}(|x|)$  such that for some  $j$ ,  $1 \leq j \leq 4\mu^2(|x|)$ ,  $\langle x, W, j \rangle \in B$  is at least  $\frac{3}{4}$ , and
- if  $x \notin L$ , the proportion of  $W \in \mathcal{Z}(|x|)$  such that for some  $j$ ,  $1 \leq j \leq 4\mu^2(|x|)$ ,  $\langle x, W, j \rangle \in B$  is 0.

Let  $N$  be a probabilistic Turing machine that, on input  $x \in \Sigma^*$ , behaves as follows:

**Step 1**  $N$  picks a weight function  $W$  as follows: For each  $i$ ,  $1 \leq i \leq \mu(|x|)$ ,  $N$  uniformly, randomly selects a binary string  $u_i$  having length  $2 + \log \mu(|x|)$ , then sets the value of  $W(i)$  to the binary integer  $1\hat{u}_i$ , where  $\hat{u}_i$  is the string  $u_i$  with its leading 0s omitted.

**Step 2**  $N$  picks  $j$ ,  $1 \leq j \leq 4\mu^2(|x|)$ , as follows:  $N$  selects a binary string  $v$  having length  $2 + 2\log \mu(|x|)$  uniformly at random. Then  $N$  sets  $j$  to

the integer whose binary encoding is  $1\hat{v}$ , where  $\hat{v}$  is the string  $v$  with its leading 0s omitted.

**Step 3**  $N$  asks its oracle whether  $g(\langle x, W, j \rangle) \in \text{USAT}$ . If the query is answered positively, then  $N$  accepts  $x$ . Otherwise, it rejects  $x$ .

Let  $x \in \Sigma^*$  be an input to  $N^{\text{USAT}}$ . Suppose  $x \in L$ . In Step 1,  $N^{\text{USAT}}$  on input  $x$  selects with probability at least  $\frac{3}{4}$  a weight function  $W$  such that for some  $j$ ,  $1 \leq j \leq 4\mu^2(|x|)$ ,  $\langle x, W, j \rangle \in B$ . Furthermore, in Step 2,  $N^{\text{USAT}}$  on input  $x$  selects each  $j$ ,  $1 \leq j \leq 4\mu^2(|x|)$ , with probability  $\frac{1}{4\mu^2(|x|)}$ . So, the probability that  $N^{\text{USAT}}$  on input  $x$  generates a query  $\langle x, W, j \rangle$  that belongs to  $B$  is at least  $\frac{3}{16\mu^2(|x|)}$ . Define  $q(n) = 22p^2(n)$ . Since for all  $n \geq 1$ ,  $\mu(n)$  is the smallest power of 2 that is greater than or equal to  $p(n)$ , for all  $n \geq 1$ ,  $2p(n) \geq \mu(n)$ . So,  $\frac{3}{16\mu^2(|x|)} \geq \frac{3}{64p^2(|x|)} \geq \frac{1}{22p^2(|x|)} = \frac{1}{q(|x|)}$ . Thus, the probability that  $N^{\text{USAT}}$  on input  $x$  accepts is at least  $\frac{1}{q(|x|)}$ . On the other hand, suppose  $x \notin L$ . Then,  $N^{\text{USAT}}$  on  $x$  rejects with probability 1. Thus,  $L \subseteq_{\text{randomized}} \text{USAT}$ .

□ Theorem 4.2

In the proof above, define

$$B' = \{ \langle x, W, j \rangle \mid x \in \Sigma^* \wedge 1 \leq j \leq 4\mu^2(|x|) \wedge W \in \mathcal{Z}(|x|) \wedge \\ ||\{y \in \mathcal{F}(x) \mid W(y) = j \wedge \langle x, y \rangle \in A'\}|| \text{ is an odd number} \}.$$

Then  $B' \in \oplus\text{P}$ . Also, in Step 3 of the program of machine  $N$ , replace the query string by  $\langle x, W, j \rangle$ . Call this new machine  $\hat{N}$ . For every  $x \in L$ , the same probability analysis holds because 1 is an odd number, so  $\hat{N}^{B'}$  on input  $x$  accepts with probability at least  $\frac{1}{q(|x|)}$ . For every  $x \in \bar{L}$ ,  $\hat{N}^{B'}$  on input  $x$  rejects with probability 1 because  $\mathcal{F}(x)$  is empty and 0 is an even number. This implies that  $L \subseteq_{\text{randomized}} B'$ . Furthermore, define  $T$  to be the probabilistic oracle Turing machine that on input  $x$ , sequentially execute independent simulation of  $\hat{N}$  on  $x$   $q(|x|)$  times, and then accepts if  $\hat{n}$  in at least one of the  $q(|x|)$  simulations and rejects otherwise. For every  $x \in L$ , the probability that  $T^{B'}$  on input  $x$  rejects is at most  $(1 - \frac{1}{q(|x|)})^{q(|x|)}$ . Since  $q(n) = 22p^2(n)$  and for all  $n \geq 0$ ,  $p(n) \geq 1$ , for all  $n \geq 0$ ,  $q(n) \geq 22$ . So, the probability that  $T^{B'}$  on input  $x$  rejects is at most  $(1 - \frac{1}{22})^{22} < \frac{1}{2}$ . On the other hand, for every  $x \in \bar{L}$ , the probability that  $T^{B'}$  on input  $x$  accepts is 0. Thus, we have proven the following theorem.

**Theorem 4.3**  $\text{NP} \subseteq \text{RP}^{\oplus\text{P}} \subseteq \text{BPP}^{\oplus\text{P}}$ .

## 4.2 Toda's Theorem: $\text{PH} \subseteq \text{P}^{\text{PP}}$

### 4.2.1 PH and $\text{BPP}^{\oplus\text{P}}$

By Theorem 4.3,  $\text{NP} \subseteq \text{BPP}^{\oplus\text{P}}$ . Since  $\text{P}^{\text{BPP}^A} = \text{BPP}^A$  for every oracle  $A$  (see Proposition 4.6 below), it holds that  $\Delta_2^{\text{P}} \subseteq \text{BPP}^{\oplus\text{P}}$ .

**Pause to Ponder 4.4** *Can we extend this inclusion in  $\text{BPP}^{\oplus \text{P}}$  to even higher levels of the polynomial hierarchy than  $\Delta_2^{\text{P}}$ ?*

In this section we show that indeed we can.

**Theorem 4.5** *For every  $k \geq 1$ ,  $\Sigma_k^{\text{P}} \subseteq \text{BPP}^{\oplus \text{P}}$ . Hence,  $\text{PH} \subseteq \text{BPP}^{\oplus \text{P}}$ .*

The proof of Theorem 4.5 is by induction on  $k$ . The base case is, of course, Theorem 4.3. For the induction step, we establish, for each  $k \geq 1$ , that  $\Sigma_{k+1}^{\text{P}} \subseteq \text{BPP}^{\oplus \text{P}}$  by combining Theorem 4.3 and our inductive hypothesis,  $\Sigma_k^{\text{P}} \subseteq \text{BPP}^{\oplus \text{P}}$ , in the following three steps:

1. (**Apply Theorem 4.3 to the base machine**) The proof of Theorem 4.3 is relativizable, so, for every oracle  $A$ ,  $\text{NP}^A \subseteq \text{BPP}^{\oplus \text{P}^A}$ . Noting that  $\Sigma_{k+1}^{\text{P}} = \text{NP}^{\Sigma_k^{\text{P}}}$ , we have the following, where the first inclusion is via the inductively true  $\Sigma_k^{\text{P}} \subseteq \text{BPP}^{\oplus \text{P}}$  and the second is via using relativized Theorem 4.3 as the oracle ranges over all  $\text{BPP}^{\oplus \text{P}}$  sets.

$$\Sigma_{k+1}^{\text{P}} \subseteq \text{NP}^{\text{BPP}^{\oplus \text{P}}} \subseteq \text{BPP}^{\oplus \text{P}^{\text{BPP}^{\oplus \text{P}}}}.$$

2. (**Swap BPP and  $\oplus \text{P}$  in the middle**) By Lemma 4.9 below,  $\oplus \text{P}^{\text{BPP}^A} \subseteq \text{BPP}^{\oplus \text{P}^A}$ , for every oracle  $A$ . So,

$$\Sigma_{k+1}^{\text{P}} \subseteq \text{BPP}^{\text{BPP}^{\oplus \text{P}^{\text{BPP}^{\oplus \text{P}}}}}.$$

3. (**Collapse  $\text{BPP}^{\text{BPP}}$  to BPP, and  $\oplus \text{P}^{\oplus \text{P}}$  to  $\oplus \text{P}$** ) By part 2 of Proposition 4.6 below,  $\text{BPP}^{\text{BPP}^A} = \text{BPP}^A$  for every oracle  $A$ . By part 2 of Proposition 4.8 below,  $\oplus \text{P}^{\oplus \text{P}} = \oplus \text{P}$ . So,

$$\Sigma_{k+1}^{\text{P}} \subseteq \text{BPP}^{\oplus \text{P}}.$$

We will first prove the two collapse results in Step 3, together with characterizations of BPP and  $\oplus \text{P}$ . The characterizations will be useful when we prove the “swapping” property in Step 2.

The results we prove in the rest of the section hold relative to any oracle. For simplicity, we prove only their nonrelativized versions.

**Proposition 4.6**

1. (**The error probability of BPP computation can be exponentially reduced without sacrificing much computation time**)  
*For every  $L \in \text{BPP}$  and every polynomial  $r$ , there exist a polynomial  $p$  and a language  $A \in \text{P}$  such that, for every  $x \in \Sigma^*$ ,*
  - a) *if  $x \in L$ , then the proportion of  $y \in \Sigma^{p(|x|)}$  such that  $x \# y$  belongs to  $A$  is at least  $1 - 2^{-r(|x|)}$ , and*
  - b) *if  $x \notin L$ , then the proportion of  $y \in \Sigma^{p(|x|)}$  such that  $x \# y$  belongs to  $A$  is at most  $2^{-r(|x|)}$ .*
2. *The BPP hierarchy collapses; i.e.,  $\text{BPP} = \text{P}^{\text{BPP}} = \text{BPP}^{\text{BPP}} = \text{P}^{\text{BPP}^{\text{BPP}}} = \text{BPP}^{\text{BPP}^{\text{BPP}}} = \dots$*

**Proof** We prove first part 1 of the proposition. Let  $L \in \text{BPP}$  via probabilistic Turing machine  $M$ . That is, for every  $x \in \Sigma^*$ ,  $M$  accepts  $x$  with probability at least  $\frac{3}{4}$  if  $x \in L$  and with probability at most  $\frac{1}{4}$  otherwise. Let  $r$  be any polynomial and let  $q(n) = 6r(n) + 1$ . Let  $N$  be a probabilistic Turing machine that, on input  $x$ , simulates  $M$  on input  $x$  exactly  $q(|x|)$  times, and accepts then if and only if  $M$  accepts in a majority of the simulations. Let  $n \geq 1$  and  $x \in \Sigma^n$ . Suppose that  $x \in L$ . Let  $\alpha$  be the probability that  $M$  on input  $x$  accepts and  $e = \alpha - \frac{1}{2}$ . Note that  $e \geq \frac{1}{4}$ . The probability that  $N$  on input  $x$  rejects is at most

$$\begin{aligned}
& \sum_{0 \leq i \leq \lfloor \frac{q(n)}{2} \rfloor} \binom{q(n)}{i} \left(\frac{1}{2} + e\right)^i \left(\frac{1}{2} - e\right)^{q(n)-i} \\
& \leq \sum_{0 \leq i \leq \lfloor \frac{q(n)}{2} \rfloor} \binom{q(n)}{i} \left(\frac{1}{2} + e\right)^{\frac{q(n)}{2}} \left(\frac{1}{2} - e\right)^{\frac{q(n)}{2}} \\
& = \sum_{0 \leq i \leq \lfloor \frac{q(n)}{2} \rfloor} \binom{q(n)}{i} \left(\frac{1}{4} - e^2\right)^{\frac{q(n)}{2}} \\
& \leq \sum_{0 \leq i \leq q(n)} \binom{q(n)}{i} \left(\frac{1}{4} - e^2\right)^{\frac{q(n)}{2}} \\
& = 2^{q(n)} \left(\frac{1}{4} - e^2\right)^{\frac{q(n)}{2}} \\
& = (1 - 4e^2)^{\frac{q(n)}{2}}.
\end{aligned}$$

This is at most  $\left(\frac{3}{4}\right)^{\frac{q(n)}{2}} \leq \left(\frac{3}{4}\right)^{3r(n)} < 2^{-r(n)}$ . Similarly, if  $x \notin L$ , then the probability that  $N$  on  $x$  accepts is at most  $2^{-r(|x|)}$ .

We view the randomized moves of  $N$  as being directed by tossing of coins. More precisely, at each randomized step of  $N$ , there are two possible choices and  $N$  selects one of the two by tossing a fair coin, the “head” for one move and the “tail” for the other. Then the coin tosses of  $N$  can be “normalized” in the sense that there is a polynomial  $p$  such that, for every  $x \in \Sigma^*$ ,  $N$  on  $x$  tosses exactly  $p(|x|)$  coins. Namely, the machine keeps track of the number of coin tosses it makes and, at the end of computation, if the number is less than  $p(|x|)$ , the machine makes dummy coin tosses to make the total number of coin tosses equal to  $p(|x|)$ . Pick such a  $p$  and let  $A$  be the set of all  $x\#y$  with  $y \in \Sigma^{p(|x|)}$  and such that  $N$  on  $x$  with coin tosses  $y$  accepts. Clearly,  $A \in \text{P}$  and, for every  $x \in \Sigma^*$ , the proportion of  $y \in \Sigma^{p(|x|)}$  such that  $x\#y \in A$  is equal to the probability that  $N$  on  $x$  accepts. So, conditions 1a and 1b both hold.

We now prove part 2 of the proposition. Let  $L \in \text{BPP}^A$  with  $A \in \text{BPP}$ . The language  $L$  is defined in terms of two probabilistic polynomial-time Tur-



ing machines, one for the base computation and the other for the oracle. Intuitively, we will show below that the two machines can be combined into a single probabilistic polynomial-time machine without creating much error.

Note that Part 1 in the above holds relative to any oracle. Let  $r(n)$  be an arbitrary strictly increasing polynomial. Then there is a polynomial-time probabilistic oracle Turing machine  $D$  such that, for every  $x \in \Sigma^*$ ,  $D^A$  on input  $x$  decides the membership of  $x$  in  $L$  correctly with probability at least  $1 - 2^{-r(|x|)}$ . We may assume that there exists a polynomial  $p$  such that, for every  $x \in \Sigma^*$  and every oracle  $Z$ ,  $D^Z$  on input  $x$  makes at most  $p(|x|)$  queries. Also, we may assume that each query of  $D$  on  $x$  is at least as long as the input. We replace the oracle  $A$  by  $\hat{A} = \{x\#y \mid y \in A\}$  and replace  $D$  by a new machine  $\hat{D}$  that, on input  $x \in \Sigma^*$ , simulates  $D$  on input  $x$  by substituting for each query  $y$  to  $A$  the a query  $x\#y$  to  $\hat{A}$ . By part 1, there is a polynomial-time probabilistic Turing machine  $N$  such that, for every  $u \in \Sigma^*$ ,  $N$  on  $u$  correctly decides the membership of  $u$  in  $\hat{A}$  with probability  $1 - 2^{-r(|u|)}$ . Let  $M$  be a probabilistic Turing machine that, on input  $x \in \Sigma^*$ , simulates  $\hat{D}$  on input  $x$  and when  $\hat{D}$  makes a query, say  $u$ , to the oracle, simulates  $N$  on input  $u$  to decide the oracle answer. For every  $x \in \Sigma^*$ ,  $\hat{D}$  on input  $x$  makes at most  $p(|x|)$  queries, and for every query  $u$  of  $\hat{D}$  on input  $x$ ,  $N$  on input  $u$  makes an error with probability at most  $2^{-r(|u|)} \leq 2^{-r(|x|)}$  since  $|u| \geq |x|$  and  $r$  is an increasing polynomial. So, for every  $x \in \Sigma^*$ , the probability of the paths on which the computation of  $M$  differs from that of  $\hat{D}^{\hat{A}}$  is at most  $p(|x|)2^{-r(|x|)}$ . Since  $D^A$  makes an error with probability at most  $2^{-r(|x|)}$ , the probability that  $M$  makes an error is at most  $(p(|x|) + 1)2^{-r(|x|)}$ . Since  $r$  is an increasing polynomial, for  $x$  sufficiently large, the error probability of  $M$  on  $x$  is smaller than  $\frac{1}{4}$ . Hence  $L \in \text{BPP}$ . Since  $\text{BPP}^{\text{BPP}} = \text{BPP}$ , clearly  $\text{BPP}^{\text{BPP}^{\text{BPP}}} = \text{BPP}^{\text{BPP}} = \text{BPP}$  via the application of this fact and, more generally, the BPP hierarchy collapses to BPP by induction.  $\square$

For a class  $\mathcal{C}$ ,  $\mathcal{C}/\text{poly}$  is the class of all languages  $L$  for which there exist an  $A \in \mathcal{C}$  and a polynomially length-bounded function  $h : \Sigma^* \rightarrow \Sigma^*$  such that, for every  $x \in \Sigma^*$ ,  $x \in L$  if and only if  $\langle x, h(0^{|x|}) \rangle \in A$  (see Sect. A.6). We have the following corollary.

**Corollary 4.7**  $\text{BPP} \subseteq \text{P}/\text{poly}$ .

**Proof** Let  $L \in \text{BPP}$ . Let  $r(n) = n + 1$ . By part 1 of Proposition 4.6, there exist a polynomial  $p$  and  $A \in \text{P}$  such that, for every  $x \in \Sigma^*$ , the proportion of  $y \in \Sigma^{p(|x|)}$  for which the equivalence,

$$x \in L \iff \langle x, y \rangle \in A,$$

does not hold is at most  $2^{-(|x|+1)}$ . Let  $n \geq 1$ . The proportion of  $y \in \Sigma^{p(n)}$  such that  $|\{x \in \Sigma^n \mid x \in L \iff \langle x, y \rangle \in A \text{ does not hold}\}| \geq 1$  is at most  $|\Sigma^n|2^{-(n+1)} = 2^n 2^{-(n+1)} < 1$ . So, there is some  $y \in \Sigma^{p(n)}$  such that, for every  $x \in \Sigma^n$ ,  $x \in L \iff \langle x, y \rangle \in A$ . Let  $h(0^n)$  be the smallest such  $y$ . Then,

for every  $x \in \Sigma^n$ ,  $x \in L$  if and only if  $\langle x, h(0^n) \rangle \in A$ . Since  $|h(0^n)| = p(n)$ , this implies that  $L \in P/\text{poly}$ .  $\square$

**Proposition 4.8**

1. For every  $L \in \oplus P$ , there exist a polynomial  $p$  and a language  $A \in P$  such that, for every  $x \in \Sigma^*$ ,  $x \in L$  if and only if  $||\{y \in \Sigma^{p(|x|)} \mid x\#y \in A\}||$  is odd.
2.  $\oplus P^{\oplus P} = P^{\oplus P} = \oplus P$ .

**Proof** To prove part 1, let  $L$  be in  $\oplus P$  via a nondeterministic polynomial-time Turing machine  $M$ . That is, for every  $x \in \Sigma^*$ ,

$$x \in L \text{ if and only if } \#acc_M(x) \text{ is odd.}$$

Let  $p$  be a polynomial bounding the runtime of  $M$ . Define  $A$  to be the set of all  $x\#y$ ,  $|y| = p(|x|)$ , such that  $y$  is an accepting computation path of  $M$  on  $x$  (that is, a sequence  $z$  of bits representing nondeterministic moves for  $M(x)$  leading to acceptance, on the path it specifies, on or after the move specified by the  $|z|$ th bit of  $z$  but before any further nondeterministic move is attempted) followed by an appropriate number of zeros. Clearly,  $A \in P$  and, for every  $x \in \Sigma^*$ , the number of  $y$ ,  $|y| = p(|x|)$ , such that  $x\#y \in A$  is  $\#acc_M(x)$ .

To prove part 2, let  $L$  be an arbitrary language in  $\oplus P^{\oplus P}$ . There exist a nondeterministic polynomial-time oracle Turing machine  $M$  and a language  $B \in \oplus P$  such that, for all  $x$ ,  $x \in L$  if and only if the number of accepting computation paths of  $M^B$  on input  $x$  is an odd number. We will construct a nondeterministic polynomial-time Turing machine  $M'$  witnessing that  $L \in \oplus P$ . Let  $N_1$  be a nondeterministic Turing machine witnessing that  $B \in \oplus P$ . As we will see in Proposition 9.3,  $\#P$  is closed under addition. So, the function  $1 + \#acc_{N_0}$  thus belongs to  $\#P$ . Let  $N_1$  be such that  $\#acc_{N_1} = 1 + \#acc_{N_0}$ . The function  $\#acc_{N_1}$  flips the parity of  $\#acc_{N_0}$ , in the sense that for all  $x$ ,  $\#acc_{N_1}(x)$  is an odd number if and only if  $\#acc_{N_0}(x)$  is an even number. Thus,  $N_1$  witnesses that  $\overline{B} \in \oplus P$ . Let  $M'$  be the nondeterministic Turing machine that, on input  $x$ , simulates  $M$  on  $x$  but each time  $M$  makes a query to the oracle, instead of making a query  $M'$  does the following two steps. (1)  $M'$  guesses a bit,  $b \in \{0, 1\}$ , about the oracle answer (where  $b = 0$  is interpreted as 'Yes' and  $b = 1$  as 'No') and a path of simulation of  $N_b$  (i.e.,  $N_0$  or  $N_1$ , depending on the choice of  $b$ ) on input  $w$ , where  $w$  is the query string of  $M$ . (2) Then  $M'$  returns to its simulation of  $M$  on input  $x$  with the guessed oracle answer. The machine  $M'$  accepts along a given path if and only if all the simulations of the machines  $N_0$ ,  $N_1$ , and  $M$  along that path accepted. We claim that  $M'$  witnesses that  $L \in \oplus P$ .

For each accepting computation path  $\pi$  of  $M'$  on  $x$ , let  $\tau(\pi)$  be the part of  $\pi$  corresponding to the computation of  $M$  on  $x$  and the guesses about the queries. That is,  $\tau(\pi)$  is  $\pi$  with all simulations of  $N_0$  and  $N_1$  removed. Only the guessed values of  $b$  remain encoded in  $\pi$ . Let  $t = \tau(\pi)$  for some

$\pi$ . How many accepting computation paths have  $t$  as their  $\tau$ -value? Let  $y_1, \dots, y_m$  be the query strings along  $\pi$  and  $b_1, \dots, b_m \in \{0, 1\}$  be the guessed values encoded in  $\pi$ . Then the number of such paths is the product of  $\#\text{acc}_{N_{b_1}}(y_1), \dots, \#\text{acc}_{N_{b_m}}(y_m)$ . This number is odd if and only if all the guesses about the queries are correct. Thus, the parity of the number of accepting computation paths of  $M'$  on  $x$  equals that of the number of accepting computation paths of  $M$  on  $x$  relative to  $B$ . Hence,  $L \in \oplus\text{P}$ .  $\square$

**Lemma 4.9**  $\oplus\text{P}^{\text{BPP}} \subseteq \text{BPP}^{\oplus\text{P}}$ .

**Proof** Let  $L \in \oplus\text{P}^{\text{BPP}}$ . We show that  $L \in \text{BPP}^{\oplus\text{P}}$ . By part 1 of Proposition 4.8, there exist a polynomial  $p$  and  $A \in \text{P}^{\text{BPP}}$  such that, for every  $x \in \Sigma^*$ ,

$$x \in L \iff ||\{y \in \Sigma^{p(|x|)} \mid x\#y \in A\}|| \text{ is odd.}$$

Furthermore, by part 2 of Proposition 4.6,  $\text{P}^{\text{BPP}} = \text{BPP}$ , so  $A \in \text{BPP}$ . Then, by part 1 of Proposition 4.6, for every polynomial  $r$ , there exist a polynomial  $p$  and  $B \in \text{P}$  such that, for every  $u \in \Sigma^*$ ,

$$\begin{aligned} &\text{the proportion of } v \in \Sigma^{p(|u|)} \text{ such that } u\#v \in B \text{ is at least} \\ &1 - 2^{-r(|u|)} \text{ if } u \in A \text{ and at most } 2^{-r(|u|)} \text{ otherwise.} \end{aligned} \quad (4.1)$$

Let  $r(n) = p(n) + 2$ . Let  $s$  be the polynomial such that, for every  $x \in \Sigma^*$  and  $y \in \Sigma^{p(|x|)}$ ,  $s(|x|) = q(|x\#y|)$ . Define (recall that  $\#$  is the specific function defined in Sect. 4.1.2)

$$\begin{aligned} C = \{ &x\#v \mid v \in \Sigma^{s(|x|)} \wedge \\ &||\{y \in \Sigma^{p(|x|)} \mid (x\#y)\#v \in B\}|| \text{ is an odd number} \}. \end{aligned}$$

Clearly,  $C \in \oplus\text{P}$ . For each  $x \in \Sigma^*$ , let

$$a(x) = ||\{y \in \Sigma^{p(|x|)} \mid x\#y \in A\}||$$

and, for each  $x \in \Sigma^*$  and  $v \in \Sigma^{s(|x|)}$ , let

$$c(x\#v) = ||\{y \in \Sigma^{p(|x|)} \mid (x\#y)\#v \in C\}||.$$

By equation 4.1, for every  $x \in \Sigma^*$ , the proportion of  $v \in \Sigma^{s(|x|)}$  satisfying the condition

$$(\forall y \in \Sigma^{p(|x|)}) [x\#y \in A \iff (x\#y)\#v \in B]$$

is at least  $1 - 2^{p(|x|)} 2^{-r(s(|x|))} \geq 1 - 2^{p(|x|) - p(s(|x|)) - 2} \geq 1 - 2^{-2} = \frac{3}{4}$ , and thus the proportion of  $v \in \Sigma^{s(|x|)}$  such that  $a(x) = c(x\#v)$  is at least  $\frac{3}{4}$ . Thus, for every  $x \in \Sigma^*$ , for at least  $\frac{3}{4}$  of  $v \in \Sigma^{s(|x|)}$ ,  $a(x)$  is odd if and only if  $c(x\#v)$  is odd. Note that  $a(x)$  is odd if and only if  $x \in L$  and that  $c(x\#v)$  is odd if and only if  $x\#v \in C$ . So, for every  $x \in \Sigma^*$ , for at least  $\frac{3}{4}$  of  $v \in \Sigma^{s(|x|)}$ ,  $x \in L$  if and only if  $x\#v \in C$ . Thus,  $L \in \text{BPP}^{\oplus\text{P}}$ .

Intuitively, the above argument can be explained as follows: We are looking at a table whose rows are  $y$ 's and whose columns are  $v$ 's, where the  $y$ 's

correspond to the nondeterministic guesses for the “parity” computation and the  $v$ ’s correspond to the random guesses used in the “BPP” computation (for testing whether a given  $x\#y$  belongs to  $A$ ). For each  $y$  and  $z$ , we place a letter “X” in the  $(y, v)$  entry of the table exactly if the randomized guesses  $v$  for the BPP computation on input  $x\#y$  lead to an error, i.e., either  $(x\#y \in A$  and  $(x\#y)\#v \notin B)$  or  $(x\#y \notin A$  and  $(x\#y)\#v \in B)$ . For each column  $v$  with no “X” the number of  $y$  such that  $(x\#y)\#v \in B$  is equal to the number of  $y$  such that  $x\#y \in A$ . So, for such column  $v$ ,  $x \in L$  if and only if the number of  $y$  such that  $(x\#y)\#v \in B$  is odd. In each row, the fraction of the entries having an “X” is at most  $2^{-r(s(|x|))}$ . There are only  $2^{p(|x|)}$  rows. Thus the fraction of the columns with an “X” is at most  $2^{-r(s(|x|)) + p(|x|)}$ . As equation 4.1 holds for any polynomial  $r$ , we can select  $r$  so that this amount is less than  $\frac{1}{4}$ . So, transpose the table: We’ll pick  $v$  first, then pick  $y$ . Then for more than  $\frac{3}{4}$  of  $v$  it holds that  $x \in L$  if and only if the number of  $y$  such that  $(x\#y)\#v \in B$  is odd. Hence we can switch the “BPP” part and the “parity” part.  $\square$

This concludes the proof of Theorem 4.5. We now show below some immediate corollaries to Theorem 4.5. Since  $\text{BPP} \subseteq \text{P/poly}$  by Corollary 4.7 and  $\text{P}^{\oplus \text{P}} = \oplus \text{P}$  by part 2 of Proposition 4.8,  $\text{BPP}^{\oplus \text{P}} \subseteq \oplus \text{P/poly}$ . Thus,  $\text{PH} \subseteq \oplus \text{P/poly}$ .

**Corollary 4.10**  $\text{PH} \subseteq \oplus \text{P/poly}$ .

By Lemma 4.9,  $\oplus \text{P}^{\text{BPP}} \subseteq \text{BPP}^{\oplus \text{P}}$ . By relativizing  $\oplus \text{P}$  by  $\text{PH}$  and then applying Theorem 4.5, we obtain the following result.

**Corollary 4.11**  $\oplus \text{P}^{\text{PH}} \subseteq \text{BPP}^{\oplus \text{P}} \subseteq \oplus \text{P/poly}$ .

#### 4.2.2 PP Is Hard for the Polynomial Hierarchy

We now prove Toda’s Theorem.

**Theorem 4.12 (Toda’s Theorem)**  $\text{PH} \subseteq \text{P}^{\# \text{P}[1]}$ .

**Corollary 4.13**  $\text{PH} \subseteq \text{P}^{\# \text{P}} = \text{P}^{\text{PP}}$ .

Theorem 4.12, Toda’s Theorem, follows immediately from Theorem 4.5 in light of the following lemma.

**Lemma 4.14**  $\text{PP}^{\oplus \text{P}} \subseteq \text{P}^{\# \text{P}[1]}$ . In particular,  $\text{BPP}^{\oplus \text{P}} \subseteq \text{P}^{\# \text{P}[1]}$ .

**Proof of Lemma 4.14** Let  $L \in \text{PP}^{\oplus \text{P}}$ . There exist a polynomial  $p$ , a function  $f \in \text{FP}$ , and a language  $A \in \text{P}^{\oplus \text{P}} = \oplus \text{P}$  (by Proposition 4.8) such that, for every  $x \in \Sigma^*$ ,

$$x \in L \iff ||\{y \in \Sigma^{p(|x|)} \mid x\#y \in A\}|| \geq f(x). \quad (4.2)$$

Let  $M$  be a nondeterministic polynomial-time Turing machine witnessing that  $A \in \oplus\text{P}$ . So, for every  $x \in \Sigma^*$ ,  $x \in A$  if and only if  $\#\text{acc}_M(x)$  is odd. Define  $s_0(z) = z$  and, for each  $i \geq 1$ , define polynomial  $s_i(z)$  with coefficients in  $\mathbb{N}$  by

$$s_i(z) = 3(s_{i-1}(z))^4 + 4(s_{i-1}(z))^3. \quad (4.3)$$

**Claim 4.15** *For every  $i \geq 0$  and every  $z \in \mathbb{N}$ , if  $z$  is even, then  $s_i(z)$  is a multiple of  $2^{2^i}$ , and if  $z$  is odd, then  $s_i(z) + 1$  is a multiple of  $2^{2^i}$ .*

**Proof of Claim 4.15** The proof is by induction on  $i$ . The claim trivially holds for the base case  $i = 0$ . For the induction step, let  $i = i_0$  for some  $i_0 \geq 1$  and suppose that the claim holds for values of  $i$  that are less than  $i_0$  and greater than or equal to 0. Suppose that  $z$  is even. By the inductive hypothesis,  $s_{i-1}(z)$  is divisible by  $2^{2^{i-1}}$ . Since  $s_i(z)$  is divisible by  $(s_{i-1}(z))^2$  and  $2^{2^i} = (2^{2^{i-1}})^2$ ,  $s_i(z)$  is divisible by  $2^{2^i}$ . Thus, the claim holds for even  $z$ . Suppose that  $z$  is odd. By the inductive hypothesis,  $s_{i-1}(z) = m2^{2^{i-1}} - 1$  for some  $m \in \mathbb{N}$ . So,

$$\begin{aligned} s_i(z) &= 3(m2^{2^{i-1}} - 1)^4 + 4(m2^{2^{i-1}} - 1)^3 \\ &= 3(m^4 2^{4(2^{i-1})} - 4m^3 2^{3(2^{i-1})} + 6m^2 2^{2(2^{i-1})} - 4m2^{2^{i-1}} + 1) \\ &\quad + 4(m^3 2^{3(2^{i-1})} - 3m^2 2^{2(2^{i-1})} + 3m2^{2^{i-1}} - 1) \\ &= 3m^4 2^{4(2^{i-1})} - 8m^3 2^{3(2^{i-1})} + 6m^2 2^{2(2^{i-1})} - 1 \\ &= 2^{2^i} (3m^4 2^{2^i} - 8m^3 2^{2^{i-1}} + 6m^2) - 1. \end{aligned}$$

Thus, the claim holds for odd  $z$  also.  $\square$  Claim 4.15

For each  $x \in \Sigma^*$ , let  $\ell_x = \lceil \log p(|x|) + 1 \rceil$  and define  $r_x(z) = (s_{\ell_x}(z))^{2^{\ell_x}}$  and  $g(x) = r_x(\#\text{acc}_M(x))$ . For every  $x \in \Sigma^*$ ,  $r_x(z)$  is a polynomial in  $z$  of degree  $2^{2^{\ell_x}}$ . The coefficients of the polynomial  $r_x$  are all nonnegative and polynomial-time computable. We claim that the function  $g$  is in  $\#P$ . This can be seen as follows. Let  $G$  be a nondeterministic Turing machine that, on input  $x$ , operates as follows:

- Step 1**  $G$  computes  $r_x(z) = a_0 z^0 + a_1 z^1 + \dots + a_m z^m$ , where  $m = 2^{2^{\ell_x}}$ .
- Step 2**  $G$  computes the list  $I = \{i \mid 0 \leq i \leq 2^{2^{\ell_x}} \wedge a_i \neq 0\}$ .
- Step 3**  $G$  nondeterministically selects  $i \in I$ .
- Step 4**  $G$  nondeterministically selects  $d$ ,  $1 \leq d \leq a_i$ .
- Step 5**  $G$  simulates  $M$  on input  $x$   $i$  times.
- Step 6**  $G$  accepts if and only if  $M$  accepts during each of the  $i$  simulations.

Then  $G$  satisfies  $g = \#\text{acc}_G$ . By Claim 4.15, the following conditions hold for every  $x \in \Sigma^*$ :

- ( $\star$ ) If  $x \in A$ , then  $\#\text{acc}_M(x)$  is odd, so  $g(x) - 1$  is a multiple of  $2^{2^{\ell_x}}$ , and thus,  $g(x)$  is of the form  $m2^{p(|x|)+1} + 1$  for some  $m$ .
- ( $\star\star$ ) If  $x \notin A$ , then  $\#\text{acc}_M(x)$  is even, so  $g(x)$  is a multiple of  $2^{2^{\ell_x}}$ , and thus,  $g(x)$  is of the form  $m2^{p(|x|)+1}$  for some  $m$ .

Define

$$h(x) = \sum_{|y|=p(|x|)} g(x\#y).$$

There is a nondeterministic Turing machine  $H$  such that  $h = \#acc_H$ , so  $h \in \#P$ . In particular,  $H$  guesses  $y \in \Sigma^{p(|x|)}$  and simulates  $G$  on  $x\#y$ . By equation 4.2,  $(\star)$ , and  $(\star\star)$ , the lowest  $p(|x|) + 1$  bits of the binary representation of  $h(x)$  represent the number of  $y \in \Sigma^{p(|x|)}$  such that  $x\#y \in A$ . So, for every  $x \in \Sigma^*$ ,  $x \in L \iff$  the leftmost  $p(|x|) + 1$  bits of  $h(x)$  is lexicographically at least  $010^{p(|x|)-1}$ . This implies that  $L$  is decidable by a polynomial-time Turing machine that makes one query to  $h$ . Since  $L$  was an arbitrary  $PP^{\#P}$  set and  $h = h_L \in \#P$ , it follows that  $PP^{\#P} \subseteq P^{\#P[1]}$ , the class of languages decided by a polynomial-time algorithm with one question to a  $\#P$  oracle.  $\square$  Lemma 4.14

So, Theorem 4.12 is established. Corollary 4.13 follows immediately from Theorem 4.12 in light of Proposition 4.16.

**Proposition 4.16**  $P^{PP} = P^{\#P}$ .

**Proof** First we show  $P^{PP} \subseteq P^{\#P}$ . Let  $L \in PP$ . There exist a polynomial  $p$ , a language  $A \in P$ , and  $f \in FP$  such that, for every  $x \in \Sigma^*$ ,

$$x \in L \iff ||\{y \mid |y| = p(|x|) \wedge \langle x, y \rangle \in A\}|| \geq f(x).$$

Let  $N$  be a nondeterministic Turing machine that, on input  $x$ , guesses  $y \in \Sigma^{p(|x|)}$ , and accepts  $x$  if and only if  $\langle x, y \rangle \in A$ . Clearly,  $N$  can be polynomial time-bounded. For every  $x \in \Sigma^*$ ,  $\#acc_N(x) = ||\{y \in \Sigma^{p(|x|)} \mid \langle x, y \rangle \in A\}||$ . Since  $f \in FP$  the membership in  $L$  can be tested in  $P^{\#P[1]}$ . Thus,  $P^{PP} \subseteq P^{\#P}$ .

Next we show  $P^{PP} \supseteq P^{\#P}$ . Let  $f$  be an arbitrary  $\#P$  function. Let  $f = \#acc_N$  for some polynomial-time nondeterministic Turing machine  $N$  and let  $p$  be a polynomial that strictly bounds the runtime of  $N$ . Then for all  $x$   $\#acc_N(x) < 2^{p(|x|)}$ . Define  $L = \{\langle x, y \rangle \mid 0 \leq y \leq 2^{p(|x|)} - 1 \wedge \#acc_N(x) \geq y\}$ . Define  $N'$  to be the nondeterministic Turing machine that, on input  $x \in \Sigma^*$ , operates as follows:  $N'$  simulates  $N$  on input  $x$  while counting in a variable  $C$  the number of nondeterministic moves that  $N$  makes along the simulated path. When  $N$  halts,  $N'$  guesses a binary string  $z$  of length  $p(|x|) - C$  using exactly length  $p(|x|) - C$  bits. Then  $N'$  accepts if and only if the simulated the path of  $N$  on  $x$  is accepting and  $z \in 0^*$ . Then for all  $x$   $\#acc_N(x) = \#acc_{N'}(x)$ . Also, for all  $x \in \Sigma^*$  and all computation paths  $\pi$  of  $N'$  on input  $x$ ,  $N'$  along path  $\pi$  makes exactly  $p(|x|)$  nondeterministic moves. Define  $D$  to the probabilistic Turing machine that, on input  $\langle x, y \rangle$ ,  $0 \leq y \leq 2^{p(|x|)} - 1$ , operates as follows:  $D$  uniformly, randomly selects  $b \in \{0, 1\}$ , and then does the following:

- If  $b = 0$ , then  $D$  uniformly, randomly selects  $z \in \{0, 1\}^{p(|x|)}$ , and then accepts if the rank of  $z$  is at most  $2^{p(|x|)} - y$  and rejects otherwise.

- If  $b = 1$ , then  $D$  simulates  $N'$  on input  $x$  by replacing each nondeterministic move of  $N'$  by a probabilistic move. More precisely, each time  $N'$  makes a nondeterministic move, deciding between two possible actions  $\alpha$  and  $\beta$ ,  $D$  selects uniformly, randomly  $c \in \{0, 1\}$ , and then  $D$  selects  $\alpha$  if  $c = 0$  and  $\beta$  otherwise.  $D$  accepts if  $N'$  on  $x$  along the simulated path accepts and rejects otherwise.

Clearly,  $D$  can be polynomial time-bounded. For every  $x \in \Sigma^*$ , the probability that  $D$  on input  $x$  accepts is

$$\frac{2^{p(|x|)} - y}{2^{p(|x|)+1}} + \frac{\#acc_{N'}(x)}{2^{p(|x|)+1}}.$$

This is greater than or equal to  $\frac{1}{2}$  if and only if  $\#acc_{N'}(x) \geq y$ . Thus,  $L \in \text{PP}$ . Hence,  $\text{P}^{\text{PP}} \supseteq \text{P}^{\#P}$ .  $\square$

We can strengthen Corollary 4.13 to show that not only the polynomial hierarchy but also  $\text{PP}^{\text{PH}}$  is included in  $\text{P}^{\text{PP}}$ .

**Corollary 4.17**  $\text{PP}^{\text{PH}} \subseteq \text{P}^{\text{PP}}$ .

**Proof** By Theorem 4.5,  $\text{PH} \subseteq \text{BPP}^{\oplus P}$  and by Lemma 4.14,  $\text{PP}^{\oplus P} \subseteq \text{P}^{\text{PP}}$ . So, it suffices to prove that, for every oracle  $X$ ,  $\text{PP}^{\text{BPP}^X} \subseteq \text{PP}^X$ . It follows that

$$\text{PP}^{\text{PH}} \subseteq \text{PP}^{\text{BPP}^{\oplus P}} \subseteq \text{PP}^{\oplus P} \subseteq \text{P}^{\text{PP}}.$$

Again, we prove only the nonrelativized version. Let  $L \in \text{PP}^{\text{BPP}}$ . There exist a polynomial  $p$ , a function  $f \in \text{FP}$ , and  $A \in \text{BPP}$  such that, for every  $x \in \Sigma^*$ ,

$$\text{if } x \in L, \text{ then } ||\{y \in \Sigma^{p(|x|)} \mid x\#y \in A\}|| \geq f(x), \text{ and} \quad (4.4)$$

$$\text{if } x \notin L, \text{ then } ||\{y \in \Sigma^{p(|x|)} \mid x\#y \in A\}|| \leq f(x) - 1. \quad (4.5)$$

Let  $r(n) = p(n) + 2$ . By part 1 of Proposition 4.6, there exist a polynomial  $q$  and a language  $B \in \text{P}$  such that, for every  $u \in \Sigma^*$ ,

$$\text{if } u \in A, \text{ then } ||\{v \in \Sigma^{q(|u|)} \mid u\#v \in B\}|| \geq 2^{q(|u|)}(1 - 2^{-r(|u|)}), \text{ and} \quad (4.6)$$

$$\text{if } u \notin A, \text{ then } ||\{v \in \Sigma^{q(|u|)} \mid u\#v \in B\}|| \leq 2^{q(|u|)}2^{-r(|u|)}. \quad (4.7)$$

Define  $s(n) = 2(n + 1 + p(n))$ . The length of  $x\#y$  with  $y \in \Sigma^{p(|x|)}$  is  $s(|x|)$ . Define  $D$  to be the set of all strings  $x\#yv$ , with  $y \in \Sigma^{p(|x|)}$  and  $v \in \Sigma^{q(s(|x|))}$ , such that  $x'\#v \in B$ , where  $x' = x\#y$ . Then  $D$  is in  $\text{P}$ . For each  $x \in \Sigma^*$ , let  $d(x) = ||\{yv \mid x\#yv \in D\}||$  and define  $g$  by  $g(x) = f(x)2^{q(s(|x|))}(1 - 2^{-r(s(|x|))})$ . Then  $g \in \text{FP}$ . For every  $x \in \Sigma^*$ , if  $x \in L$ , then by equations 4.4 and 4.6,  $d(x) \geq f(x)2^Q(1 - 2^{-R}) = g(x)$ , where  $P$ ,  $Q$ , and  $R$  respectively denote  $p(|x|)$ ,  $q(s(|x|))$ , and  $r(s(|x|))$ . On the other hand, if  $x \notin L$ , then by equations 4.5 and 4.7

$$\begin{aligned}
d(x) &\leq (f(x) - 1)2^Q + (2^P - f(x) + 1)2^Q 2^{-R} \\
&= f(x)2^Q - 2^Q + 2^{P+Q-R} - f(x)2^{Q-R} + 2^{Q-R} \\
&= f(x)2^Q(1 - 2^{-R}) - 2^Q(1 - 2^{P-R} - 2^{-R}) \\
&= g(x) - 2^Q(1 - 2^{P-R} - 2^{-R}).
\end{aligned}$$

Since  $r(n) = p(n) + 2$  and  $s(n) > n$ ,  $1 - 2^{P-R} - 2^{-R}$  is at least  $1 - \frac{1}{4} - \frac{1}{4} > 0$ . So,  $d(x) < g(x)$ . Hence, for every  $x \in \Sigma^*$ ,  $x \in L$  if and only if  $d(x) \geq g(x)$ . Thus,  $L \in \text{PP}^A$ .  $\square$

### 4.3 NL/poly = UL/poly

In the previous sections we saw a number of results that connect the polynomial hierarchy to polynomial-time counting complexity classes. Do the analogs of those results hold for logspace classes? In particular, do the logspace analogs of  $\text{PH} \subseteq \oplus\text{P/poly}$  (Corollary 4.10) and  $\text{PH} \subseteq \text{P}^{\text{PP}}$  (Corollary 4.13) hold? Since the NL hierarchy (under the Ruzzo–Simon–Tompia relativization, see Chap. 9) collapses to NL since  $\text{NL} = \text{coNL}$  (see Sect. A.7), we can simplify the question of whether the logspace analog of the former inclusion holds to the question of whether  $\text{NL} \subseteq \oplus\text{L/poly}$  and the question of whether the logspace analog of the latter inclusion holds to the question of whether  $\text{NL} \subseteq \text{L}^{\text{PL}}$ . Here the latter inclusion,  $\text{NL} \subseteq \text{L}^{\text{PL}}$ , trivially holds because  $\text{NL} \subseteq \text{PL}$ . Can we use the isolation technique to prove  $\text{NL} \subseteq \oplus\text{L/poly}$ ?

**Pause to Ponder 4.18** *Does  $\text{NL} \subseteq \oplus\text{L/poly}$  hold?*

The answer to this question is in the affirmative. In fact, we can prove something stronger:  $\text{NL/poly} = \text{UL/poly}$ . In other words, if all nondeterministic logspace machines are given access to advice functions having polynomial length, then NL and UL are equivalent.

#### 4.3.1 An NL-Complete Set

The Graph Accessibility Problem is the problem of deciding, for a given directed graph  $G$  and two nodes  $s$  and  $t$  of  $G$ , whether  $t$  is reachable from  $s$  in  $G$ . We consider a restricted version of the problem in which  $G$  has no self-loops, and  $s$  is the first node and  $t$  is the last node, where we order the nodes in  $G$  according to the adjacency matrix representation of  $G$ . More precisely, we consider the set,  $\widehat{\text{GAP}}$ , of all  $a_{11}a_{12} \cdots a_{1n}a_{21}a_{22} \cdots a_{2n} \cdots, a_{n1}a_{n2} \cdots a_{nn}$ ,  $n \geq 2$ , such that the diagonal elements  $a_{11}, \dots, a_{nn}$  are each 0, and  $n$  is reachable from 1 in  $G$ , where  $G$  is the directed graph whose adjacency matrix's  $(i, j)$ th element is  $a_{ij}$ . Since GAP, the Graph Accessibility Problem (without constraints on the numbering of the start and finish nodes, and without prohibiting self-loops), is well-known to be NL-complete, and since a logspace machine, given  $G$ ,  $s$ , and  $t$ , can swap the names of the source node



$s$  and 1, can swap the names of the sink node  $t$  and  $n$ , and can eliminate all self-loops,  $\text{GAP} \leq_m^L \widehat{\text{GAP}}$ .  $\widehat{\text{GAP}}$  is clearly in NL. Hence, our problem  $\widehat{\text{GAP}}$  is NL-complete too.

### 4.3.2 NL/poly = UL/poly

We show how to apply the Isolation Lemma (Lemma 4.1) to prove  $\text{NL} \subseteq \text{UL/poly}$ . Suppose we wish to decide the membership in  $\widehat{\text{GAP}}$  of an arbitrary  $n$ -node directed graph without self-loops. Let our universe  $\mathcal{U}(n)$  be the set of all potential edges in such a graph. Then  $|\mathcal{U}(n)| = n(n-1)$ . Let our weight functions map each edge in  $\mathcal{U}(n)$  to an integer between 1 and  $2n^3$ . For a given  $n$ -node directed graph  $G$  without self-loops, and for each  $i$ ,  $1 \leq i \leq n$ , define  $\mathcal{F}(n, G)_i$  to be the set of all simple paths in  $G$  from 1 to  $i$ . We view each element of  $\mathcal{F}(n, G)_i$  as a subset of  $\mathcal{U}(n)$ . Since  $\mathcal{F}(n, G)_i$  is a collection of simple paths from 1 to  $i$ , no two elements in  $\mathcal{F}(n, G)_i$  specify identical paths. Then a weight function  $W$  is good for  $\mathcal{F}(n, G)_i$  if and only if the minimum-weight path in  $G$  from 1 to  $i$  with respect to  $W$  is unique. Now apply Lemma 4.1 with  $m = n$ ,  $\mathcal{U} = \mathcal{U}(n)$ ,  $\mathcal{Z} = \mathcal{Z}(n)$ ,  $\mathcal{F}_1 = \mathcal{F}(n, G)_1, \dots, \mathcal{F}_n = \mathcal{F}(n, G)_n$ ,  $D = n(n-1)$ ,  $R = 2n^3 > 2mD$ , and  $\alpha = \frac{1}{2}$ . Let  $\mathcal{Z}(n)$  be the set of all weight functions whose values are at most  $2n^3$ . Then we have the following lemma.

**Lemma 4.19** *Let  $n \geq 2$  and let  $G$  be an  $n$ -node directed graph. Let  $\mathcal{U}(n)$ ,  $\mathcal{Z}(n)$ , and  $\mathcal{F}(n, G)_1, \dots, \mathcal{F}(n, G)_n$  be as stated above. Then more than half of the edge-weight functions in  $\mathcal{Z}$  are good for  $\mathcal{F}(n, G)_1, \dots, \mathcal{F}(n, G)_n$ .*

Suppose we wish to select, for each  $n \geq 2$ , a sequence of some  $m(n)$  weight functions,  $W_1, \dots, W_{m(n)} \in \mathcal{Z}(n)$ , such that for all  $n$ -node directed graphs  $G$ , there is some  $i$ ,  $1 \leq i \leq m(n)$ , such that  $W_i$  is good for  $\mathcal{F}(n, G)_1, \dots, \mathcal{F}(n, G)_n$ . How large  $m(n)$  should be? The following lemma states that  $m(n)$  can be as small as  $n^2$ .

**Lemma 4.20** *Let  $n \geq 2$ . Let  $\mathcal{U}(n)$ ,  $\mathcal{Z}(n)$ , and  $\mathcal{F}(n, G)_1, \dots, \mathcal{F}(n, G)_n$  be as stated above. There is a collection of edge-weight functions  $W_1, \dots, W_{n^2}$  in  $\mathcal{Z}(n)$  such that, for every  $n$ -node directed graph without self-loops,  $G$ , there is some  $k$ ,  $1 \leq k \leq n^2$ , such that  $W_k$  is good for  $\mathcal{F}(n, G)_1, \dots, \mathcal{F}(n, G)_n$ .*

**Proof of Lemma 4.20** Let  $n \geq 2$ . By Lemma 4.19, for every  $n$ -node directed graph without self-loops,  $G$ , the proportion of edge-weight functions in  $\mathcal{Z}(n)$  that are good for  $\mathcal{F}(n, G)_1, \dots, \mathcal{F}(n, G)_n$  is more than a half. So, for all  $n$ -node directed graphs without self-loops, the proportion of  $(W_1, \dots, W_{n^2})$  such that for all  $k$ ,  $1 \leq k \leq n^2$ ,  $W_k$  is bad for  $\mathcal{F}(n, G)_1, \dots, \mathcal{F}(n, G)_n$  is less than  $2^{-n^2}$ . There are  $2^{n(n-1)}$  directed  $n$ -node directed graphs without self-loops. So, the proportion of  $(W_1, \dots, W_{n^2})$  such that, for some  $n$ -node directed graph without self-loop  $G$ , for all  $i$ ,  $1 \leq i \leq n^2$ ,  $W_i$  is bad for  $\mathcal{F}(n, G)_1, \dots, \mathcal{F}(n, G)_n$  is less than  $2^{n(n-1)} 2^{-n^2} < 1$ . This implies that

there is some  $\langle W_1, \dots, W_{n^2} \rangle$  such that for all directed  $n$ -node directed graph without self-loop  $G$ , there is some  $i$ ,  $1 \leq i \leq n^2$ , such that  $W_i$  is good for  $\mathcal{F}(n, G)_1, \dots, \mathcal{F}(n, G)_n$ .  $\square$  Lemma 4.20

We define our advice function  $h$  as follows. For every  $n \geq 2$ ,  $h$  maps each string of length  $n$  to a fixed collection  $\langle W_1, \dots, W_{n^2} \rangle$  of  $n^2$  legitimate weight functions possessing the property in Lemma 4.20. For  $n = 1$ ,  $h$  maps each string of length  $n$  to the empty string. The domain size  $D$  is  $n(n-1)$  and the largest weight  $R$  is  $2n^3$ . So, by encoding each weight in binary, the encoding length of  $h(n)$  will be  $\mathcal{O}(n^4 \log n)$ .

Now we prove the following theorem.

**Theorem 4.21**  $\text{NL} \subseteq \text{UL/poly}$ , and thus,  $\text{NL/poly} = \text{UL/poly}$ .

In fact, we will prove the following result, from which Theorem 4.21 immediately follows.

**Theorem 4.22** *There is a UL machine that solves  $\widehat{\text{GAP}}$  using a polynomially length-bounded advice function.*

**Proof** For simplicity, in the following, let  $n \geq 2$  be fixed and let  $W_1 \dots W_{n^2}$  be the advice for length  $n$  (i.e., what the advice function gives, call it  $h$ ). Also, let  $G$  be an  $n$ -node graph  $G$  whose membership in  $\widehat{\text{GAP}}$  we are testing.

We need to define some notions and notation. For each  $i$ ,  $1 \leq i \leq n^2$ , and  $j$ ,  $1 \leq j \leq n$ , define  $\text{MinWeight}(i, j)$  to be the weight of the minimum-weight paths from 1 to  $j$  with respect to the weight function  $W_i$ ; if  $j$  is not reachable from 1 in  $G$ , then  $\text{MinWeight}(i, j) = \infty$ . For each  $i$ ,  $1 \leq i \leq n^2$ , and  $d \geq 0$ , define  $\text{Reach}(i, d)$  to be the set of all nodes  $j$ ,  $1 \leq j \leq n$ , that are reachable from 1 via paths of weight at most  $d$  with respect to the weight function  $W_i$ , define  $\text{Count}(i, d) = |\text{Reach}(i, d)|$ , and define  $\text{WeightSum}(i, d) = \sum_j \text{MinWeight}(i, j)$ , where  $j$  ranges over all elements in  $\text{Reach}(i, d)$ ; also, we say that  $W_i$  is  $d$ -nice if, for every  $j \in \text{Reach}(i, d)$ , there is a unique minimum-weight path from 1 to  $j$  in  $G$  with respect to  $W_i$ .

Due to our construction of  $h$ , every minimum-weight path has weight at most  $n(2n^3) = 2n^4$ . So, for every  $i$ ,  $1 \leq i \leq n^2$ , and for every  $d$ ,  $d \geq 2n^4$ , it holds that  $\text{Reach}(i, d) = \text{Reach}(i, d+1)$ ,  $\text{Count}(i, d) = \text{Count}(i, d+1)$ , and  $\text{WeightSum}(i, d) = \text{WeightSum}(i, d+1)$ . Note that 1 is the only node that can be reached from 1 without traversing edges. So, for all  $i$ ,  $1 \leq i \leq n^2$ , it holds that  $\text{MinWeight}(i, 1) = 0$ ,  $\text{Reach}(i, 0) = \{1\}$ ,  $\text{Count}(i, 0) = 1$ ,  $\text{WeightSum}(i, 0) = 0$ , and  $W_i$  is 0-nice.

We prove that if  $W_i$  is  $d$ -nice and if we know  $\text{Count}(i, d)$  and  $\text{WeightSum}(i, d)$ , then for any  $j$ ,  $1 \leq j \leq n$ , we can test, via unambiguous logspace computation, whether  $j$  belongs to  $\text{Reach}(i, d)$ . Recall that in the previous section we presented a nondeterministic logspace procedure for guessing a path,  $\pi_j$ , from 1 to a given node  $j$ . Let us modify this procedure as follows:

- For each node  $j$ ,  $1 \leq j \leq n$ , attempt to guess a path from 1 to  $j$  having weight at most  $d$  (with respect to  $W_i$ ) and having length at most  $n-1$ .

Count the number of  $j$  for which the guess is successful (call this number  $C$ ) and compute the sum of  $W_i(\pi_j)$  for all successful  $j$  (call this number  $S$ ).

- Output “successful” if  $C = \text{Count}(i, d)$  and  $S = \text{WeightSum}(i, d)$ . Output “failure” otherwise.

Note that if  $W_i$  is  $d$ -nice and both  $\text{Count}(i, d)$  and  $\text{WeightSum}(i, d)$  are correctly computed, then there is only one computation path in the above along which it holds that  $C = \text{Count}(i, d)$  and  $S = \text{WeightSum}(i, d)$ . Furthermore, the space requirement for this procedure is  $\mathcal{O}(\log n)$ , since the guessing part can be sequential,  $C \leq n$ , and  $S \leq n(2n^4) = 2n^5$ .

Now modify this procedure further, so that (i) it takes a number  $j$ ,  $1 \leq j \leq n$ , as an additional input, (ii) it memorizes whether the guess is successful for  $j$ , and if so, it memorizes the weight of the path it guesses, and (iii) if the computation is successful (namely, when  $C = \text{Count}(i, d)$  and  $S = \text{WeightSum}(i, d)$ ) it outputs the information that it has memorized in (ii). We call this modified version  $\text{ReachTest}$ . For a  $d$ -nice  $W_i$ , given  $\text{Count}(i, d)$  and  $\text{WeightSum}(i, d)$ ,  $\text{ReachTest}(j)$  behaves as an unambiguous logspace procedure. Since the modification does not change the space requirement, if  $W_i$  is  $2n^4$ -nice, then  $\text{ReachTest}(n)$  will discover, via unambiguous logspace computation, whether  $G \in \widehat{\text{GAP}}$ .

Now we have only to develop a UL procedure for finding an  $i$ ,  $1 \leq i \leq n^2$ , such that  $W_i$  is  $2n^4$ -nice, and for computing  $\text{Count}(i, 2n^4)$  and  $\text{WeightSum}(i, 2n^4)$  for that  $i$ . We design an inductive method for accomplishing this task. We vary  $i$  from 1 to  $n^2$  and, for each  $i$ , we vary  $d$  from 0 to  $2n^4$ . For each combination of  $i$  and  $d$ , we test whether  $W_i$  is  $d$ -nice, and if the test is passed, we compute  $\text{Count}(i, d)$  and  $\text{WeightSum}(i, d)$ . Note for every  $i$ ,  $1 \leq i \leq n^2$ , and for every  $d$ ,  $0 \leq d < 2n^4$ , that if  $W_i$  is not  $d$ -nice, then  $W_i$  is not  $(d+1)$ -nice. Thus, if we discover that  $W_i$  is not  $d$ -nice for some  $d$ , then we will skip to the next value of  $i$  without investigating larger values of  $d$ . Recall that for every  $i$ ,  $1 \leq i \leq n^2$ ,  $W_i$  is 0-nice,  $\text{Reach}(i, 0) = \{1\}$ ,  $\text{Count}(i, 0) = 1$ , and  $\text{WeightSum}(i, 0) = 0$ . Iterating the variables  $i$  and  $d$  requires only  $\mathcal{O}(\log n)$  space. So, it suffices to prove that there is a UL procedure that given  $i$ ,  $1 \leq i \leq n^2$ , and  $d$ ,  $0 \leq d \leq 2n^4 - 1$ , such that  $W_i$  is  $d$ -nice,  $\text{Count}(i, d)$ , and  $\text{WeightSum}(i, d)$ , tests whether  $W_i$  is  $(d+1)$ -nice, and if so computes  $\text{Count}(i, d+1)$  and  $\text{WeightSum}(i, d+1)$ . To obtain such an algorithm, the following fact is useful.

**Fact 4.23** *Let  $1 \leq i \leq n^2$  and  $0 \leq d \leq 2n^4 - 1$ . Suppose that  $W_i$  is  $d$ -nice. Then the following conditions hold:*

1. *For every  $u$ ,  $1 \leq u \leq n$ , the condition  $u \in \text{Reach}(i, d+1) - \text{Reach}(i, d)$  is equivalent to:  $u \notin \text{Reach}(i, d)$  and there is some  $v \in \text{Reach}(i, d)$  such that  $(v, u)$  is an edge of  $G$  and  $\text{MinWeight}(i, v) + W_i(v, u) = d+1$ .*
2.  *$W_i$  is  $(d+1)$ -nice if and only if for every  $u \in \text{Reach}(i, d+1) - \text{Reach}(i, d)$  there is a unique node  $v$  such that  $\text{MinWeight}(i, v) + W_i(v, u) = d+1$ .*

**Proof of Fact 4.23** Part 1 as well as the left to right direction of part 2 is straightforward. To prove the right to left direction of part 2, suppose  $W_i$  is not  $(d+1)$ -nice but is  $d$ -nice. Then there exists some  $u \in \text{Reach}(i, d+1) - \text{Reach}(i, d)$  such that there are two distinct paths from 1 to  $u$ , with respect to  $W_i$ . Since  $u \in \text{Reach}(i, d+1) - \text{Reach}(i, d)$ , the weight of the two paths should be  $d+1$ . So, they are minimum-weight paths.  $\square$  Fact 4.23

Now we build a UL algorithm for the incremental steps. Let  $1 \leq i \leq n^2$  and  $1 \leq d \leq 2n^4$ . Suppose that  $W_i$  is  $(d-1)$ -nice and that  $\text{Count}(i, d-1)$  and  $\text{WeightSum}(i, d-1)$  are known.

**Step 1** Set counters  $c$  and  $s$  to 0.

**Step 2** For each node  $u$ ,  $1 \leq u \leq n$ , do the following:

- (a) Call **ReachTest** to test whether  $u \in \text{Reach}(i, d-1)$ . Then
  - if the **ReachTest** outputs “failure,” then output “failure” and halt, else
  - if **ReachTest** asserts that  $u \in \text{Reach}(i, d-1)$ , then skip to the next  $u$ , else
  - if **ReachTest** asserts that  $u \notin \text{Reach}(i, d-1)$ , then proceed to (b).
- (b) Set the counter  $t$  to 0, then for each node  $v$  such that  $(v, u)$  is an edge in  $G$  call **ReachTest** to test whether  $v \in \text{Reach}(i, d-1)$  and
  - if **ReachTest** returns “failure,” then output “failure” and halt, else
  - if **ReachTest** asserts that  $v \in \text{Reach}(i, d-1)$  and  $\text{MinWeight}(i, v) + W_i(v, u) = d$ , then increment  $t$ .

Next,

- if  $t = 0$ , then move on to the next  $u$  without touching  $c$  or  $s$ , else
- if  $t = 1$ , then increment  $c$  and add  $d$  to  $s$ , else
- if  $t > 1$ , then assert that  $W_i$  is not  $d$ -nice and halt.

**Step 3** Set  $\text{Count}(i, d)$  to  $\text{Count}(i, d-1) + c$ , set  $\text{WeightSum}(i, d)$  to  $\text{WeightSum}(i, d-1) + s$ , and halt.

The correctness of the algorithm follows from Fact 4.23. It is clear that the space requirement is  $\mathcal{O}(\log n)$ . Note that  $W_i$  being  $d$ -nice guarantees that **ReachTest** $(i, d)$  produces exactly one successful computation path. So, there is a unique successful computation path of this algorithm, along which exactly one of the following two events occurs:

- We observe that  $W_i$  is  $(d+1)$ -nice and obtain  $\text{Count}(i, d+1)$  and  $\text{WeightSum}(i, d+1)$ .
- We observe that  $W_i$  is not  $(d+1)$ -nice.

Thus, we can execute the induction step by a UL algorithm. Putting all together, we have a UL machine that decides  $\widehat{\text{GAP}}$  with  $h$  (i.e.,  $W_1 W_2 \cdots W_{n^2}$  on inputs of length  $n$ ) as the advice. This completes the proof of Theorem 4.22.  $\square$  Theorem 4.22

## 4.4 OPEN ISSUE: Do Ambiguous and Unambiguous Nondeterminism Coincide?

In Sect. 4.3 we showed that the classes NL and UL are equal under polynomial-size advice. Can we get rid of the polynomial-size advice, i.e., is NL equal to UL? One tempting approach would be to derandomize the Isolation Lemma, however no one has yet succeeded along that path. It is now known that the equivalence holds if there is a set in DSPACE[ $n$ ] that requires circuits of size at least  $2^{cn}$  for some  $c > 0$ . In particular,  $NL = UL$  if SAT requires circuits of size at least  $2^{cn}$  for some  $c > 0$ .

Also, what can we say about the question of whether  $NP = UP$ ? There is an oracle relative to which  $NP \neq UP$ . Since there is also an oracle relative to which  $NP = UP$  (e.g., any PSPACE-complete set, as  $NP^{PSPACE} = UP^{PSPACE} = PSPACE$ ), relativizable proof techniques cannot settle the  $NP = UP$  question. In fact, it even remains an open question whether the assumption  $NP = UP$  implies a collapse of the polynomial hierarchy.

## 4.5 Bibliographic Notes

Part 2 of Proposition 4.6 is due to Ko [Ko82]. Lemma 4.9 is due to Schöning [Sch89]. Proposition 4.8 is due to Papadimitriou and Zachos [PZ83]. Regan [Reg85] first noted the closure of  $\#P$  under addition. The observation applies to  $\#L$  as well. Proposition 4.16 is due to Balcázar, Book, and Schöning ([BBS86], see also [Ang80,GJ79]). Buntrock et al. [BDHM92] proved that  $\oplus L^{\oplus L} = \oplus L$ . The oracle, mentioned in Sect. 4.4, relative to which  $NP \neq UP$  is due to Rackoff [Rac82].

The Isolation Lemma was established by Mulmuley, Vazirani, and Vazirani [MVV87]. Their version deals only with a single collection of sets. Our version, which deals with multiple collections of sets, is taken from the work of Gál and Wigderson [GW96].

Toda's Theorem is due to Toda [Tod91c], and Lemma 4.14, Theorem 4.5, Theorem 4.12, Corollary 4.10, and Corollary 4.17 are all from his seminal paper. Part 1 of Proposition 4.6 is often written as  $BPP = BP \cdot P$ , where  $BP \cdot$  is what is known as the BP quantifier or the BP operator. Heller and Zachos [ZH86] introduced this quantifier and first proved part 1 of Proposition 4.6. Corollary 4.7 generalizes Adleman's [Adl78] early result  $RP \subseteq P/poly$ , and can be found in Bennett and Gill [BG81] and Schöning [Sch86b]. One other form of Toda's Theorem is  $PP^{PH} \subseteq BP \cdot PP$ .

Toda and Ogiwara [TO92] showed that  $C=P^{PH} \subseteq BP \cdot C=P$  and, for every  $k \geq 2$ , that  $Mod_k P^{PH} \subseteq BP \cdot Mod_k P$ . Tarui [Tar93] showed that  $R \cdot$ , the one-sided error version of  $BP \cdot$ , can replace the  $BP \cdot$  on the right-hand side regarding  $PP$  and  $C=P$ , i.e.,  $PP^{PH} \subseteq R \cdot PP$  and  $C=P^{PH} \subseteq R \cdot C=P$ .

Gupta [Gup93] showed that  $R \cdot C=P = BP \cdot C=P$ . Green et al. [GKR<sup>+</sup>95] observed that PH is included in the class of decision problems whose memberships are determined by the “middle bit” of  $\#P$  functions.

The isolation technique used in the celebrated paper by Toda is the one by Valiant and Vazirani [VV86]. Roughly speaking, in order to reduce the cardinality of an unknown nonempty set  $S$  of nonzero vectors in  $\{0, 1\}^n$ , one applies a sequence of filters. Each filter can be written as  $b \cdot x = 0$ , where  $\cdot$  is the inner product modulo 2 of  $n$  dimensional vectors and only vectors satisfying  $b \cdot x = 0$  are passed through the filter. Valiant and Vazirani show that, for any nonempty  $S \subseteq \{0, 1\}^n - \{0^n\}$ , if a sequence of  $n$  random filters  $b_1, \dots, b_n \in \{0, 1\}^n$  is chosen, the probability that at some point  $i$ ,  $0 \leq i \leq n$ , there is exactly one vector in  $S$  that pass through all the filters up to  $b_i$  is at least  $\frac{1}{4}$ . Thus with this technique, one needs quadratically many bits to achieve a constant success probability in order to reduce the cardinality to one. We may ask whether it is possible to use fewer bits to achieve a constant success probability. Naik, Regan, and Sivakumar [NRS95] developed a reduction scheme that uses a quasilinear number of bits to achieve constant success probability.

Toda’s Theorem has applications in circuit theory. A translation of Theorem 4.5 into circuit classes is:  $AC^0$  (the class of languages recognized by families of polynomial-size, constant-depth, unbounded fan-in AND-OR circuits) is included in the class of languages recognized by families of size  $2^{\log^{O(1)} n}$ , depth-2 probabilistic circuits with a PARITY gate at the top and polylogarithmic fan-in AND gates at the bottom. This inclusion was first proven by Allender [All89a] using a different technique. Later, Allender and Hertrampf [AH94] showed that, for every prime number  $p$ , the class  $ACC_p$ —the class of languages recognized by families of polynomial size, constant-depth, circuits consisting of unbounded fan-in ANDs, unbounded fan-in ORs and unbounded fan-in MODULO  $p$  (computing whether the number of 1s in the inputs is not divisible by  $p$ ) gates—is included in the following three classes of languages: (a) the class of languages recognized by depth-4, polylogarithmic bottom-fan-in, size  $2^{\log^{O(1)} n}$  circuits consisting of unbounded fan-in ANDs, unbounded fan-in ORs, and unbounded fan-in MODULO  $p$  gates; (b) the class of languages recognized by depth-3, polylogarithmic bottom-fan-in, size  $2^{\log^{O(1)} n}$  circuits consisting solely of MAJORITY gates (computing whether the majority of the inputs are 1s); and (c) the class of languages recognized by depth-2, polylogarithmic bottom-fan-in, size  $2^{\log^{O(1)} n}$  probabilistic circuits with a MODULO  $p$  gate at the top and AND gates at the bottom. They also showed that if the  $ACC_p$  class is uniform, then the classes (a), (b), and (c) can be all uniform. Kannan et al. [KVVY93] independently proved the uniformity result regarding the inclusion involving class (c). Yao [Yao90] showed that the inclusion involving class (b) holds for nonprime modulo  $p$  as well. Tarui [Tar93] showed that  $AC^0$  is included in the class of languages recognized by depth-2, polylogarithmic bottom-fan-in, size  $2^{\log^{O(1)} n}$  proba-

bilistic circuits with a MAJORITY gate at the top and AND gates at the bottom. Beigel and Tarui [BT94] showed that in the inclusion involving class (c), deterministic circuits suffice. Beigel, Reingold, and Spielman [BRS91] showed that the class of languages accepted by a constant depth, polynomial-size circuits with any symmetric gate (i.e., the output depending only on the number of 1s in the inputs) and unbounded fan-in ANDs and ORs elsewhere is included in the class of languages recognized by a depth-2, size  $2^{\log^{O(1)} n}$  circuits with essentially the same symmetric gate at the top and polylogarithmic fan-in AND gates at the bottom.

Another application of the isolation technique is the probability one “inclusion” of NPMV in  $\text{FP}_{tt}^{\text{NP}}$ . Watanabe and Toda [WT93] proved that with probability one relative to a random oracle NPMV (the class of multivalued nondeterministic polynomial-time functions) has a refinement (see Chap. 3) in  $\text{FP}_{tt}^{\text{NP}}$ , the class of functions computable in polynomial time with parallel access to an NP oracle.

Wigderson [Wig94] showed how to apply the Isolation Lemma to prove  $\text{NL} \subseteq \oplus\text{L}/\text{poly}$ . In an expanded version of that paper, Gál and Wigderson [GW96] asked whether there was any use of the multiple collection version of the Isolation Lemma. Allender and Reinhardt [RA99] gave an affirmative answer to that question by proving Theorem 4.22. Chari, Rohatgi, and Srinivasan [CRS95] developed an isolation method that uses fewer random bits than that of Mulmuley, Vazirani, and Vazirani. The results that relate the  $\text{NL} = \text{UL}$  question to the circuit complexity of SAT and that of  $\text{DSPACE}(n)$ , mentioned in Sect. 4.4 are by Allender, Reinhardt, and Zhou [ARZ99]. For the fact that “standard” graph accessibility problem is NL-complete, see [Sav70].





## 5. The Witness Reduction Technique

### Pause to Ponder 5.1 *Is SAT in P?*

Don't you feel cheated when someone tells you the answer to a question before you've had a chance to ponder the issue for yourself? Sure you do, but this happens all the time. For example, if you are reading this book, you probably already know the beautiful theory of NP-completeness that was built by Cook, Levin, and Karp a few decades ago. So you already know the standard, striking, subtle answer the field has built to Pause to Ponder 5.1, namely, "We don't know whether or not SAT is in P, but we do know that SAT is in P if and only if all of the following thousand problems are in P, and we also know that SAT is in P if and only if at least one of the following (same) thousand problems is in P." (For reasons of space, we omit the list of one thousand NP-complete problems.) Basically, your ability to consider Pause to Ponder 5.1 as a fresh problem has been pretty thoroughly tainted.

We cannot give you back your intellectual virginity regarding NP-completeness. However, in this chapter, we will try to do the next best thing. In particular, in Sect. 5.1 we pose a seemingly simple question, Pause to Ponder 5.5, that you perhaps have not seen before. Of course, the question is not as important as " $P = NP$ "—after all, what question is?—but the question turns out to raise some quite subtle and interesting issues. In answering it, one might happen to build a theory of one's own that, at least in its general flavor, is not too dissimilar to the theory of NP-completeness. So, we urge you to take some time to pause, ponder, and—as an intellectual challenge and exercise—investigate Pause to Ponder 5.5, which is framed in the following section. Section 5.2 presents a theory that was built as an attempt to understand what the answer to Pause to Ponder 5.5 might be.

### 5.1 Framing the Question: Is $\#P$ Closed Under Proper Subtraction?

For the purpose of the rest of this chapter, we will use natural numbers and binary strings interchangeably, via the standard natural bijection that associates the natural number  $n$  with the lexicographically  $n + 1$ st string. That is, the number 0 corresponds to the empty string, the number 1 corresponds

to the string 0, the number 2 corresponds to the string 1, the number 3 corresponds to the string 00, and so on. In light of this bijection, we will usually, in this chapter, speak of integers rather than strings. Since actual Turing machines operate only on strings, when we say that  $f(n)$  is a polynomial-time computable operation, we mean the runtime is polynomial in the length of the string corresponding to  $n$ , and in fact the actual input to the Turing machine is the string corresponding to  $n$ —since in this chapter integers are just alternate interpretations of binary strings via the already-mentioned bijection. For example, there certainly is a Turing machine that takes two strings and, in polynomial-time, outputs the string corresponding to the sum of the integers corresponding to its two input strings, and so addition can be said to be a polynomial-time computable operation.

For the rest of this chapter, we use the term *operation* to describe any mapping from  $\mathbb{N} \times \mathbb{N}$  to  $\mathbb{N}$ .

**Definition 5.2** *Let  $\sigma$  be an operation and let  $\mathcal{F}$  be a class of functions from  $\mathbb{N}$  to  $\mathbb{N}$ . We say that  $\mathcal{F}$  is closed under (the operation)  $\sigma$  if*

$$(\forall f_1 \in \mathcal{F})(\forall f_2 \in \mathcal{F})[h_{f_1, f_2} \in \mathcal{F}],$$

where  $h_{f_1, f_2}(n) = \sigma(f_1(n), f_2(n))$ .

This definition merely captures one's natural intuition about what it means to be closed under an operation. In the definition we have used the operation  $\sigma$  as a 2-argument function, but when the operation is a “traditional” one, such as addition or proper subtraction, we will feel free to write expressions such as  $f_1(n) + f_2(n)$  rather than  $\text{addition}(f_1(n), f_2(n))$ .

Let us consider two examples.

**Example 5.3**  $\#P$  is closed under addition. This can be seen as follows. Let  $f_1$  and  $f_2$  be  $\#P$  functions. By the definition of  $\#P$ , this means there are nondeterministic machines  $N_1$  and  $N_2$  such that, on each input  $x$ ,  $f_1(x)$  equals the number of accepting paths of  $N_1(x)$  and  $f_2(x)$  equals the number of accepting paths of  $N_2(x)$ . To prove that  $\#P$  is closed under addition, consider the nondeterministic machine  $N$  that, on input  $x$ , makes one initial nondeterministic choice, namely, whether it will simulate  $N_1$  or  $N_2$ . Then the machine simulates the machine it chose. Note that, in effect, the computation tree of  $N(x)$  is a tree that has a root with two children, one child being the computation tree of  $N_1(x)$  and the other child being the computation tree of  $N_2(x)$ . So it is clear that the number of accepting paths of  $N(x)$  is exactly  $f_1(x) + f_2(x)$ .

**Example 5.4**  $\#P$  is also closed under multiplication. The proof is similar to that for addition. Let  $f_1$  and  $f_2$  again be  $\#P$  functions. As in the previous example, by the definition of  $\#P$  this means that there are nondeterministic machines  $N_1$  and  $N_2$  such that, on each input  $x$ ,  $f_1(x)$  equals the number of

accepting paths of  $N_1(x)$  and  $f_2(x)$  equals the number of accepting paths of  $N_2(x)$ . Consider a nondeterministic machine  $N$  that on input  $x$  nondeterministically guesses one computation path of  $N_1(x)$  and one computation path of  $N_2(x)$  and then accepts if both guessed paths are accepting paths. Clearly the number of accepting paths of  $N(x)$  is exactly  $f_1(x)f_2(x)$ , thus showing that #P is closed under multiplication.

It is not hard to see that #P has a variety of other closure properties involving binomial and multinomial coefficients. However, our interest is not in properties that are easy to show that #P *has*. Rather, we are interested in properties that it might have, and yet for which it currently seems hard (or impossible) to prove that #P has them. If this seems strange, consider as motivation the question of whether SAT is in P. It might be the case that  $\text{SAT} \in \text{P}$ . However, it probably is hard or impossible to prove that  $\text{SAT} \in \text{P}$ ; certainly, no one has yet proven  $\text{SAT} \in \text{P}$ . Nonetheless, the theory of NP-completeness does imply interesting things about whether  $\text{SAT} \in \text{P}$ , namely, that among all NP problems, SAT is a problem that is, logically speaking, least likely to be in P: If SAT is in P, then all NP problems are in P.

This brings us to our problem. Of course, an NPTM (nondeterministic polynomial-time Turing machine) cannot have a negative number of accepting paths. So if we are interested in closure under subtraction, we must restrict our attention to proper subtraction, i.e., the operation, from  $\mathbb{N} \times \mathbb{N}$  to  $\mathbb{N}$ , that is defined by  $a \ominus b = \max\{0, a - b\}$ .

**Pause to Ponder 5.5** *Is #P closed under proper subtraction?*

Section 5.2 provides a theory that explores this question, but we urge the reader to ponder the question first, as an interesting exercise. In case one gets stuck, the footnote to the present sentence gives some very strong hints as to how one can go about this.<sup>1</sup>

## 5.2 GEM: A Complexity Theory for Feasible Closure Properties of #P

This is a good point at which to explain the meaning of this chapter's title, the Witness Reduction Technique. To us, the term “witness reduction” actually encapsulates both an intuition and a technique. The intuition is that

<sup>1</sup> There are two promising approaches. One approach is that one can attempt to find a complexity class collapse that completely characterizes whether #P is closed under  $\ominus$ . (Hint: Such a characterization can be obtained). Another approach is to show that, in a sense very similar to that in which SAT is an NP problem that is “logically least likely” to be in P (i.e., if it is, then all problems are),  $\ominus$  is a polynomial-time computable operation under which #P is “logically least likely” to be closed (i.e., if #P is closed under  $\ominus$ , then #P is closed under *every* polynomial-time computable operation). The following section will follow both these approaches.

reducing the number of accepting paths (i.e., “witnesses”) of nondeterministic machines is difficult, and often has severe consequences. It is true that Example 5.3 makes it clear that *adding*, on each input, one accepting path to the number of accepting paths of a machine is possible (and indeed, is easy). The intuition of witness reduction says that *removing* one accepting path on every input on which the machine had a nonzero number of accepting paths will not be so easy, and may not be possible. Indeed, we’ll eventually see, in Sect. 5.3, that if one could do this—which after all is not general subtraction but is just decrementation—some surprising complexity class collapses would follow.

By witness reduction we are also referring to a very informal and loose technique, which will be used often in this chapter. The goal of the technique is to show that if a class, often  $\#P$ , has some closure property that reduces the number of witnesses, then some complexity class collapse occurs. The idea of the technique, for the case of  $\#P$ , is to jump back and forth between  $\#P$  functions and nondeterministic machines accepting languages. This is possible because every nondeterministic polynomial-time machine not only defines a  $\#P$  function but also defines an NP language. However, beyond that, if one can make one’s  $\#P$  function of a special form, then the function may define a language in a more restrictive class. For example, if a  $\#P$  function on each input either computes the integer 0 or the integer 1, then each machine constructing that function implicitly defines a UP language (and indeed the same UP language for each such machine). The general scheme of the witness reduction technique is that given the assumption that some particular closure property holds, we wish to prove that some collapse occurs, e.g.,  $UP = PP$ , as follows:

1. We take some set in the larger class, e.g.,  $PP$ , and take the machine for that set and (perhaps after some normalization/manipulation) coerce the machine into a  $\#P$  function.
2. Then we use the assumed closure to create a new  $\#P$  function.
3. Then we take that new  $\#P$  function and coerce it back into a machine, in particular into a machine defining a language in the smaller class, e.g.,  $UP$ .

Of course, the scheme just described is an abstracted and idealized template. Though in some cases one can see it used in exactly the form described above—e.g., the proof of Theorem 5.9—often the template has to be used in more creative ways. For example, in the main theorem of this section—our proof regarding proper subtraction—a large complexity class collapse is pieced together by linking two smaller collapses each of which is individually obtained via separate uses of the above template.

This concludes our comments about the flavor of the witness reduction technique. Now let us turn to discussing how we will approach a particular problem, namely, Pause to Ponder 5.5. As mentioned in footnote 1, we will

pursue in parallel two different approaches to studying whether #P is closed under proper subtraction.

One approach is to build a theory much like the theory of NP-completeness. We do not know whether SAT is in P, but one consequence of the fact that SAT is NP-complete is that SAT is not in P unless *all* NP sets are in P. Similarly, we will show that, though we do not know whether #P is closed under proper subtraction, it holds that #P is not closed under proper subtraction unless it is closed under *all* polynomial-time computable operations.

The other approach is to seek to completely characterize, in terms of complexity class collapses, the issue of whether #P is closed under proper subtraction. In particular, we will see that #P is closed under proper subtraction if and only if every probabilistic polynomial-time set is in fact in unambiguous polynomial time (i.e.,  $UP = PP$ ). This is a very dramatic collapse, and easily implies  $NP = coNP$ , as will be made explicit later in this section as Theorem 5.7.

The following theorem, which is proved via the witness reduction technique, simultaneously realizes the goals of both the approaches discussed above.

**Theorem 5.6** *The following statements are equivalent:*

1. #P is closed under proper subtraction.
2. #P is closed under every polynomial-time computable operation.
3.  $UP = PP$ .

**Proof** That part 2 implies part 1 is immediate. We will also show that part 1 implies part 3, and that part 3 implies part 2, thereby establishing the theorem.

Let us show that part 1 implies part 3. So assume that #P is closed under proper subtraction. It certainly suffices to show that this implies both  $PP \subseteq coNP$  and  $coNP \subseteq UP$ , as these together yield  $PP \subseteq UP$ . Since  $UP \subseteq PP$  holds without any assumption, we may then conclude  $UP = PP$ .

Let  $L$  be an arbitrary PP language. So, from the alternate definition of PP in Fig. A.19, there is a polynomial  $q$  and a polynomial-time predicate  $R$  such that

$$L = \{x \mid |\{y \mid |y| = q(|x|) \wedge R(x, y)\}| \geq 2^{q(|x|)-1}\}. \quad (5.1)$$

Letting, if needed,  $q_{new}(n) = q_{orig}(n) + 1$  and, for  $b \in \{0, 1\}$ ,  $R_{new}(x, yb) = R_{orig}(x, y)$ , it is clear that in equation 5.1, we may without loss of generality require that, for all  $n$ , it holds that  $q(n) \geq 1$ , and we do so. (The case  $q(|x|) = 0$  is what we are sidestepping here.)

There is an NPTM that on input  $x$  guesses each  $y$  such that  $|y| = q(|x|)$ , and then tests  $R(x, y)$ . Thus there is a #P function  $f$  such that  $x \notin L \implies f(x) < 2^{q(|x|)-1}$  and  $x \in L \implies f(x) \geq 2^{q(|x|)-1}$ . The function  $g(x) = 2^{q(|x|)-1} - 1$  is also a #P function, as  $q$  is a fixed polynomial and  $q(n) \geq 1$ .

Since  $\#P$  is by our assumption closed under proper subtraction, and as  $f$  and  $g$  are both  $\#P$  functions, we have that  $h \in \#P$ , where

$$h(x) = f(x) \ominus g(x).$$

Since  $h$  is a  $\#P$  function, there is an NPTM  $N$  such that, on each  $x$ ,  $h(x)$  equals the number of accepting paths of  $N(x)$ . However, note that  $f(x) \ominus g(x) = 0$  if  $x \notin L$ , and  $f(x) \ominus g(x) \geq 1$  if  $x \in L$ . Thus, viewed as an NP machine,  $L(N) = L$ . So our arbitrary PP language is in NP. Thus  $PP \subseteq NP$ . Since  $PP = \text{coPP}$ , this implies  $PP \subseteq \text{coNP}$ .

Still under the assumption that  $\#P$  is closed under proper subtraction, we now seek to prove that  $\text{coNP} \subseteq \text{UP}$ . Let  $L$  be an arbitrary coNP language. Let  $N$  be an NPTM accepting  $\bar{L}$ . Viewing  $N$  as a machine defining a  $\#P$  function  $f$ , note that  $x \in L \implies f(x) = 0$  and  $x \notin L \implies f(x) \geq 1$ . The constant function  $g(x) = 1$  is a  $\#P$  function. Since  $\#P$  is by assumption closed under proper subtraction,  $h(x) = g(x) \ominus f(x)$  must also be a  $\#P$  function. However,

$$x \in L \implies h(x) = 1$$

and

$$x \notin L \implies h(x) = 0.$$

So the NP machine corresponding to  $h$  proves that  $L \in \text{UP}$ . Since  $L$  was an arbitrary coNP language,  $\text{coNP} \subseteq \text{UP}$ .

Summing up, we have shown that if  $\#P$  is closed under proper subtraction then

$$PP \subseteq \text{coNP} \text{ and } \text{coNP} \subseteq \text{UP},$$

and thus, as discussed earlier in the proof, we have shown that part 1 implies part 3.

We now show that part 3 implies part 2. So, assume that  $\text{UP} = \text{PP}$ . Let  $op : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$  be any polynomial-time computable operation. Let  $f$  and  $g$  be arbitrary  $\#P$  functions. We will show that  $h(x) = op(f(x), g(x))$  is itself a  $\#P$  function.

It is not hard to see—and we suggest as an easy exercise that the reader verify these—that

$$B_f = \{\langle x, n \rangle \mid f(x) \geq n\} \in \text{PP}$$

and

$$B_g = \{\langle x, n \rangle \mid g(x) \geq n\} \in \text{PP}.$$

So the language

$$V = \{\langle x, n_1, n_2 \rangle \mid \langle x, n_1 \rangle \in B_f \wedge \langle x, n_1 + 1 \rangle \notin B_f \wedge \langle x, n_2 \rangle \in B_g \wedge \langle x, n_2 + 1 \rangle \notin B_g\}$$

must also be in PP, as  $V$  4-truth-table reduces to the PP language  $B_f \oplus B_g$  (with  $\oplus$  denoting disjoint union:  $Y \oplus Z = \{0x \mid x \in Y\} \cup \{1x \mid x \in Z\}$ ). However, PP is closed under bounded-truth-table reductions by Theorem 9.17.

$UP = PP$  by assumption, so  $V \in UP$ . (If one wants to see this without having to employ Theorem 9.17, one can do so by noting—in fact we will prove this and more as Theorem 5.7—that  $UP = PP$  implies that  $UP = P^{PP}$ , and so we are done as  $P^{PP}$  easily contains all sets that bounded-truth-table reduce to sets in  $PP$ .)

Since  $f$  and  $g$  are #P functions, there is a polynomial  $q$  such that, for all  $x$ ,  $\max\{f(x), g(x)\} \leq 2^{q(|x|)}$ . Consider the NP machine,  $N$ , that on input  $x$  does the following:

1. Nondeterministically choose an integer  $i$ ,  $0 \leq i \leq 2^{q(|x|)}$ .
2. Nondeterministically choose an integer  $j$ ,  $0 \leq j \leq 2^{q(|x|)}$ .
3. Nondeterministically guess a computation path of the UP machine for  $V$  on input  $\langle x, i, j \rangle$ . If the guessed path rejects then our simulation's current path rejects. If the guessed path accepts then nondeterministically guess an integer  $k$ ,  $1 \leq k \leq op(i, j)$ , and accept. (Note: If  $op(i, j) = 0$  then we will in this step generate no accepting paths, as no such  $k$  can be generated.)

So  $h(x) = op(f(x), g(x))$  must be a #P function, since for each  $x$ ,  $N(x)$  is an NPTM having exactly  $op(f(x), g(x))$  accepting paths. This is so as the  $V$  test in step 3 above succeeds only when  $i = f(x)$  and  $j = g(x)$ , and even that case succeeds on exactly one path of the machine for  $V$ , as that machine itself is a UP machine. Thus, since  $op$  was an arbitrary polynomial-time operation and  $f$  and  $g$  were arbitrary #P functions, we have shown that part 3 implies part 2.  $\square$

To help get an intuition as to how strong the collapse “ $UP = PP$ ” of Theorem 5.6 is, we state the following result that shows that it implies a huge number of collapses of more familiar complexity classes.

**Theorem 5.7** *The following statements are equivalent:*

1.  $UP = PP$ .
2.  $UP = NP = coNP = PH = \oplus P = PP = PP \cup PP^{PP} \cup PP^{PP^{PP}} \cup \dots$ .

**Proof**  $UP \subseteq NP \subseteq PP$ . So, since  $PP$  is closed under complementation,  $coNP \subseteq PP$ . Also, if  $NP = coNP$  then  $PH = NP$ . Thus, if  $UP = PP$  then  $UP = NP = PP = coNP = PH$ . Since  $P^{UP} \subseteq PH$ , under our assumption it holds that  $P^{UP} = UP$ . Consider  $PP^{\oplus P}$ . By Lemma 4.14, this is contained in  $P^{PP}$ , so by our assumption that  $UP = PP$  it is contained in  $P^{UP}$ , which as just noted under our assumption equals  $UP$ . So under our assumption  $PP^{\oplus P} = UP$ , and so  $PP^{\oplus P} = PP = \oplus P = UP$ . We are now done by easy induction, via repeated use of Lemma 4.14, e.g., taking the case of a “stack” of three  $PP$ 's,  $PP^{PP^{PP}} \subseteq PP^{PP^{\oplus P}} \subseteq PP^{P^{PP}} \subseteq PP^{PP} \subseteq PP^{\oplus P} = UP$ . (Note: by drawing on facts that are not hard but that we have not proven, such as that  $PP^{UP} = PP$ , one could give an alternate proof that does not invoke Lemma 4.14.)  $\square$

Theorem 5.6 shows that proper subtraction is, among all polynomial-time computable operations, the one most resistant to being a closure property of  $\#P$ . Is it the only operation to have this distinction? In fact, it is not. Just as there are many sets that are NP-complete, so also are there a variety of polynomial-time computable operations that, like proper subtraction, seem deeply resistant to being closure properties of  $\#P$ . If the notion of “deeply resistant to being closure properties” seems overly informal (since either they are or they are not closure properties—though we note that NP sets either are in P or are not in P, yet most people feel comfortable informally thinking of SAT as an NP set that is most unlikely to be in P), then in light of Theorem 5.6, we can simply seek polynomial-time operations  $\sigma$  for which one can prove that  $\sigma$  is a closure property of  $\#P$  if and only if all polynomial-time computable operations are closure properties of  $\#P$ . Here, we look at just one other such property, namely, integer division, i.e., the operation, from  $\mathbb{N} \times \mathbb{N}$  to  $\mathbb{N}$ , that is defined by  $a \oslash b = \lfloor a/b \rfloor$ .

Of course, an NPTM cannot have a fractional number of accepting paths, and thus studying division itself would be difficult. However, by studying integer division, we need not address that problem. Yet there is still a problem left, namely, division by zero. Throughout this chapter, closure will in general be defined via Definition 5.2. However, integer division is an exception. To avoid problems with division by zero, we formalize this exceptional case not by Definition 5.2, but rather by Definition 5.8, which explicitly contains a clause regarding that case.

**Definition 5.8** *Let  $\mathcal{F}$  be a class of functions from  $\mathbb{N}$  to  $\mathbb{N}$ . We say that  $\mathcal{F}$  is closed under integer division ( $\oslash$ ) if*

$$(\forall f_1 \in \mathcal{F})(\forall f_2 \in \mathcal{F} : (\forall n)[f_2(n) > 0])[f_1 \oslash f_2 \in \mathcal{F}],$$

where the 0 above is the integer zero (i.e., the integer represented by the empty string).

**Theorem 5.9** *The following statements are equivalent:*

1.  $\#P$  is closed under integer division.
2.  $\#P$  is closed under every polynomial-time computable operation.
3.  $UP = PP$ .

**Proof** Of course, part 2 trivially implies part 1. By Theorem 5.6, parts 2 and 3 are equivalent. Thus, we need only prove that part 1 implies part 3 and we are done.

Suppose that  $\#P$  is closed under integer division. Let  $L$  be any PP set. We seek to prove that  $L \in UP$ . It is not hard to see that if  $L \in PP$ , then there is an NPTM  $N$  and an integer  $k \geq 1$  such that

1. on each input  $x$ ,  $N(x)$  has exactly  $2^{|x|^k}$  computation paths, each containing exactly  $|x|^k$  binary choices,



2. on each input  $x$ ,  $x \in L$  if and only if  $N(x)$  has at least  $2^{|x|^k-1}$  accepting paths, and
3. on each input  $x$ ,  $N(x)$  has at least one rejecting path.

The number of accepting paths of  $N$  defines a  $\#P$  function, call it  $f(x)$ . Consider the function  $g(x) = 2^{|x|^k-1}$ , which clearly is a  $\#P$  function. So, by our hypothesis, the function  $h$  defined by  $h(x) = f(x) \odot g(x)$  must also be a  $\#P$  function. Note that if  $x \in L$  then  $h(x) = 1$ , and if  $x \notin L$  then  $h(x) = 0$ . Thus, the nondeterministic machine corresponding to  $h$  is itself a UP machine for  $L$ . So  $L$  is in UP.  $\square$

Finally, note that we have in fact proven more in this section than was explicitly claimed. In particular, note that in the proof of Theorem 5.9, the function that we divided by,  $g(x) = 2^{|x|^k-1}$ , though certainly a  $\#P$  function, is also a polynomial-time computable function. Thus, the proof actually establishes that  $\#P$  is closed under integer division (i.e., if  $f$  and  $g$  are in  $\#P$  and  $g$  is strictly positive then  $f(x) \odot g(x)$  is in  $\#P$ ) if and only if  $\#P$  is closed under integer division by natural-valued polynomial-time computable functions (i.e., if  $f$  is a  $\#P$  function and  $g$  is a polynomial-time computable function whose output is always a natural number greater than zero, then  $f(x) \odot g(x)$  is in  $\#P$ ), and both these conditions are themselves equivalent to  $UP = PP$ . Similarly, it also holds that  $\#P$  is closed under proper subtraction if and only if  $\#P$  is closed under proper subtraction by natural-valued polynomial-time computable functions, and both these conditions are themselves equivalent to  $UP = PP$ .

### 5.3 Intermediate Potential Closure Properties

In the previous section, we saw that some operations, such as addition, are closure properties of  $\#P$ . We also saw that if  $\#P$  is closed under either proper subtraction or integer division, then  $\#P$  is closed under *all* polynomial-time computable operations and  $UP = PP$ .

In this section, we study a different collection of operations—operations that  $\#P$  is not known to possess, but that also are not known to have the property that  $\#P$  is closed under them if and only if  $\#P$  is closed under all polynomial-time operations. Thus, for these operations, it remains possible that  $\#P$  is closed under them, yet is not closed under proper subtraction. Again returning to the analogy with NP-completeness theory, these operations are in some sense analogous to potentially “intermediate” sets—sets, such as the set of all primes, that are in NP yet are neither known to be NP-complete nor known to be in P. Among the closure properties that, as far as is currently known, fall into this strange intermediate territory, are taking minimums, taking maximums, proper decrement, and integer division by two. To be rigorous—both about what we mean by these operations and because the latter two of these operations stretch our notion of operation from 2-ary

operations to 1-ary operations—let us explicitly define these operations. The first two parts are just an application of Definition 5.2 to the particular two argument operators.

### Definition 5.10

1. We say that  $\#P$  is closed under minimum if, for each  $f, g \in \#P$ , the function  $h(x) = \min\{f(x), g(x)\}$  is in  $\#P$ .
2. We say that  $\#P$  is closed under maximum if, for each  $f, g \in \#P$ , the function  $h(x) = \max\{f(x), g(x)\}$  is in  $\#P$ .
3. We say that  $\#P$  is closed under proper decrement if, for each  $f \in \#P$ , the function  $h(x) = f(x) \ominus 1$  is in  $\#P$ .
4. We say that  $\#P$  is closed under integer division by two if, for each  $f \in \#P$ , the function  $h(x) = \lfloor f(x)/2 \rfloor$  is in  $\#P$ .

Regarding closure under proper decrement, the following partial results are the best known. The two parts of the theorem do not match perfectly, yet they are not too far apart: SPP is the “gap analog” of UP.

### Theorem 5.11

1. If  $\#P$  is closed under proper decrement, then  $\text{coNP} \subseteq \text{SPP}$  (equivalently,  $\text{NP} \subseteq \text{SPP}$ ).
2. If  $\text{UP} = \text{NP}$ , then  $\#P$  is closed under proper decrement.

**Proof** We will first prove part 1. Assume that  $\#P$  is closed under proper decrement. Let  $L$  be an arbitrary NP language. Let  $N$  be an NPTM for  $L$ , and let  $f$  be the  $\#P$  function defined by the cardinality of  $N$ 's accepting paths. Since  $\#P$  is closed under proper decrement,  $g(x) = f(x) \ominus 1$  is a  $\#P$  function. So there is an NPTM,  $N'$ , and a polynomial,  $p$ , such that on each input  $x$  it holds that  $N'(x)$  has exactly  $2^{p(|x|)}$  paths, and exactly  $g(x)$  of those are accepting paths. It follows, by reversing the accepting and rejecting path behavior of each path of  $N'$ , that the function  $2^{p(|x|)} - g(x)$  is a  $\#P$  function. Since  $\#P$  is closed under addition, it follows that the function  $f(x) + (2^{p(|x|)} - g(x))$  is a  $\#P$  function. However, as this function equals  $2^{p(|x|)}$  if  $f(x) = 0$  and equals  $2^{p(|x|)} + 1$  otherwise, the NPTM whose accepting path cardinalities define this  $\#P$  function is itself a machine showing that  $L \in \text{SPP}$ . Thus  $\text{NP} \subseteq \text{SPP}$ . Since  $\text{SPP} = \text{coSPP}$ ,  $\text{NP} \subseteq \text{SPP}$  and  $\text{coNP} \subseteq \text{SPP}$  are equivalent statements.

We turn to part 2 of the theorem. Assume that  $\text{UP} = \text{NP}$ . Let  $f$  be an arbitrary  $\#P$  function. Let  $N$  be an NPTM such that  $f(x)$  expresses the number of accepting paths of  $N$  on input  $x$ . Let  $B$  be the set of all strings  $\langle x, \phi \rangle$  such that  $\phi$  is an accepting path of  $N(x)$  and there exists an accepting path of  $N(x)$  that is lexicographically larger than  $\phi$ .  $B \in \text{NP}$ , so since  $\text{UP} = \text{NP}$  by assumption,  $B$  is in UP. Let  $N'$  be an NPTM accepting  $B$  such that on each input  $x$  it holds that  $N'(x)$  has at most one accepting computation path. We describe an NPTM,  $N''$ , such that on each input  $x$

the number of accepting paths of  $N''(x)$  is  $f(x) \ominus 1$ . On any input  $x$ ,  $N''$  nondeterministically guesses a computation path  $\phi$  of  $N(x)$  and then on its current path simulates  $N'(\langle x, \phi \rangle)$ . So, all  $N(x)$ 's accepting paths except the lexicographically largest one will contribute one of the accepting paths of  $N''(x)$ , and the lexicographically largest accepting path of  $N(x)$  will generate zero accepting paths. Thus, we indeed have proper-decremented  $f$ .  $\square$

Regarding closure of  $\#P$  under integer division by two, it is known to have seemingly unlikely consequences, but no “if and only if” characterization is known.

**Theorem 5.12** *If  $\#P$  is closed under integer division by two, then  $\oplus P = SPP$  (and thus  $PH \subseteq PP$ ).*

**Proof** This proof is similar in spirit to part 1 of Theorem 5.11. Assume that  $\#P$  is closed under integer division by two. Let  $L$  be an arbitrary  $\oplus P$  language. Let  $N$  be an NPTM and  $p$  be a polynomial such that (i) on each input  $x$  it holds that  $N(x)$  has exactly  $2^{p(|x|)}$  computation paths, and (ii) on each input  $x$  it holds that  $N(x)$  has an odd number of accepting paths if and only if  $x \in L$ . Let  $f$  be the  $\#P$  function defined by the cardinality of  $N$ 's accepting paths. Since  $\#P$  is by assumption closed under integer division by two, and is unconditionally closed under multiplication multiplication by fixed constants,

$$g(x) = 2(f(x) \oslash 2)$$

is a  $\#P$  function.

Also, the number of rejecting paths of  $N(x)$ , namely,  $f'(x) = 2^{p(|x|)} - f(x)$ , is clearly a  $\#P$  function. Since  $\#P$  is closed under addition, we have that

$$f'(x) + g(x) = (2^{p(|x|)} - f(x)) + 2(f(x) \oslash 2)$$

is a  $\#P$  function. However,  $f'(x) + g(x)$  equals  $2^{p(|x|)}$  if  $f(x)$  is even and equals  $2^{p(|x|)} - 1$  if  $f(x)$  is odd. So the NPTM whose accepting path cardinalities define this  $\#P$  function is itself a machine showing that  $L \in \text{coSPP}$ . Thus  $\oplus P \subseteq \text{coSPP}$ . Since  $SPP = \text{coSPP}$ , we conclude that  $\oplus P \subseteq SPP$ . Thus, since  $SPP \subseteq \oplus P$  holds unconditionally,  $\oplus P = SPP$ .

So if  $\#P$  is closed under integer division by two, then  $\oplus P = SPP$ . Note that Corollary 4.11 certainly implies that  $PH \subseteq PP^{\oplus P}$ . So if  $\#P$  is closed under integer division by two, then  $PH \subseteq PP^{SPP}$ . However,  $PP^{SPP} = PP$  (see Fig. A.20), so  $PH \subseteq PP$ .  $\square$

Regarding closure of  $\#P$  by minimum or maximum, no “if and only if” characterization is known, though some necessary conditions are known. Of course, since both minimum and maximum are polynomial-time computable operations, it goes without saying, via Theorem 5.6, that  $UP = PP$  is a sufficient condition.

**Theorem 5.13**

1. If  $\#P$  is closed under minimum then  $NP = UP$ .
2. If  $\#P$  is closed under maximum or under minimum then  $C=P = SPP$ .

**Proof** We first prove part 1. Let  $L$  be any NP language, fix an NPTM accepting  $L$ , and let  $f$  be the  $\#P$  function defined by the machine's number of accepting paths. Assume that  $\#P$  is closed under minimum. Then  $\min\{f(x), 1\}$  is a  $\#P$  function, but the NPTM with this number of accepting paths on each input  $x$  in fact is a machine proving that  $L \in UP$ .

We now prove part 2. Following the alternate definition in Sect. A.12, a language  $L$  is in  $C=P$  if there exists a polynomial  $q$  and a polynomial-time predicate  $R$  such that, for each  $x$ ,

$$x \in L \iff ||\{y \mid |y| = q(|x|) \wedge R(x, y)\}|| = 2^{q(|x|)-1}.$$

Note that for strings  $x$  that do not belong to  $L$ , all this says is that the number of strings  $y$  of length  $q(|x|)$  for which  $R(x, y)$  holds is *not*  $2^{q(|x|)-1}$ . However, those numbers could be either greater than or less than  $2^{q(|x|)-1}$ —indeed, perhaps greater than  $2^{q(|x|)-1}$  on some inputs and less than  $2^{q(|x|)-1}$  on other inputs. This makes it hard to exploit maximization or minimization to get useful conclusions. What we first need is a new characterization of  $C=P$  in which all rejection cardinalities fall “on the same side” of the acceptance cardinality. We state this in the following lemma. This lemma is somewhat related to some propositions in Chap. 9. In particular, Proposition 9.7 and (via a “multiplying a GapP function by negative one” tweak) Proposition 9.8 can alternatively be seen as following from the lemma.

**Lemma 5.14** *A language  $L$  is in  $C=P$  if and only if there exists a polynomial  $r$  and a polynomial-time predicate  $S$  such that, for each  $x$ ,*

1. *if  $x \in L$  then  $||\{y \mid |y| = r(|x|) \wedge S(x, y)\}|| = 2^{r(|x|)-2}$ , and*
2. *if  $x \notin L$  then  $||\{y \mid |y| = r(|x|) \wedge S(x, y)\}|| < 2^{r(|x|)-2}$ .*

**Proof of Lemma 5.14** Recall that a language  $L$  is in  $C=P$  exactly if there exists a polynomial  $q$  and a polynomial-time predicate  $R$  such that, for each  $x$ ,  $x \in L \iff ||\{y \mid |y| = q(|x|) \wedge R(x, y)\}|| = 2^{q(|x|)-1}$ . Let  $L$  be an arbitrary  $C=P$  language, and let  $q$  and  $R$  satisfy the definition just given. Let  $r(n) = 2q(n)$ . Let  $S(x, z)$  be the predicate that accepts exactly if

$$(\exists w_1, w_2 : |w_1| = |w_2| = q(|x|))[(w_1 \cdot w_2 = z) \wedge R(x, w_1) \wedge \neg R(x, w_2)],$$

where “ $\cdot$ ” denotes concatenation of strings. Let  $f(x)$  denote  $||\{y \mid |y| = q(|x|) \wedge R(x, y)\}||$ . The number of strings  $z$  of length  $r(|x|)$  for which  $S(x, z)$  accepts is exactly

$$h(x) = f(x)(2^{q(|x|)} - f(x)).$$

Note, crucially, that this function peaks when  $f(x) = 2^{q(|x|)-1}$ , at which point  $h(x)$  takes on the value  $2^{q(|x|)-1}2^{q(|x|)-1} = 2^{r(|x|)-2}$ . For all values of  $f(x)$ ,  $0 \leq f(x) \leq 2^{q(|x|)}$ , other than  $f(x) = 2^{q(|x|)-1}$ ,  $h(x) < 2^{q(|x|)-1}2^{q(|x|)-1} = 2^{r(|x|)-2}$ . Thus,  $r$  and  $S$  have exactly the property we were seeking.  $\square$  Lemma 5.14

We continue with the proof of part 2 of Theorem 5.13. We will prove separately the claim for maximum and for minimum.

Assume that  $\#P$  is closed under maximum. Let  $L$  be an arbitrary  $C=P$  language. Let  $r$  and  $S$  be as in Lemma 5.14. So clearly there is a  $\#P$  function  $f$  such that, for each  $x$ ,

1.  $x \in L \implies f(x) = 2^{r(|x|)-2}$ , and
2.  $x \notin L \implies f(x) < 2^{r(|x|)-2}$ .

The function  $g(x) = 2^{r(|x|)-2} - 1$  clearly belongs to  $\#P$ . By our assumption of closure under maximum,  $h(x) = \max\{f(x), g(x)\}$  is a  $\#P$  function. However, if  $x \in L$  then  $h(x) = 2^{r(|x|)-2}$ , and if  $x \notin L$  then  $h(x) = 2^{r(|x|)-2} - 1$ . Thus,  $L \in \text{SPP}$ . As  $L$  was an arbitrary  $C=P$  language,  $C=P \subseteq \text{SPP}$ , and thus  $C=P = \text{SPP}$  as  $\text{SPP} \subseteq C=P$  holds unconditionally.

We turn to the case in which we assume that  $\#P$  is closed under minimum. Let  $L$  be an arbitrary  $C=P$  language. Let  $r$  and  $S$  be as in Lemma 5.14. Let (polynomial-time) predicate  $S'(x, y)$  be defined such that it holds exactly when  $\neg S(x, y)$  holds. So, via the NPTM that guesses strings  $y$  of length  $r(|x|)$  and then checks  $S'(x, y)$ , clearly there is a  $\#P$  function  $f$  such that, for each  $x$ ,

1.  $x \in L \implies f(x) = (3/4)2^{r(|x|)}$ , and
2.  $x \notin L \implies f(x) > (3/4)2^{r(|x|)}$ .

The function  $g(x) = 1 + (3/4)2^{r(|x|)}$  clearly belongs to  $\#P$ . By our assumption of closure under minimum,  $h(x) = \min\{f(x), g(x)\}$  is a  $\#P$  function. However, if  $x \in L$  then  $h(x) = (3/4)2^{r(|x|)}$ , and if  $x \notin L$  then  $h(x) = 1 + (3/4)2^{r(|x|)}$ . Thus,  $L \in \text{coSPP}$ . However,  $\text{SPP} = \text{coSPP}$ , thus  $L \in \text{SPP}$ . So, again, we may conclude that  $C=P = \text{SPP}$ .  $\square$

## 5.4 A Complexity Theory for Feasible Closure Properties of OptP

$\#P$ , which captures the cardinality of the accepting path sets of NPTMs, is not the only computationally central class of functions. Another important class of functions is OptP, which captures the notion of maximizing over the outputs of an NPTM. This can be formalized as follows. Consider special NPTMs for which each path outputs some nonnegative integer—paths that do not explicitly do so are by convention viewed as implicitly having output the integer 0. A function  $f$  is an OptP function if there is some such machine,  $N$ , for which, on each  $x$ ,

$$f(x) = \max\{i \in \mathbb{N} \mid \text{some path of } N(x) \text{ has } i \text{ as its output}\}.$$

Just as we say that proper subtraction is, among all polynomial-time computable operations, in some sense the “least likely” closure property of  $\#P$ , can we also find a polynomial-time operation that is a “least likely” closure property of, for example,  $\text{OptP}$ ? And if so, is there also some complexity class collapse that characterizes whether  $\text{OptP}$  has all polynomial-time computable closure properties?

The answer is somewhat surprising. For  $\text{OptP}$  proper subtraction again is a “least likely” closure property. The same also holds for the well-studied function class  $\text{SpanP}$ . For both  $\text{OptP}$  and  $\text{SpanP}$ , there is a complexity class collapse that completely characterizes whether the class is closed under all polynomial-time computable operations. However, in all three cases— $\#P$ ,  $\text{OptP}$ , and  $\text{SpanP}$ —the characterizations differ, notwithstanding the fact that proper subtraction in each case is a least likely closure property.

**Theorem 5.15** *The following statements are equivalent:*

1.  $\text{OptP}$  is closed under proper subtraction.
2.  $\text{OptP}$  is closed under every polynomial-time computable operation.
3.  $\text{NP} = \text{coNP}$ .

**Proof** Part 2 immediately implies part 1.

We now argue that part 3 implies part 2. Assume  $\text{NP} = \text{coNP}$ . Let  $f$  and  $g$  be arbitrary  $\text{OptP}$  functions. Let  $N_f$  and  $N_g$  be NPTMs that prove that these are  $\text{OptP}$  functions. That is, on each input  $x$ , the maximum value output among all paths of  $N_f(x)$  will be  $f(x)$ , and on each input  $x$ , the maximum value output among all paths of  $N_g(x)$  will be  $g(x)$ . Let  $op : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$  be any polynomial-time computable operation. Define

$$L_f = \{\langle x, i \rangle \mid f(x) > i\}$$

and

$$L_g = \{\langle x, i \rangle \mid g(x) > i\}.$$

Clearly,  $L_f \in \text{NP}$  and  $L_g \in \text{NP}$ . Since  $\text{NP} = \text{coNP}$ ,  $\overline{L_f} \in \text{NP}$  and  $\overline{L_g} \in \text{NP}$ , say via, respectively, NPTMs  $N_1$  and  $N_2$ .

We now describe an NPTM that, viewed as a machine defining an  $\text{OptP}$  function, computes  $op(f(x), g(x))$ . On input  $x$  our machine will guess a computation path of  $N_f(x)$  and will guess a computation path of  $N_g(x)$ , and it will find the output of each of these computation paths. Let us call those outputs  $w_f$  and  $w_g$ , respectively. Our machine then guesses a path  $\rho_1$  of  $N_1(\langle x, w_f \rangle)$  and guesses a path  $\rho_2$  of  $N_2(\langle x, w_g \rangle)$ . The current path of our machine then outputs  $op(w_f, w_g)$  if

$$\begin{aligned} &(\rho_1 \text{ is an accepting path of } N_1(\langle x, w_f \rangle)) \wedge \\ &(\rho_2 \text{ is an accepting path of } N_2(\langle x, w_g \rangle)), \end{aligned}$$

and outputs 0 otherwise.

This machine will output 0 on each of its paths that does not guess paths having  $f(x)$  as  $w_f$  and having  $g(x)$  as  $w_g$ , or that does guess such paths but makes a bad guess for  $\rho_1$  or  $\rho_2$ . It will correctly output  $op(f(x), g(x))$  on each path that does guess paths achieving as their outputs  $f(x)$  and  $g(x)$  and that also guesses accepting paths of  $N_1$  and  $N_2$ . Note that some path indeed will make the correct guesses.

We now argue that part 1 implies part 3. Assume that OptP is closed under proper subtraction. Let  $L$  be an arbitrary NP language. Consider an NPTM that simulates a standard NPTM for  $L$  but that on each rejecting path outputs 0 and on each accepting path outputs 1. This machine proves that the characteristic function of  $L$  is an OptP function. The function  $g(x) = 1$  is also an OptP function.

Since OptP is by assumption closed under proper subtraction,  $h(x) = g(x) \ominus f(x)$  is an OptP function. Let  $N$  be an NPTM that computes  $h$  in the sense of an OptP machine, i.e., on each input  $x$ , the largest value output by  $N(x)$  is always  $h(x)$ . Note that  $h(x) = 0$  if  $x \in L$  and  $h(x) = 1$  if  $x \notin L$ . So consider the NPTM,  $N'$ , that on each input  $x$  guesses a path of  $N(x)$  and accepts (on the current path) if the guessed path outputs 1.  $L(N') = \bar{L}$ , so our arbitrary NP language in fact belongs to coNP. Thus  $\text{NP} = \text{coNP}$ .  $\square$

We leave as an exercise for the reader to prove for the function class SpanP (see Sect. A.16) the analog of Theorems 5.6 and 5.15.

**Theorem 5.16** *The following statements are equivalent:*

1. SpanP is closed under proper subtraction.
2. SpanP is closed under every polynomial-time computable operation.
3.  $\text{PP}^{\text{NP}} = \text{PH} = \text{NP}$ .

## 5.5 OPEN ISSUE: Characterizing Closure Under Proper Decrement

The open issue we would most like to bring to the reader's attention is a very natural one, yet it has long resisted solution. In Sect. 5.3, we saw a necessary condition— $\text{NP} \subseteq \text{SPP}$ —for  $\#P$  to be closed under proper decrement, and we also saw a sufficient condition— $\text{UP} = \text{NP}$ —for  $\#P$  to be closed under proper decrement. Can one find a complete characterization?

**Open Question 5.17** *Proper decrement is the (unary) operation  $\sigma(n) = n \ominus 1$ , i.e.,  $\sigma(n) = \max\{n - 1, 0\}$ . Find standard complexity classes  $C$  and  $D$  such that:*

*$\#P$  is closed under proper decrement if and only if  $C = D$ .*

## 5.6 Bibliographic Notes

The closure of  $\#P$  under addition and multiplication, Examples 5.3 and 5.4, was known to researchers, and appeared in notes, as far back as the early 1980s [Reg85,Reg01]. A variety of other closure properties of  $\#P$  were obtained (see the discussion in [HO93]) by Cai et al. [CGH<sup>+</sup>89] and Beigel and Gill [BG92].

Section 5.2 is due to Ogiwara and Hemachandra [OH93], except that the term “witness reduction” and the discussion of the general “philosophy” of witness reduction at the start of the section reflect the viewpoint of Gupta [Gup95]. The text before Theorem 5.9 mentions in passing that, in addition to proper subtraction and integer division, other operations are known to be closure properties of  $\#P$  if and only if  $UP = PP$ . Such operations include various operations having to do with the span and plurality tests on sets of functions, and can be found in Ogiwara and Hemachandra [OH93].

Section 5.3 is due to Ogiwara and Hemachandra [OH93], except that part 1 of Theorem 5.11 is due to Torán (see [OH93]). Also, part 2 of Theorem 5.11 here extends the following result of Ogiwara and Hemachandra [OH93]: If  $coNP \subseteq UP$ , then  $\#P$  is closed under proper decrement. To see the relationship between that result and part 2 of Theorem 5.11, note that  $coNP \subseteq UP \implies UP = coUP = NP = coNP$ . Thus  $coNP \subseteq UP \implies UP = NP$ . However, the converse is not known to hold.

Part 1 of Theorem 5.11 shows that if  $\#P$  is closed under proper decrement, then  $NP \subseteq SPP$ . We note here a different conclusion that also follows from the same hypothesis, and in doing so we will introduce a new, flexible class for dealing with modulo-based computation. For each  $k \geq 2$ , define the class  $FTM_kP$  (“finely tuned mod  $k$ ”) to be the collection of all  $L$  satisfying: For every polynomial-time computable function  $f : \Sigma^* \rightarrow \{0, 1, \dots, k-1\}$ , there is an NPTM  $N$  such that, for each  $x$ ,

1. if  $x \notin L$  then  $N(x)$  has no accepting paths, and
2. if  $x \in L$  then the number of accepting paths is congruent, mod  $k$ , to  $f(x)$ .

It is not hard to see that, for each  $k \geq 2$ ,  $FTM_kP \subseteq ModZ_kP$ , where the  $ModZ_kP$  are the “ModZ” classes defined by Beigel [Bei91b]. We can now state the additional claim that holds regarding proper decrement.

**Theorem 5.18** *If  $\#P$  is closed under proper decrement then, for each  $k \geq 2$ , it holds that  $NP \subseteq FTM_kP$ .*

The proof is simple. Given an NP language  $L$ , consider the machine that computes it. That defines a  $\#P$  function  $g$ . Since by assumption  $\#P$  is closed under proper decrement, each of the following  $k$  functions is a  $\#P$  function:  $kg(x) \ominus 0, kg(x) \ominus 1, \dots, kg(x) \ominus (k-1)$ . Now note that, for each polynomial-time function  $f : \Sigma^* \rightarrow \{0, 1, \dots, k-1\}$ , there thus will be a  $\#P$  function that on each input  $x$  evaluates  $f(x)$ , sees what it is congruent to modulo  $k$ ,



and then simulates the machine whose number of accepting paths define the appropriate one of the  $k$  functions mentioned above. So  $L$  indeed is in  $\text{FTM}_k\text{P}$ .

Section 5.4 is due to Ogiwara and Hemachandra [OH93]. For  $\text{SpanP}$  and  $\text{OptP}$ , Ogiwara and Hemachandra in fact prove that a variety of other operations—including proper division, spans, and, in the case of  $\text{SpanP}$ , pluralities—are also “least likely” polynomial-time computable closure properties.

Regarding Sect. 5.5, we mention that for the case of integer division by two (rather than proper subtraction by one), and even some more general division patterns, Gupta [Gup92] has obtained a complete characterization, in terms of complexity class collapses, regarding whether  $\text{GapP}$  has such a closure property. However, for  $\#P$  the issue remains open.

Though this chapter is concerned, except in parts of Sect. 5.3, with operations that operate on two arguments, one can also study operations on one argument. For this case, Cai et al. ([CGH<sup>+</sup>89], see the discussion in [HO93]) showed that  $\#P$  is closed under any finite sum of multiples of binomial coefficients whose upper element is the input and whose lower element is a constant, and Hertrampf, Vollmer, and Wagner ([HVW95], see also [Bei97]) showed that every one-argument operation other than those fails, in at least one relativized world, to be a closure property of relativized  $\#P$ . They also achieve a similar characterization for multi-argument operations. This approach—seeking which operations fail in at least one relativized world to be closure properties—differs from both the approaches (namely, characterizing closures in terms of complexity class collapses, and linking the relative likelihood of closure properties) pursued in Sect. 5.2. In some sense, it gives a somewhat less refined resolution than the approach of Sect. 5.2. For example, consider an operator under which  $\#P$  is closed if and only if  $\text{UP} = \text{PP}$  (equivalently,  $\text{UP} = \text{coUP} = \text{PP}$ ) and consider another operator under which  $\#P$  is closed if and only if  $\text{UP} = \text{coUP}$ . Having such characterizations gives perhaps greater insight into the relative likelihood that  $\#P$  has these closure properties than does merely knowing that for each of the two operations there is some relativized world in which  $\#P$  is not closed under the operation. On the other hand, obtaining “if and only if” characterizations linking the collapses of complexity classes to whether  $\#P$  has a given operation is relatively difficult, and no such complete characterizations have yet been obtained for many natural operations, e.g., those discussed in Sect. 5.3—though, even in those cases, the partial results that are known are sufficient to yield, via standard oracle results, the fact that there are relativized worlds in which the operations are not closure properties of  $\#P$ .

Gupta ([Gup95], see also [Gup92]) has suggested a very interesting alternate approach to closure properties. Given that it seems unlikely that  $\text{UP} = \text{PP}$ , and thus unlikely that  $\#P$  is closed under proper subtraction, he frames a different question: Is the proper subtraction of two  $\#P$  functions

such that it can be “approximated with high probability” by a function that is in a certain probabilistic version of  $\#P$ ? In his study, Gupta includes a fascinating essay linking whether functions fail to be closure properties to the fact that they reduce the number of witnesses (this should be contrasted with the later work of Hemaspaandra, Ogihara, and Wechsung on the reduction of numbers of solutions [HOW00] and of Durand, Hermann, and Kolaitis on the reduction of numbers of witnesses [DHK00]). In this book, the Isolation Technique chapter is also about witness reduction, and Gupta links the work underlying that chapter with the theory of closure properties. In addition, Gupta introduced, independently of Fenner, Fortnow, and Kurtz [FFK94], the class GapP. Fenner, Fortnow, and Kurtz studied some closure properties that GapP possesses, and Gupta ([Gup95,Gup92], see also [Bei97] regarding one-argument properties of GapP that fail in some relativized world) built, analogously to Sect. 5.2, a rich complexity theory for those properties that GapP seems not to possess. Beyond that, he also built a subtle and cohesive complexity theory for the class of functions that are quotients of GapP functions [Gup95,Gup92].

Yet another alternative approach to closure properties involves asking whether a class is “almost” closed under an operation, in the sense that some amount of extra pre- or post-processing brings the operation within the reach of the class. This approach has been investigated by Ogihara et al. [OTTW96], and there also, the consequences of the Isolation Technique (Chap. 4) play an important role.

Reduction not of the cardinality of accepting paths but rather of the cardinality of the acceptance type of NPMV functions has been studied by Hemaspaandra, Ogihara, and Wechsung [HOW00]. Their results contrast sharply with the theme of this chapter, as they show that in that setting cardinality reduction is possible in many cases. In fact, Hemaspaandra, Ogihara, and Wechsung [HOW00] give a sufficient condition for such cardinality reduction. They also show that for many cases not meeting the sufficient condition cardinality reduction is not possible unless the polynomial hierarchy collapses to  $\Sigma_2^P$ , and Kosub [Kos00] has shown that for each finite-cardinality-type case not meeting the sufficient condition there is at least one relativized world in which cardinality reduction for that case is not possible.

Finally, throughout this chapter, we have discussed and characterized whether classes ( $\#P$ , OptP, and SpanP) are closed under all *polynomial-time computable* operations. However, for each of these three function classes  $\mathcal{C}$ , it is reasonable to ask whether  $\mathcal{C}$  is closed under all  *$\mathcal{C}$ -computable* operations. In fact, Ogiwara and Hemachandra [OH93] have shown that, for each of these three classes, it holds that:  $\mathcal{C}$  is closed under all *polynomial-time computable* operations if and only if  $\mathcal{C}$  is closed under all  *$\mathcal{C}$ -computable* operations.

## 6. The Polynomial Interpolation Technique

A standard view of mathematical statements is that they should be accompanied by succinctly written, easily verifiable certificates. To wit, open one of your favorite mathematics or theoretical computer science textbooks (if you have one; if not, perhaps the present text will become your favorite). You'll see that all the formal statements there are accompanied by strings of text called proofs, which the author believes to be easily verifiable by anyone with enough background.

Pushing a little harder on that “easily verifiable” property of certificates, one arrives at the concept of algorithmic verification of mathematical statements. It is this concept that led Alan Turing to invent his “computation” model: the Turing machine.

Thus when we talk about standard Turing machine computation, mathematical statements are thought of as deterministically verifiable. For example, we often view NP as the class of languages with the property that every member has short, deterministically verifiable certificates but no nonmember has such certificates.

The interactive proof system, the focus of this chapter, reflects a new approach to verification. Two features are added to proof systems: the use of randomness and interactions with a machine (or a set of machines) that provides information. We no longer require that mathematical statements possess deterministically verifiable proofs. The mathematical correctness of statements is verified through interactions between two machines, called the verifier and the prover. While the computational power of the verifier is limited to polynomial time endowed with the ability to use randomness, the power of the prover is unlimited. The objective of the verifier is to determine with a high level of confidence whether the input is a valid statement, while the objective of the prover is to make the verifier believe that the statement is valid with as high confidence as possible, regardless of what probabilistic choices the verifier makes. A language has an interactive proof system if there is a protocol that has the following two properties: (1) for every member, a prover is able to make the verifier believe with very high probability (close to one) that it is a valid member, and (2) for every nonmember, the possibility that the verifier believes that the nonmember is a valid member is close to zero no matter what the prover does.

What kinds of languages can be verified by interactive proof systems? What are the relationships between this new system of proofs and the traditional system of succinct proof verification? Would there be any difference if more provers were added to the system? Precise answers to these questions have been given, through remarkable developments in the technique of constructing protocols between the verifier and the provers. The technique consists of two parts: (1) converting membership questions into arithmetic formulas and then (2) verifying evaluation of such formulas with gradual, random instantiation of the variables in the formula (polynomial interpolation).

In this chapter we swim through the progress in this area by outlining the development of the technique. In Sect. 6.1 we prove that  $P^{\#P}$  has interactive proof systems. In Sect. 6.3, improving upon the technique for  $P^{\#P}$ , we prove that the class of languages with interactive proof systems is precisely PSPACE. In Sect. 6.4 we prove that the class of languages with multiple-prover interactive proof systems is precisely NEXP. Section 6.2 presents an application of the polynomial interpolation technique to the problem of enumerating candidates for the permanent function.

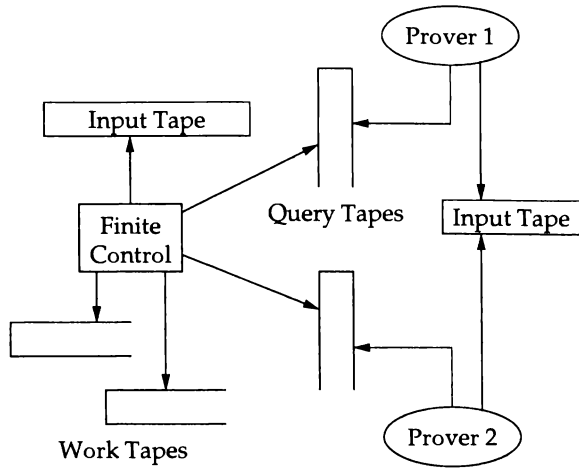
## 6.1 GEM: Interactive Protocols for the Permanent

### 6.1.1 Interactive Proof Systems

Let us formally define interactive proof systems (see Fig. 6.1). An interactive proof system has two components, a verifier and a set of provers. A *verifier* is a polynomial time-bounded probabilistic oracle Turing machine with  $k$  query tapes and  $k$  query states for some  $k \geq 1$ . For each  $i$ ,  $1 \leq i \leq k$ , the  $i$ th query tape and the  $i$ th query state are associated with a unique machine, called a *prover*. For each  $k \geq 1$ , when the verifier enters the  $k$ th query state, the contents of the  $k$ th query tape are read by the  $k$ th prover and its answer to the query replaces the query; all these actions take place in one step. The provers can use unlimited computational resources and randomness, and can remember previous interactions with the verifier. However, when there is more than one prover, they cannot communicate with each other.

**Definition 6.1** *For any  $k \geq 1$ , a language  $L$  has a  $k$ -prover interactive proof system if there exists a polynomial time verifier  $V$  interacting with  $k$  provers such that, for every  $x \in \Sigma^*$ , the following conditions hold:*

1. **(Completeness)** *If  $x \in L$ , then there is a set of  $k$  machines  $P_1, \dots, P_k$ , such that that  $V$  accepts  $x$  with probability greater than  $\frac{3}{4}$  with  $P_1, \dots, P_k$  as provers.*
2. **(Soundness)** *If  $x \notin L$ , then through interactions with any set of provers,  $V$  on input  $x$  rejects with probability greater than  $\frac{3}{4}$ .*



**Fig. 6.1** A two-prover interactive proof system

IP (respectively, MIP) is the class of all languages  $L$  that have one-prover interactive proof systems (respectively,  $k$ -prover interactive proof systems for some  $k$ ).

We'll drop the "one-prover" whenever it is clear from the context that there is only one prover.

### 6.1.2 Low-Degree Polynomials and Arithmetization

One of the two basic ingredients of the polynomial interpolation technique is *arithmetization*—transforming computational problems to those of evaluating (algebraic) formulas involving polynomials. Two properties of polynomials are crucial:

- *Low-degree, nonzero polynomials have a small number of zeros.*
  - (Univariate Polynomials) If  $f$  is a polynomial of degree  $d$  over a field  $F$ , then the number of roots of  $f$  is at most  $d$ . (See the proof of Lemma 6.2.)
  - (Multivariate Polynomials) If  $f$  is an  $s$ -variate polynomial of total degree at most  $d$  over a field  $F$ , then the number of elements of  $F^s$  that are roots of  $f$  is at most  $d||F||^{d-1}$ . (See Lemma 6.32).
- *Low-degree polynomials have a robust characterization*, in the following sense:
  - (Univariate Polynomials) If  $f$  is a polynomial of degree  $d$  over a field  $F$ , then  $f$  can be specified uniquely either by the  $d + 1$  coefficients of  $f$  or by a list of  $d + 1$  points that  $f$  passes through. (See Lemma 6.28.)

- (Multivariate Polynomials) If  $f$  is an  $s$ -variate polynomial of total degree at most  $d$  over a field  $F$ , then for every  $y, z \in F^s$ , it holds that  $\sum_{0 \leq i \leq d+1} \gamma_i f(y + iz) = 0$ , where for every  $i$ ,  $0 \leq i \leq d+1$ ,  $\gamma_i = \binom{d+1}{i} (-1)^i$ . (See Lemma 6.31.)

To develop interactive proof systems for  $P^{\#P}$  ( $= P^{PP}$ ), PSPACE, and NEXP, we use some specific properties of these traditional complexity classes.

- $P^{\#P}$ : We show that the permanent function, which is complete for  $\#P$ , has an interactive proof system, i.e., the permanent can be verified interactively. The basic property we use is that the permanent of a matrix can be uniquely recovered from the permanent of its minors (see part 3 of Proposition 6.3).
- PSPACE: We arithmetize the reachability problem on the computation tree of a deterministic polynomial-space Turing machine. There are no branches in the computation tree of a deterministic Turing machine, so for every  $k \geq 1$ , if there is a length  $2^k$  path from a configuration  $u$  to a configuration  $v$ , then there is a unique “middle point” configuration  $w$  such that there is a length  $2^{k-1}$  path from  $u$  to  $w$  as well as a length  $2^{k-1}$  path from  $w$  to  $v$ . We will transform this observation into a sequence of verification.
- For NEXP, we arithmetize the computation of an exponential-time nondeterministic Turing machine by applying the tableau method. We obtain a characterization of each NEXP language  $L$ : For every  $x$ , there is a 3CNF formula  $\varphi_x$  having exponentially many clauses over exponentially many variables, and  $x \in L$  if and only if the formula  $\varphi_x$  is satisfiable. We develop a protocol for verifying that  $\varphi_x$  is satisfiable.

### 6.1.3 A Robust Characterization of Low-Degree Univariate Polynomials

As stated in the following lemma, low-degree polynomials that are different from each other cannot agree at many points. For a ring  $R$  and a set of variables  $X_1, \dots, X_m$ ,  $R[X_1, \dots, X_m]$  denotes the set of all polynomials in  $X_1, \dots, X_m$  with coefficients in  $R$ .

**Lemma 6.2 (The number of roots of a polynomial)** *Let  $R$  be a ring without zero divisors. Let  $d \geq 1$  be an integer such that if the multiplicative group of  $R$  is finite, then its order is greater than  $d$ . Let  $f$  and  $g$  be polynomials in  $R[X]$  of degree at most  $d$  that are different from each other. Then  $f(r) = g(r)$  for at most  $d$  values of  $r$ .*

Multivariate polynomials have a similar property (see in Lemma 6.32 in Sect. 6.4).

**Proof** Let  $h(X) = f(X) - g(X)$ . Then  $h$  is a nonzero polynomial of degree  $e$ ,  $0 \leq e \leq d$ . Let  $a$  be the coefficient of  $X^e$  in  $h(X)$ . Since there is no zero-divisor in  $R$ ,  $h'(X) = h(X)/a$  is defined. Then, for every  $r \in R$ ,  $f(r) =$

$g(r) \iff h'(r) = 0$ . Since the coefficient of  $X^e$  in  $h'(X)$  is 1, there are at most  $d$  roots of  $h'$ . So,  $f(r) = g(r)$  for at most  $d$  values of  $r$ .  $\square$

#### 6.1.4 The Permanent Function

Let  $n \geq 1$  be an integer. By  $\mathcal{M}_n(\mathbb{Z})$  we denote the set of all  $n \times n$  matrices over  $\mathbb{Z}$ . For  $A = (a_{ij}) \in \mathcal{M}_n(\mathbb{Z})$ , the *permanent* of  $A$ , denoted by  $\text{perm}(A)$ , is  $\sum_{\pi} \prod_{1 \leq i \leq n} a_{i\pi(i)}$ , where  $\pi$  ranges over all permutations of  $\{1, \dots, n\}$ . The *dimension* of  $A$ , denoted by  $\dim(A)$ , is  $n$ . If  $n \geq 2$ , for each  $i, j$ ,  $1 \leq i, j \leq n$ , the  $(i, j)$ th *minor* of  $A$ , denoted by  $A_{i|j}$ , is the matrix constructed from  $A$  by striking out the  $i$ th row and the  $j$ th column simultaneously.

#### Proposition 6.3

1. For each  $f \in \#P$ , there exist two polynomial-time computable functions  $R_1$  and  $R_2$  such that, for every  $x \in \Sigma^*$ , the following two conditions hold:
  - $R_1(x)$  is a square matrix all of whose entries are nonnegative integers.
  - $f(x) = R_2(\langle x, \text{perm}(R_1(x)) \rangle)$ .
2. The problem of computing the permanent of matrices whose entries are nonnegative integers belongs to  $\#P$ .
3. Let  $A \in \mathcal{M}_n(\mathbb{Z})$ , for some  $n \geq 2$ . Then

$$\text{perm}(A) = \sum_{1 \leq i \leq n} \text{perm}(A_{1|i}) a_{1i}.$$

4. Let  $A \in \mathcal{M}_n(\mathbb{Z})$ , for some  $n \geq 1$ . Let  $m$  be an integer such that each entry of  $A$  is in the interval  $[-2^m, 2^m]$ . Then

$$\text{perm}(A) \in [-2^{n(m+\log n)}, 2^{n(m+\log n)}].$$

5. Let  $A = (a_{ij}), B = (b_{ij}) \in \mathcal{M}_n(\mathbb{Z})$ , for some  $n \geq 1$ . Define  $E(y) = yA + (1-y)B = (a_{ij}y + b_{ij}(1-y))$  and  $f(y) = \text{perm}(E(y))$ . Then  $\text{perm}(A) = f(1)$  and  $\text{perm}(B) = f(0)$ . Also,  $f \in \mathbb{Z}[y]$ , the degree of  $f$  is at most  $n$ , and for every  $m \in \mathbb{Z}$ ,

$$\text{perm}(E(m)) = f(m).$$

6. Let  $n \geq 1$  and let  $E(y) \in \mathcal{M}_n(\mathbb{Z}[y])$  be such that each entry of  $E(y)$  is a linear function in  $y$ . Let  $m$  be such that the coefficients of each entry of  $E(y)$  are in the interval  $[-2^m, 2^m]$ . Then each coefficient of  $\text{perm}(E(y))$  is in the interval  $[-2^{n(m+2\log n)}, 2^{n(m+2\log n)}]$ .

**Proof** For part 1, see the Bibliographic Notes. To prove part 2, let  $N$  be the nondeterministic Turing machine that, on input  $A = (a_{ij}) \in \mathbb{N}_{n \times n}$ , for some  $n \geq 1$ , behaves as follows:

**Step 1** For each  $i$ ,  $1 \leq \dots \leq n$ ,  $N$  guesses a number  $j_i$ ,  $1 \leq j_i \leq n$ .

**Step 2** Let  $\pi$  be the function that maps each  $i$ ,  $1 \leq i \leq n$ , to  $j_i$ .  $N$  tests whether  $\pi$  is a permutation of  $\{1, \dots, n\}$ . If  $\pi$  does not pass the test,  $N$  rejects  $A$ . Otherwise,  $N$  proceeds to Step 3.

**Step 3**  $N$  sets  $P$  to the product  $P = \prod_{1 \leq i \leq n} a_{i\pi(i)}$ .  $N$  computes  $t = \lceil \log(P + 1) \rceil$ .

**Step 4**  $N$  nondeterministically guesses a string  $y$  of length  $t$ . If the rank of  $y$  in  $\Sigma^t$  is less than or equal to  $P$ , then  $P$  accepts  $A$ . Otherwise,  $P$  rejects  $A$ .

Let  $A$  be an input to  $N$ . Let  $n$  be the dimension of  $A$ . Note that, for every permutation  $\sigma$  of  $\{1, \dots, n\}$ , there is exactly one set of guesses  $j_1, \dots, j_n$  in Step 1 such that the mapping  $\pi$  defined by  $j_1, \dots, j_n$  is  $\sigma$ . Let  $\sigma$  be a permutation of  $\{1, \dots, n\}$ . Suppose that  $N$  selects  $\sigma$  as  $\pi$  in Step 1. Then  $\pi$  passes the test in Step 2, so  $N$  enters Step 3. The number of accepting paths  $N$  produces for  $\sigma$  in Step 3 is  $P$ , which is equal to  $\prod_{1 \leq i \leq n} a_{i\sigma(i)}$ . So, the total number of accepting computation paths of  $N$  on input  $A$  is  $\text{perm}(A)$ . Let  $H$  be the largest entry of  $A$ . Then  $H < 2^{|A|}$ . So, the product  $P$  in Step 3, if  $N$  arrives at Step 3, is less than  $2^{n|A|}$ , so  $t \leq n|A|$ . Since  $n \leq |A|$ ,  $t \leq |A|^2$ . This implies that  $N$  can be made to run in polynomial time.

Part 3 can be proven by routine calculation.

Part 4 holds because the absolute value of the permanent is bounded by

$$n!(2^m)^n \leq n^n 2^{mn} \leq 2^{n(m+\log n)}.$$

To prove part 5, let  $n$ ,  $A$ , and  $B$  be as in the hypothesis. Define  $E(y) = yA + (1 - y)B$  and  $f(y) = \text{perm}(E(y))$ . For all  $m \in \mathbb{Z}$ ,  $f(m) = \text{perm}(E(m))$ . In particular,  $f(0) = \text{perm}(B)$  and  $f(1) = \text{perm}(A)$ . Note that

$$f(y) = \sum_{\pi} \prod_{1 \leq i \leq n} (a_{i\pi(i)}y + b_{i\pi(i)}(1 - y)),$$

where  $\pi$  ranges over all permutations of  $\{1, \dots, n\}$ . For all permutations  $\pi$  and all integers  $i$ ,  $1 \leq i \leq n$ ,  $a_{i\pi(i)}y + b_{i\pi(i)}(1 - y)$  is a linear function in  $y$ . So,  $\prod_{1 \leq i \leq n} (a_{i\pi(i)}y + b_{i\pi(i)}(1 - y))$  is a polynomial in  $y$  of degree at most  $n$ . This implies that  $f(y)$  is a polynomial in  $y$  of degree at most  $n$ .

Part 6 holds because for each  $d$ ,  $0 \leq d \leq n$ , the absolute value of the coefficient of  $y^d$  in  $\text{perm}(E(y))$  is bounded by

$$n!(2^m)^n \binom{n}{d} \leq (n!)^2 (2^m)^n \leq 2^{n(m+2 \log n)}.$$

□

### 6.1.5 An Interactive Proof System for the Permanent

In the rest of the section we prove that  $P^{\#P}$  has an interactive proof system.

**Theorem 6.4**  $P^{\#P} \subseteq \text{IP}$ .



Combining Theorem 6.4 and Toda's Theorem,  $\text{PH} \subseteq \text{P}^{\#\text{P}}$  (see Theorem 4.12), we learn that every language in the polynomial hierarchy has an interactive proof system.

**Corollary 6.5**  $\text{PH} \subseteq \text{IP}$ .

To prove Theorem 6.4, we'll develop an interactive protocol for the permanent function of integer matrices.

Let  $L \in \text{P}^{\#\text{P}}$ . By parts 1 and 2 of Proposition 6.3, it is  $\#\text{P}$ -complete to compute the permanent of matrices whose entries are nonnegative integers. So, we may assume that there is a polynomial time-bounded Turing machine  $M$  that decides  $L$  with  $\text{perm}$  as an oracle. Since  $M$  is polynomial time-bounded, there is a polynomial  $p$  such that for every  $x \in \Sigma^*$   $M$  satisfies the following two conditions:

- Regardless of its oracle,  $M$  on input  $x$  makes at most  $p(|x|)$  queries.
- For each potential query  $A$  of  $M$  on input  $x$ ,  $\dim(A) \leq p(|x|)$  and every entry of  $A$  is in the interval  $[-2^{p(|x|)}, 2^{p(|x|)}]$ .

We will construct an interactive proof system  $(P, V)$  for  $L$ . Let  $x \in \Sigma^*$  be a string whose membership in  $L$  we are testing and  $n = |x|$ . The verifier  $V$  simulates  $M$  on input  $x$  deterministically, and accepts or rejects accordingly. When  $M$  is about to make a query  $A$  to its oracle  $\text{perm}$ , instead of making that query to  $P$ ,  $V$  executes the protocol presented in Fig. 6.2. During the protocol,  $V$  maintains a list of matrix-integer pairs,  $\Lambda = [(B_1, v_1), \dots, (B_m, v_m)]$ , such that  $P$  has promised that for all  $i$ ,  $1 \leq i \leq m$ ,  $\text{perm}(B_i) = v_i$ , where for some  $d \geq m \geq 1$ ,  $B_1, \dots, B_m \in \mathcal{M}_d(\mathbb{Z})$  and  $v_1, \dots, v_m \in \mathbb{Z}$ . At the start of the protocol  $V$  obtains from the prover  $P$  a value  $u$  that  $P$  claims is  $\text{perm}(A)$  and sets  $\Lambda$  to  $[(A, u)]$ . Then  $V$  interacts with  $P$  to reduce the dimension of the matrix entries in  $\Lambda$  to 1. At that point, since the number of pairs in  $\Lambda$  will not exceed the dimension of the matrices, there is only one pair  $(B, v)$  in the list and the prover has promised that  $\text{perm}(B)$ , which is the unique entry of  $B$  by definition, is  $v$ . So,  $V$  checks whether what the prover has promised is correct. If so,  $V$  returns to the simulation of  $M$  assuming that  $\text{perm}(A) = u$ . Otherwise,  $V$  terminates its computation by rejecting the input  $x$ .

We claim that this protocol witnesses that  $L \in \text{IP}$ . To prove our claim we need to show the following:

1. The protocol is complete, i.e., for every  $x \in L$ , there exists a prover  $\tilde{P}$  such that  $V$  accepts  $x$  with probability at least  $\frac{3}{4}$  through interactions with  $\tilde{P}$ .
2. The protocol is sound, i.e., for every  $x \in \bar{L}$ , and every prover  $P$ ,  $V$  accepts  $x$  with probability at most  $\frac{1}{4}$ .
3.  $V$  can be polynomial time-bounded.

---

<p><b>Step 1</b> Send <math>A</math> to <math>P</math> and obtain from <math>P</math> a value <math>u</math>. Set <math>\Lambda</math> to <math>[(A, u)]</math>.</p> <p><b>Step 2</b> (a) Let <math>(B, v)</math> the unique element of <math>\Lambda</math>. Remove <math>(B, v)</math> from <math>\Lambda</math>.          (b) Set <math>d</math> to <math>\dim(B)</math>. If <math>d = 1</math>, then goto Step 4.          (c) For each <math>i</math>, <math>1 \leq i \leq d</math>, send <math>B_{1 i}</math> to <math>P</math> and obtain from <math>P</math> a value <math>v_i</math> that <math>P</math> claims is <math>\text{perm}(B_{1 i})</math>.          (d) Test whether <math>v = \sum_{1 \leq j \leq d} b_{1j} v_j</math>; if the test fails, reject <math>x</math>.          (e) Set <math>\Lambda</math> to <math>[(B_{1 1}, v_1), \dots, (B_{1 d}, v_d)]</math> and proceed to Step 3.</p> <p><b>Step 3</b> Repeat (a)–(f) until <math> \Lambda  = 1</math>:          (a) Remove the first two elements <math>(B, v)</math> and <math>(C, w)</math> from <math>\Lambda</math>.          (b) Let <math>y</math> be a variable and <math>m = \dim(B) (= \dim(C))</math>. Compute the matrix <math>E(y) = (e_{ij}(y)) \in \mathcal{M}_m(\mathbb{Z}[y])</math> defined for all <math>i, j</math>, <math>1 \leq i, j \leq m</math>, by</p> $e_{ij}(y) = b_{ij}y + (1 - y)c_{ij},$ <p style="text-align: center;">where <math>B = (b_{ij})</math> and <math>C = (c_{ij})</math>.          (c) Send <math>E(y)</math> to <math>P</math> and obtain a polynomial <math>f(y) \in \mathbb{Z}[y]</math> that <math>P</math> claims is <math>\text{perm}(E(y))</math>.          (d) Test whether <math>v = f(1)</math> and <math>w = f(0)</math>; if the test fails, then reject <math>x</math>.          (e) Pick <math>r \in \{0, \dots, 2^{p(n)} - 1\}</math> under discrete uniform distribution and compute <math>D = E(r) = (e_{ij}(r))</math> and <math>z = f(r)</math>.          (f) Append <math>(D, z)</math> into <math>\Lambda</math>.          Return to Step 2.</p> <p><b>Step 3</b> Test whether <math>v</math> is the unique entry in <math>B</math>. If the test succeeds, return to the simulation with <math>u</math> and if the test fails, reject <math>x</math>.</p>
---

---

**Fig. 6.2** Interactive protocol for the permanent function

---

**6.1.5.1 Completeness of the Protocol.** In order to see why the protocol is complete, let  $x$  be an arbitrary member of  $L$ . Let  $\tilde{P}$  be the prover that always provides correct answers to the queries of  $V$ . Since  $\tilde{P}$  will always provide correct answers, regardless of the probabilistic choices of  $V$ , all tests in Fig. 6.2 will succeed. This implies that  $V$  on  $x$  through interactions with  $\tilde{P}$  will follow the computation path that  $M$  on input  $x$  would with perm as the oracle, and thus, will accept  $x$ . So, the probability that  $V$  on input  $x$  accepts with  $\tilde{P}$  as the prover is 1. Thus, the protocol is complete.

**6.1.5.2 Soundness of the Protocol.** In order to prove that the protocol is sound, let  $x$  be an arbitrary element in  $\bar{L}$ . Let  $\rho$  be the maximum probability of acceptance that  $V$  on input  $x$  has through interactions with any prover. We claim that  $\rho < \frac{1}{4}$ . We prove the claim by contradiction. Assume, to the contrary, that  $\rho \geq \frac{1}{4}$ . Let  $P$  be a prover that achieves  $\rho$  as the acceptance probability of  $V$  on input  $x$ .

Note that, to achieve a nonzero probability of acceptance,  $P$  has to provide an incorrect answer in Step 1 of the protocol to at least one query that  $V$  produces on input  $x$ . To see why, assume that  $P$  provides a correct answer in Step 1 of the protocol to each query that  $V$  produces on input  $x$ . Take an arbitrary computation path,  $\pi$ , of  $V$  on input  $x$  through interaction with

*P*. Suppose that  $V$  on input  $x$  along path  $\pi$  terminates before  $V$  completes its simulation of  $M$  on  $x$ . Since  $V$  never accepts during the execution of the protocol, this implies that  $V$  rejects  $x$  along path  $\pi$ , and thus,  $\pi$  does not contribute to the probability of acceptance of  $V$  on input  $x$ . So, suppose that  $V$  on input  $x$  along path  $\pi$  completes its simulation of  $M$  on input  $x$ . By assumption,  $P$  provides the correct answer in Step 1 to each query that  $V$  on input  $x$  makes. So, the answers that  $V$  obtains from  $P$  along path  $\pi$  are those  $M$  on input  $x$  would receive with perm as the oracle. This implies that the computation of  $M$  on input  $x$  that is simulated by  $V$  on input  $x$  along path  $\pi$  is precisely that  $M$  on input  $x$  with oracle perm. Since  $M$  is deterministic and  $x \in L$ , this implies that  $V$  on input  $x$  along path  $\pi$  rejects. Hence, regardless of whether  $V$  finishes its simulation of  $M$  on input  $x$ ,  $V$  on input  $x$  along path  $\pi$  rejects. This implies  $\rho = 0$ , a contradiction.

By the above discussion, suppose that  $V$  has made a query  $A$  having dimension greater than 1 to  $P$  and  $P$  has provided a value  $u \neq \text{perm}(A)$  to the query  $A$  in Step 1 of the protocol. Then  $V$  sets the value of  $\Lambda$  to  $\{(A, u)\}$ . We will examine the subsequent execution of Steps 2 and 3. To simplify our discussion, call a matrix-integer pair  $(B, v)$  in  $\Lambda$  *correct* if  $v = \text{perm}(B)$ ; otherwise, call the pair *incorrect*. Call  $\Lambda$  *correct* if every matrix-integer pair in it is correct; otherwise, call  $\Lambda$  *incorrect*. Note that the following three conditions hold:

- Immediately after Step 1,  $\Lambda$  is incorrect because its unique element  $(A, u)$  is incorrect by our supposition.
- In  $V$  halts before reaching Step 4, then  $V$  does so by rejecting  $x$ .
- In Step 4,  $V$  returns to its simulation of the machine  $M$  on input if  $\Lambda$  is correct and rejects  $x$  otherwise.

We will show that the probability that  $V$  reaches Step 4 and  $\Lambda$  becomes correct before  $V$  reaches Step 4 is less than  $\frac{1}{4}$ .

First suppose that  $\Lambda$  is incorrect at the beginning of Step 2. Let  $(B, v)$  be the unique matrix-integer pair in  $\Lambda$  and  $d$  be the dimension of  $B$ . We can assume that  $d > 1$ . Otherwise,  $V$  will immediately jump to Step 4 and the incorrectness of  $\Lambda$  will be preserved. Suppose that  $V$  has obtained from  $P$  in the subsequent Step 2(c)  $v_1, \dots, v_d$  as the permanents of the minors  $B_{1|1}, \dots, B_{1|d}$ , respectively. In the subsequent Step 2(d),  $V$  tests whether  $v = \sum_{1 \leq i \leq d} b_{1i} v_i$ . Suppose that this test succeeds. Then, since  $\text{perm}(B) = \sum_{1 \leq i \leq d} b_{1i} \text{perm}(B_{1|i})$  and  $v \neq \text{perm}(B)$ , it must be the case that for at least one  $i$ ,  $1 \leq i \leq d$ ,  $\text{perm}(B_{1|i}) \neq v_i$ . Thus,  $\Lambda$  at the beginning of subsequent Step 3 is incorrect. In other words, if  $V$  does not reject  $x$  in Step 2, then the property that  $\Lambda$  is incorrect should be preserved during Step 2, and  $\Lambda$  remains incorrect at the beginning of Step 3.

So, suppose that the list  $\Lambda$  is incorrect at the beginning of Step 3. If  $\Lambda$  has only one element,  $V$  immediately returns to Step 2 without modifying  $\Lambda$ , so the incorrectness of  $\Lambda$  is preserved. So suppose that  $\Lambda$  has at least two elements. Let  $(B, v)$  and  $(C, w)$  be the pairs that  $V$  pops from  $\Lambda$  in

Step 3(a). Suppose that there remains at least one incorrect pair in  $\Lambda$ . If so, since the incorrect pair will not be removed from  $\Lambda$  in the subsequent Steps 3(b) through 3(f), the incorrectness of  $\Lambda$  will be preserved in the subsequent Steps 3(b) through 3(f). So, suppose that there remains no incorrect pair in  $\Lambda$  after popping  $(B, v)$  and  $(C, w)$ . Since  $\Lambda$  is incorrect, it must be the case that at least one of  $(B, v)$  and  $(C, w)$  is incorrect. Let  $f$  be the polynomial that  $P$  provides for the permanent of  $E(y) = yB + (1 - y)C$ . Suppose that  $f$  survives the test in Step 3(d). Then  $f(1) = v$  and  $f(0) = w$ . Since  $E$  satisfies  $\text{perm}(B) = \text{perm}(E(1))$  and  $\text{perm}(C) = \text{perm}(E(0))$  and, by our assumption, either  $v \neq \text{perm}(B)$  or  $w \neq \text{perm}(C)$ , we have  $f(y) \neq \text{perm}(E(y))$ . Since  $\dim(B) = \dim(C) \leq \dim(A) \leq p(n)$ ,  $\text{perm}(E(y))$  is a polynomial of degree at most  $p(n)$ . So, by the Polynomial Interpolation Lemma (Lemma 6.2), there are at most  $p(n)$  many  $r$  for which  $\text{perm}(E(r)) = f(r)$ . Thus, the probability that the pair  $(D, z)$  that  $V$  produces from  $(B, v)$  and  $(C, w)$  is correct is at most  $\frac{p(n)}{2^{p(n)}}$ . This implies that the probability that the incorrectness of  $\Lambda$  is preserved during a single run of the loop body is at least  $1 - \frac{p(n)}{2^{p(n)}}$ . The number of times that the loop body of Step 3 is executed is

$$\sum_{1 \leq i \leq \dim(A)-1} (i-1) \leq \frac{p(n)(p(n)-1)}{2},$$

so the probability that the unique pair in  $\Lambda$  in Step 4 is incorrect is at least

$$\left(1 - \frac{p(n)}{2^{p(n)}}\right)^{\frac{p(n)(p(n)-1)}{2}} \geq 1 - \frac{p^3(n)}{2^{p(n)}}.$$

So,  $P$  fails with probability at least  $1 - \frac{p^3(n)}{2^{p(n)}}$ . This is greater than  $\frac{3}{4}$  provided  $p(n) \geq 14$ , so  $\rho < \frac{1}{4}$ . This is a contradiction. Thus, the protocol is sound.

**6.1.5.3 Running-Time Analysis.** To prove that  $V$  is polynomial time-bounded, we may assume that in Step 3(c), the prover returns the polynomial  $f$  to  $V$  by providing integers  $\alpha_0, \dots, \alpha_d$ , such that  $f(r) = \alpha_d r^d + \dots + \alpha_1 r + \alpha_0$ , where  $d = \dim(E(r))$ . Let  $A$  be a query of  $M$  and  $d = \dim(A)$ . Assuming that  $V$  reaches Step 4,  $V$  executes Step 3(e)  $m = \sum_{2 \leq i \leq d} (i-1) = \frac{d(d-1)}{2}$  times. For each  $i$ ,  $0 \leq i \leq m$ , let  $t_i$  be the smallest integer  $\ell$  such that the interval  $[-2^\ell, 2^\ell]$  contains all the integers that have been seen either as coefficients of polynomial entries in  $E(y)$  or entries of matrices in the list  $\Lambda$  by the end of the  $i$ th execution of Step 3(c). Then  $t_0 \leq p(n)$ . For every  $i$ ,  $1 \leq i \leq m$ , each entry of  $E(y)$  in the  $i$ th execution of Step 3(c) takes the form of  $b + (1 - y)c$  such that  $|b|, |c| \leq 2^{t_{i-1}}$  and a random value assigned to  $y$  is selected from  $[0, 2^{p(n)} - 1]$ . This implies that for every  $i$ ,  $1 \leq i \leq m$ ,  $t_i \leq t_{i-1} + p(n)$ . Thus,  $t_m \leq (m+1)p(n) \leq (\frac{d(d-1)}{2} + 1)p(n) \leq d^2 p(n) \leq p^3(n)$ . Then, by part 4 of Proposition 6.3, for every integer matrix  $B$  that appears during the protocol,  $\log |\text{perm}(B)|$  is at most

$$p(n)(p^3(n) + \log p(n)).$$

This is less than  $p^5(n)$  for  $p(n) \geq 2$ . Also, by part 6 of Proposition 6.3, for each polynomial  $f(y)$  appearing during the protocol, the log of each coefficient of  $f$  is at most

$$p(n)(p^3(n) + 2 \log p(n)).$$

This is less than  $p^5(n)$  for  $p(n) \geq 2$ . Thus, all the integers that appear during the execution of the protocol are at most  $p^5(n)$  bits long. Modify  $V$  so that it will spend at most  $p^5(n)$  steps for reading a number supplied by the prover. Then  $V$  will be polynomial time-bounded and able successfully to execute the protocol for every input  $x \in L$  while interacting with  $\tilde{P}$ . This concludes the proof of Theorem 6.4.

## 6.2 Enumerators for the Permanent

$\text{PH} \subseteq \text{P}^{\#\text{P}}$  (by Theorem 4.12) and  $\text{perm}$  is complete for  $\#\text{P}$  (by part 4 of Proposition 6.3), so we may not hope to be able to compute the permanent function in polynomial time unless  $\text{PH} = \text{P}$ . Then we ask: Is there an easy way to generate, given an integer matrix  $A$ , a short list of candidates for  $\text{perm}(A)$  so that one of the candidates is the correct value of  $\text{perm}(A)$ ? We combine the polynomial interpolation technique and the self-reducibility of the permanent (i.e., the permanent of an  $n \times n$  matrix with  $n \geq 2$  can be reduced to the problem of computing the permanents of all its minors), we show that we cannot hope to have such an enumeration algorithm either, unless  $\text{P} = \text{PP}$ .

We first formalize the concept of candidate generation.

**Definition 6.6** *Let  $f : \Sigma^* \rightarrow \mathbb{N}$  be a function. A function  $E$  is an enumerator for  $f$  if for every  $x \in \Sigma^*$  there exist some  $m \geq 1$  and some  $a_1, \dots, a_m \in \mathbb{N}$  such that*

1.  $E(x) = \langle m, a_1, \dots, a_m \rangle$  and
2.  $f(x) \in \{a_1, \dots, a_m\}$ .

**Theorem 6.7** *If there is a polynomial-time computable enumerator for  $\text{perm}$ , then  $\text{perm} \in \text{FP}$ .*

**Proof** Suppose that there is a polynomial-time enumerator  $E$  for the permanent function. Let  $n \geq 1$ . Let  $A = (a_{ij})$  be an  $n \times n$  matrix whose permanent we want to compute. Let  $m$  be the smallest integer such that each entry of  $A$  has absolute value at most  $2^m$ . Then,  $|A|$ , the encoding length of  $A$ , is at least  $n^2 + m$ . By part 4 of Proposition 6.3,  $|\text{perm}(A)| \leq 2^{n(m + \log n)}$ . For all  $n, m \geq 1$ ,  $(n^2 + m)^2 > n(m + \log n)$ . So,  $|\text{perm}(A)| < |A|^2$ . Let  $s = 2\lceil \log |A| \rceil + 1$ . Then  $2^s > 2|\text{perm}(A)|$ . We reduce the problem of computing  $\text{perm}(A)$  to the problem of computing  $\text{perm}(A) \bmod Q_i$  for some  $s$

distinct primes  $Q_1, \dots, Q_s$ . Once these values have been computed, since  $Q_1 \cdots Q_s \geq 2^s > 2|\text{perm}(A)|$ , using the Chinese Remainder Theorem, we can recover the exact value of  $\text{perm}(A)$ .

To compute  $\text{perm}(A) \bmod Q$  for a prime number  $Q$ , we execute the following algorithm that uses a subroutine  $\mathcal{F}$ .

**Step 1** Set  $A_n$  to the  $n \times n$  matrix such that for all integers  $i$  and  $j$ ,  $1 \leq i, j \leq n$ , its  $(i, j)$ th entry is  $a_{ij} \bmod Q$ .

**Step 2** Set  $A_n$  to the  $n \times n$  matrix such that for Execute the following for  $i = n - 1, \dots, 1$ .

(a) Construct from  $A_{i+1}$  an  $i \times i$  matrix  $B_i(X)$ , defined as

$$\sum_{1 \leq k \leq i+1} \delta_k(X) a_{1k} A_{1|k}.$$

Here for every  $k$ ,  $1 \leq k \leq i+1$ ,  $a_{1k}$  is the  $(1, k)$ th entry of  $A_{i+1}$ ,  $A_{1|k}$  denotes the  $(1, k)$ th minor of  $A_{i+1}$ , and

$$\delta_k(X) = \prod_{j \in \{1, \dots, i+1\} - \{k\}} (X - j)(i - j)^{-1},$$

where for all  $j \in \{1, \dots, i+1\} - \{k\}$ ,  $(i - j)^{-1}$  is the multiplicative inverse of  $i - j$  in  $\mathbb{Z}_Q$ .

(b) Present  $B_i(X)$  to the subroutine  $\mathcal{F}$  to obtain candidate polynomials  $g_1, \dots, g_t$  for  $\text{perm}(B_i(X))$ , where  $t^2 n^2 \leq Q$  and these polynomials are pairwise distinct modulo  $Q$ . Set  $r_i$  to the smallest  $r \in \{0, \dots, Q - 1\}$  such that for all  $j, k$ ,  $1 \leq j < k \leq t$ ,  $g_j(r) \not\equiv g_k(r) \pmod{Q}$ .

(c) Set  $A_i$  to  $B_i(r_i) \bmod Q$ .

**Step 3** Compute  $v_1 = \text{perm}(A_1)$ , where  $v_1$  is the only entry of  $A_1$ .

**Step 4** Execute the following for  $i = 1, \dots, n - 1$ .

(a) Let  $g_1, \dots, g_t$  be the candidates generated for  $\text{perm}(B_i(X))$  in Step 2(b). Find the unique  $k$ ,  $1 \leq k \leq t$ , such that  $g_k(r_i) \bmod Q = v_i$ .

(b) Compute  $v_{i+1}$  as  $(\sum_{1 \leq j \leq i+1} g_k(j)) \bmod Q$ .

**Step 5** Output  $v_n$  as  $\text{perm}(A) \bmod Q$ .

Note that for every  $i$ ,  $1 \leq i \leq n - 1$ , in Step 2(a), each entry of  $B_i(X)$  is an element of  $\mathbb{Z}_Q[X]$  of degree at most  $i$ , so  $\text{perm}(B_i(X))$  is a polynomial in  $\mathbb{Z}_Q[X]$  and has degree at most  $i^2 \leq n^2$ . Note also that for every  $i$ ,  $1 \leq i \leq n - 1$ , it holds that  $\text{perm}(A_{i+1}) \equiv \sum_{1 \leq t \leq i+1} \text{perm}(B_i(t)) \pmod{Q}$ . In Step 2(b) above, since  $\text{perm}(B_i(X))$  is a polynomial of degree at most  $i^2 \leq n^2$ , by Lemma 6.32, for each pair  $(j, k)$ ,  $1 \leq j < k \leq i$ , there are at most  $n^2$  many values of  $r$  in  $\{0, \dots, Q - 1\}$  such that  $g_j(r) \equiv g_k(r) \pmod{Q}$ . Since there are  $\binom{t}{2} < t^2$  combinations of  $j$  and  $k$ ,

$$\begin{aligned} & ||\{r \mid r \in \{0, \dots, Q - 1\} \wedge \\ & (\exists j, k)[1 \leq j < k \leq t \wedge g_j(r) \equiv g_k(r) \pmod{Q}]\} || < t^2 n^2. \end{aligned}$$

So, if  $t^2n^2 \leq Q$ , then there is at least one  $r \in \{0, \dots, Q-1\}$  such that for all  $j, k$ ,  $1 \leq j < k \leq t$ ,  $g_j(r) \not\equiv g_k(r) \pmod{Q}$ . Thus, for all  $i$ ,  $1 \leq i \leq n-1$ , the value  $r_i$  is defined as long as the number  $t$  of candidates that  $\mathcal{F}$  generates satisfies  $t^2n^2 \leq Q$ .

To describe how  $\mathcal{F}$  works we need to introduce a new function  $h$ . The domain of  $h$ ,  $\text{domain}(h)$ , is the set of all square matrices  $N$  such that each entry of  $N$  is a polynomial belonging to  $\mathbb{N}[X]$  having degree at most  $\dim(N)$ . For all  $N \in \text{domain}(h)$   $\text{perm}(N)$  is a polynomial belonging to  $\mathbb{N}[X]$  having degree at most  $(\dim(N))^2$ . The value of  $h(N)$  is an integer that encodes the coefficients of  $\text{perm}(N)$ . For each matrix  $N \in \text{domain}(h)$ , let  $\ell(N)$  denote the smallest integer  $t$  such that every coefficient of every entry of  $N$  is less than  $2^t$ . Then there exists some integer constant  $c > 0$  such that, for all matrices  $N \in \text{domain}(h)$  and all  $i$ ,  $0 \leq i \leq (\dim(N))^2$ , the coefficient  $C_i$  of  $X^i$  in  $\text{perm}(N)$  is less than

$$m!(m+1)^m(2^{\ell(N)})^m < 2^{c(m+\ell(N))}. \quad (6.1)$$

Then we define the value of  $h(N)$  to be

$$\sum_{0 \leq i \leq m^2} (2^{c(m+\ell(N))})^i C_i.$$

Then by equation 6.1, the coefficients  $C_0, \dots, C_{m^2}$  can be recovered from  $h(N)$ . Furthermore, we claim that  $h$  is a #P function. To see why, consider a nondeterministic Turing machine  $U$  that, on input  $N$ , does the following:

- $U$  tests whether  $N \in \text{domain}(h)$ . If the test fails,  $U$  immediately rejects  $N$ .
- $U$  guesses  $i \in \{0, \dots, (\dim(N))^2\}$  and for each  $j$ ,  $1 \leq j \leq \dim(N)$ , an integer  $d_j \in \{0, \dots, \dim(N)\}$ .  $U$  tests whether  $i = \sum_{1 \leq j \leq \dim(N)} d_j$ . If the test fails,  $U$  immediately rejects  $N$ .
- For each  $j$ ,  $1 \leq j \leq \dim(N)$ ,  $U$  guesses an integer  $p_j \in \{1, \dots, \dim(N)\}$ . Let  $\pi$  be the mapping from  $\{1, \dots, \dim(N)\}$  to itself defined for all  $i$ ,  $1 \leq i \leq \dim(N)$ , by  $\pi(i) = p_i$ .  $U$  tests whether  $\pi$  is a permutation. If the test fails,  $U$  immediately rejects  $N$ .
- $U$  computes the product  $P$  for all  $j$ ,  $1 \leq j \leq \dim(N)$ , of the coefficient of  $X^{d_j}$  in the  $(j, \pi(j))$ th entry of  $N$ .
- $U$  guesses an integer  $k$ ,  $1 \leq k \leq 2^{c(\dim(N)+\ell(N))i} P$ , and then accepts  $N$  otherwise.

It is easy to see that  $U$  can be polynomial time-bounded and, for all  $N \in \text{domain}(h)$ ,  $\#\text{acc}_U(N) = h(N)$ . Thus,  $h \in \#P$ .

Since  $h$  is a member of #P, by part 1 of Proposition 6.3, there is a polynomial time reduction to  $(R_1, R_2)$  such that for every  $N \in \text{domain}(h)$   $h(N) = R_2(N, \text{perm}(R_1(N)))$ . Define the action of the oracle  $\mathcal{F}$  as follows: On input  $N \in \text{domain}(h)$ ,  $\mathcal{F}$  does the following:

- $\mathcal{F}$  computes  $W = R_1(N)$  and evaluates  $E(W)$  to obtain candidates  $v_1, \dots, v_p$  for  $\text{perm}(W)$ .

- For each  $i$ ,  $1 \leq i \leq p$ ,  $\mathcal{F}$  computes the  $i$ th polynomial  $g_i$  from  $R_2(W, v_i)$  by taking modulo  $Q$ .
- Trash all the candidate polynomials of degree greater than  $m^2$ , where  $m$  is the dimension of  $N$ . Also, if a polynomial is repeated in the list, eliminate duplicates.
- Return the list of remaining polynomials.

Since  $R_1, R_2$ , and  $E$  are all polynomial time computable, the procedure  $\mathcal{F}$  runs in polynomial time. So, there is some  $\alpha > 0$  such that, for all  $N \in \text{domain}(h)$ , the number of candidates that  $\mathcal{F}$  produces on input  $N$  is bounded by  $(\dim(N) + \ell(N))^\alpha$ .

During the execution of the procedure for computing  $\text{perm}(A) \bmod Q$ , for each query  $N$  made to  $\mathcal{F}$ ,  $\dim(N) \leq n \leq |A|$  and  $\ell(N) \leq \log Q \leq |A|$ . Suppose that the prime numbers  $Q_1, \dots, Q_s$  lie in the interval  $[|A|^\beta, |A|^\gamma]$  for some  $\beta, \gamma \geq 1$ . Then the number of candidates  $t$  that  $\mathcal{F}$  outputs at each query during the computation is at most

$$(|A| + \gamma \log |A|)^\alpha < |A|^{\alpha+1}$$

for all but finitely many  $A$ . The requirement for  $Q$  is that  $t^2 n^2 \leq Q$ . So,  $|A|^{2\alpha+4} \leq |A|^\beta$  has to be met. Let  $\beta = 2\alpha + 4$  and  $\gamma = 2\beta$ . The number of primes we need is  $|A|^2$ . The following theorem, which we state without a proof, is well known and useful.

**Theorem 6.8 (The Prime Number Theorem)** *For every integer  $l \geq 1$  there are at least  $2^l$  primes in the interval  $[2^l, 2^{2l}]$ .*

Since  $\beta \geq 1$  and  $\gamma = 2\beta$ , by Theorem 6.8, there are at least  $|A|^2$  primes in the interval  $[|A|^\beta, |A|^\gamma]$ . Since the largest prime we deal with is at most  $|A|^\gamma$ , by a trivial division procedure we can find in time polynomial in  $|A|$  the primes we need. This implies that the permanent is polynomial time computable. This proves the theorem.  $\square$

## 6.3 IP = PSPACE

In this section we prove the following result.

**Theorem 6.9**  $\text{IP} = \text{PSPACE}$ .

We divide the proof into two parts:  $\text{IP} \subseteq \text{PSPACE}$  and  $\text{PSPACE} \subseteq \text{IP}$ .

### 6.3.1 $\text{IP} \subseteq \text{PSPACE}$

**Lemma 6.10**  $\text{IP} \subseteq \text{PSPACE}$ .



**Proof** Let  $L \in \text{IP}$ . Take an interactive proof system for  $L$ . Let  $V$  be the verifier and  $p$  be a polynomial bounding the runtime of  $V$ . Without loss of generality, we may assume that the prover provides a single-bit answer to each query of  $V$ . We may also assume that there exist polynomials  $m$ ,  $q$ , and  $r$  such that, for every input  $x$ , the verifier makes exactly  $q(|x|)$  queries to the prover, each having length  $m(|x|)$ , and tosses exactly  $r(|x|)$  coins before the first query, after the last query, and between every two consecutive queries.

We can assume that the objective of  $P$  is to maximize the probability that  $V$  accepts, regardless of whether the input belongs to  $L$  or not. To see why, suppose that the input belongs to  $L$ . The completeness condition requires that there is a prover that makes  $V$  accept with probability more than  $\frac{3}{4}$ , and this is the same as requiring that the highest acceptance probability that is achieved by any prover is more than  $\frac{3}{4}$ . Next suppose that the input does not belong to  $L$ . Then the soundness condition requires that regardless of the protocol of the prover, the acceptance probability of  $V$  is less than  $\frac{1}{4}$ . This is the same as requiring that the highest probability that is achievable is less than  $\frac{1}{4}$ .

We claim that the highest acceptance probability can be achieved by a prover that works deterministically. Here the reader should be cautioned that deterministic provers are not necessarily oracles, for provers can select their answers based on the history of communication.

To prove the claim, first note that we can assume that there is a recursive function  $f : \Sigma^* \rightarrow \mathbb{N}$  such that, for every  $x \in \Sigma^*$ ,  $V$  on input  $x$  asks exactly  $f(x)$  queries to the prover, regardless of its coin tosses and of the prover. Suppose there is no such  $f$  exists. Since  $V$  witnesses that  $L \in \text{IP}$ , there is a prover  $P$  such that, for every  $x \in L$ ,  $V$  on input  $x$  runs for at most  $p(|x|)$  regardless of its coin tosses and of the prover. Define  $V'$  to be the machine that, on input  $x$ , simulates  $V$  on input  $x$  for at most  $p(|x|)$  steps, i.e., if  $V$  on input  $x$  along the simulated path attempts to make the  $(p(|x|) + 1)$ st step,  $V'$  aborts the simulation. While executing the simulation  $V'$  counts the number of queries that  $V$  on input  $x$  makes to the prover along the simulated path of  $V$  on input  $x$ . When the simulation is either completed or aborted,  $V'$  adds dummy queries (e.g., about the empty string) to make the total number of queries equal to  $p(|x|)$  and then accepts if  $V$  on input  $x$  along the simulated path accepts. For all  $x \in \Sigma^*$ , the number of queries that  $V'$  on input  $x$  makes is  $p(|x|)$ . For all  $x \in L$ , the probability that  $V'$  on input  $x$  accepts through interactions with  $P$  is equal to the probability that  $V$  on input  $x$  accepts through interactions with  $P$ . For all  $x \in \bar{L}$  and for all prover  $P'$ , the probability that  $V'$  on input  $x$  accepts through interactions with  $P'$  is not greater than the probability that  $V$  on input  $x$  accepts through interactions with  $P'$ . So,  $V'$  is an interactive proof system for  $L$ .

Now, suppose that  $V$  has just made its last query to the prover. Since it can compute the function  $f$  using its unlimited computational power, the prover knows that this is the last query of  $V$ . Let  $\rho$  and  $\sigma$  respectively be

the highest probability of acceptance that can be achieved beyond this point provided that the prover answers with a 0 and provided that the prover answers with a 1. These two probabilities are well-defined, since the runtime of  $V$  is bounded, so using its unlimited computational power, the prover can calculate them. Parameterize the strategy of the prover at this very point with  $\alpha$ ,  $0 \leq \alpha \leq 1$ , in such a way that it provides a 0 as the answer with probability  $\alpha$ . Then the overall probability that  $V$  accepts beyond this point is

$$\alpha\rho + (1 - \alpha)\sigma = \sigma + (\rho - \sigma)\alpha.$$

This is maximized at  $\alpha = 1$  if  $\rho \geq \sigma$  and at  $\alpha = 0$  if  $\rho < \sigma$ . So, in order to maximize this amount, the prover has only deterministically to answer with a 0 if  $\rho \geq \sigma$  and with a 1 otherwise. Thus, the strategy of the prover at this point could be deterministic. Since the same argument could be made for any “query” point, working up from the last query to the first query, we can argue that the entire strategy of the prover could be made deterministic without decreasing the probability that  $V$  accepts.

For each  $x \in \Sigma^*$ , let  $H_x$  denote the set of all possible communication histories between  $V$  and some prover. Here a communication history between  $V$  and a prover is the record of all the queries and answers exchanged between them before some computational step. More precisely, for each  $x \in \Sigma^*$ , the following strings constitute  $H_x$ :

- The empty string  $\lambda$ .
- All strings of the form  $y_1\#b_1\$ \cdots \$y_k\#b_k$  for some  $k$ ,  $1 \leq k \leq p(|x|)$ ,  $y_1, \dots, y_k \in \Sigma^{m(|x|)}$ ,  $b_1, \dots, b_k \in \{0, 1\}$ . Here for each  $i$ ,  $1 \leq i \leq k$ ,  $y_i$  is the  $i$ th query of  $V$  to the prover and  $b_i$  is the prover’s answer to  $y_i$ .
- The strings of the form  $y_1\#b_1\$ \cdots \$y_{k-1}\#b_{k-1}\$y_k$  for some  $k$ ,  $1 \leq k \leq p(|x|)$ ,  $y_1, \dots, y_k \in \Sigma^{m(|x|)}$ ,  $b_1, \dots, b_{k-1} \in \{0, 1\}$ . Here for each  $i$ ,  $1 \leq i \leq k$ ,  $y_i$  is the  $i$ th query of  $V$  to the prover, for each  $i$ ,  $1 \leq i \leq k-1$ ,  $b_i$  is the prover’s answer to  $y_i$ , and the answer to  $y_k$  is yet to be given.

For all  $x \in \Sigma^*$  and  $w \in H_x$ , define  $R(x, w)$  to be the maximum probability that  $V$  on input  $x$  accepts when the history of communication has  $w$  as a prefix. Then  $R(x, \lambda)$  is the highest acceptance probability that  $V$  on input  $x$  through interactions with any prover. So, for all  $x \in \Sigma^*$ ,

$$x \in L \iff R(x, \lambda) \geq \frac{3}{4}.$$

To prove that  $L \in \text{PSPACE}$  it now suffices to show that  $R$  is polynomial-space computable.

Consider the procedure  $\text{RCOMP}$ , described in Fig. 6.3, for computing  $R(x, w)$  given  $x \in \Sigma^*$  and  $w \in H_x$ .

It is easy to see that the procedure works correctly. Let us analyze the space requirement of this procedure. For all  $x \in \Sigma^*$ ,  $\text{RCOMP}(x, \lambda)$  has recursion depth  $2q(|x|)$ . When a recursive call is made in either Step 2 or Step 3(b),

---

<p><b>Step 1</b> If <math>w</math> is of the form <math>y_1 \# b_1 \\$ \cdots \\$ y_{q( x )} \# b_{q( x )}</math>, then do the following:</p> <ul style="list-style-type: none"> <li>(a) Set <math>S</math> to 0.</li> <li>(b) For each binary string <math>\pi</math> of length <math>(q( x ) + 2)r( x )</math>, do the following: <ul style="list-style-type: none"> <li>(i) Simulate the computation of <math>V</math> on input <math>x</math> along path <math>\pi</math> assuming that for each <math>i</math>, <math>1 \leq i \leq q( x )</math>, the prover's answer to the <math>i</math>th query is <math>b_i</math>,</li> <li>(ii) Check whether <math>V</math> accepted in the simulation and whether for every <math>i</math>, <math>1 \leq i \leq q( x )</math>, the <math>i</math>th query of <math>V</math> in the simulation was <math>y_i</math>.</li> <li>(iii) If both tests succeed, increment <math>S</math> by 1.</li> </ul> </li> <li>(c) Return <math>S/2^{(q( x )+2)r( x )}</math>.</li> </ul> <p><b>Step 2</b> If <math>w</math> is of the form <math>y_1 \# b_1 \\$ \cdots \\$ y_k</math>, then return <math>\max\{\text{RCOMP}(x, w\#0), \text{RCOMP}(x, w\#1)\}</math>.</p> <p><b>Step 3</b> If either <math>w = \lambda</math> or <math>w</math> is of the form <math>y_1 \# b_1 \\$ \cdots \\$ y_k \# b_k</math> for some <math>k &lt; q( x )</math>, then do the following:</p> <ul style="list-style-type: none"> <li>(a) Set <math>S</math> to 0.</li> <li>(b) For each string <math>z</math> of length <math>m( x )</math>, compute <math>\rho = \text{RCOMP}(x, w\\$z)</math> and add <math>\rho</math> to <math>S</math>.</li> <li>(c) Return <math>S</math>.</li> </ul>
---

---

**Fig. 6.3** Algorithm RCOMP for computing  $R$

---

some pieces of information need to be stored: the current location in the RCOMP program, the sum  $S$  (Step 3 only), the current value of  $\rho$  (Step 3 only), which of the two recursive calls is being executed (Step 2 only), and the output of the first recursive call in the case when the second recursive call is about to be made in Step 2. Since the total number of coin tosses of  $V$  is  $(q(|x|) + 2)r(|x|)$ ,  $R$  has precision of  $(q(|x|) + 2)r(|x|)$  bits. So, the amount of information to be stored is  $\mathcal{O}((q(|x|) + 2)r(|x|)) = \mathcal{O}(q(|x|)r(|x|))$ . Thus, the entire procedure requires  $\mathcal{O}(q(|x|)^2 r(|x|))$  space. Hence, RCOMP is a polynomial-space algorithm, and thus,  $L \in \text{PSPACE}$ .  $\square$

### 6.3.2 PSPACE $\subseteq$ IP

Now we prove the other inclusion of Theorem 6.9.

**Lemma 6.11** PSPACE  $\subseteq$  IP.

We first provide a brief overview of the proof of Lemma 6.11. Let  $L$  be an arbitrary language in PSPACE. Let  $D$  be a polynomial space-bounded Turing machine witnessing that  $L \in \text{PSPACE}$ . We develop a protocol for verifying computation of  $D$ . The idea behind the protocol is Savitch's Theorem, which states that for all space-constructible function  $S(n) = \Omega(\log n)$ , nondeterministic  $S(n)$ -space-bounded computation can be deterministically simulated in space  $S^2(n)$ . To describe the theorem we use the concept of configurations. A configuration of the machine  $D$  describes the contents of its tapes, the positions of its head, and its state. Here we assume that  $D$  is a one-tape machine

and that there exists a polynomial  $s$  such that, for all  $x \in \Sigma^*$ , the configuration at each step of  $D$  on input  $x$  can be encoded as a string in  $\Sigma^{s(|x|)}$ . Also, we assume that, for each  $x \in \Sigma^*$ , if  $D$  on input  $x$  accepts then  $D$  does so by erasing all the tape squares it has ever visited and moving its head to position 1. This implies that for every  $x \in \Sigma^*$ , there is a unique accepting configuration of  $D$  on input  $x$ . Finally, assume that there is a polynomial  $r$  such that, for all  $x \in \Sigma^*$ ,  $D$  on input  $x$  halts at step  $2^{r(|x|)}$ . Then, for all  $x \in \Sigma^*$ ,  $x \in L$  if and only if the unique accepting configuration of  $D$  on input  $x$  is reached in exactly  $2^{r(|x|)}$  steps.

Suppose we are testing the membership of  $x \in \Sigma^*$  in  $L$ . We define a family of polynomials  $R_j$ ,  $0 \leq j \leq r(|x|)$ , with the following property: For all  $j$ ,  $0 \leq j \leq r(|x|)$ , and all configurations  $C$  and  $C'$  of  $D$  on input  $x$ ,

$$R_j(w, w') = \begin{cases} 1 & \text{if } C' \text{ is reachable from } C \text{ by } D \\ & \text{in exactly } 2^j \text{ steps,} \\ 0 & \text{otherwise.} \end{cases}$$

Here  $w$  and  $w'$  are respectively the encoding of  $C$  in  $\Sigma^{s(|x|)}$  and the encoding of  $C'$  in  $\Sigma^{s(|x|)}$ . Let  $C_0$  be the initial configuration of  $D$  on input  $x$ . Let  $C_1$  be the accepting configuration of  $D$  on input  $x$ , in which the tape head of  $D$  is at position 1 and each tape square has a blank. Let  $w_0$  and  $w_1$  be respectively the encoding of  $C_0$  and the encoding of  $C_1$ . Then  $x \in L \iff R_{r(|x|)}(w_0, w_1) = 1$ . We develop a protocol for testing whether  $R_{r(|x|)}(w_0, w_1) = 1$ . The basis of the protocol is an arithmetic characterization (called *arithmetization*) of the predicate  $R$ .

Two properties of  $R$  play a crucial role here. First, for every  $k \geq 1$ , and every pair of configurations  $C$  and  $C'$  of  $D$  on input  $x$ ,

$$R_k(w, w') = 1 \iff (\exists z \in \Sigma^{s(|x|)}) [(R_{k-1}(w, z) = 1) \wedge (R_{k-1}(z, w') = 1)],$$

where  $w$  is the encoding of  $C$  and  $w'$  is the encoding of  $C'$ . Second, the predicate  $R_0(w, w')$  can be written as a polynomial of  $2s(n)$  variables (corresponding to the bits of  $w$  and  $w'$ ) having a small total degree.

**Proof of Lemma 6.11** Let  $L$  be an arbitrary language in PSPACE. Let  $D$  be a machine witnessing that  $L \in \text{PSPACE}$ . We first make a few assumptions about the machine  $D$ :

1.  $D$  has only one tape and the tape is one-way infinite.
2. The state set of  $D$  is  $Q = \{q_1, \dots, q_M\}$  and the alphabet of  $D$  is  $\Gamma = \{a_1, \dots, a_N\}$ .
3. There is a polynomial  $p$  having the following two properties:
  - For every  $x \in \Sigma^*$ ,  $D$  on input  $x$  uses at most  $p(|x|)$  tape squares.
  - For every  $n \geq 0$ ,  $p(n) \geq \max\{M, N\}$ .
4. For each  $n \geq 1$ , there is a unique accepting configuration of  $D$  for any input of length  $n$ . For example, for all input  $y \in \Sigma^*$ , we can assume that

when it is about to accept or reject  $y$ ,  $D$  writes a blank on tape squares  $1, \dots, p(|y|)$ , moves the head to the leftmost position, and then enters a unique accept state.

Let  $\Delta \subseteq (Q \times \Gamma) \times (Q \times \Gamma \times \{+1, 0, -1\})$  be the transition function of  $D$ , where  $((q, a), (q', a', d)) \in \Delta$  signifies that if the current state is  $q$  and the symbol currently scanned is  $a$ , then in one step  $D$  overwrites the currently scanned  $a$  by an  $a'$ , enters state  $q'$ , and changes the head position by  $d$  squares on the tape.

Let  $x \in \Sigma^*$  be a string whose membership in  $L$  we are testing. As in the tableau method, we encode each configuration of  $D$  on input  $x$  using a set of boolean variables. We will use the following boolean variables:

- $stt[i]$ ,  $i = 1, \dots, M$ .  
For every  $i$ ,  $1 \leq i \leq M$ ,  $stt[i] = 1$  if and only if the current state is  $q_i$ .
- $pos[i]$ ,  $i = 1, \dots, p(|x|)$ .  
For every  $i$ ,  $1 \leq i \leq p(|x|)$ ,  $pos[i] = 1$  if and only if the head is located on the  $i$ th tape square.
- $sym[i, j]$ ,  $i = 1, \dots, p(|x|)$ ,  $j = 1, \dots, N$ .  
For all  $i$ ,  $1 \leq i \leq p(|x|)$ , and  $j$ ,  $1 \leq j \leq N$ ,  $sym[i, j] = 1$  if and only if  $a_j$  is stored in the  $i$ th tape square.

Let  $s(n) = M + (N + 1)p(n)$ . Then  $s$  is a polynomial in  $n$  and the total number of variables used is  $s(|x|)$ . Fix an enumeration of the  $s(|x|)$  variables. Let  $\alpha$  be an assignment to the  $s(|x|)$  variables. We say that  $\alpha$  is *legitimate* if the following conditions hold:

- There is exactly one  $i$ ,  $1 \leq i \leq M$ , such that  $stt[i] = 1$ .
- There is exactly one  $i$ ,  $1 \leq i \leq p(|x|)$ , such that  $pos[i] = 1$ .
- For every  $i$ ,  $1 \leq i \leq p(|x|)$ , there is exactly one  $j$ ,  $1 \leq j \leq N$ , such that  $sym[i, j] = 1$ .

Then, there is a one-to-one correspondence between the set of all potential configurations of  $D$  on input  $x$  and the set of all legitimate assignments.

**Proposition 6.12** *There exists a polynomial  $R_0 \in \mathbb{Z}[\xi_1, \dots, \xi_{s(|x|)}, \theta_1, \dots, \theta_{s(|x|)}]$  that satisfies the following conditions:*

1. *An expression of  $R_0$  can be computed in polynomial time. Furthermore, for any integer  $Q \geq 2$  and  $\alpha, \beta \in (\mathbb{Z}_Q)^{s(|x|)}$ ,  $R_0(\alpha, \beta) \bmod Q$  can be evaluated in time polynomial in  $|x| + \log Q$ .*
2.  *$R_0$  is a polynomial in degree at most  $p(|x|) + 2$  in each variable.*
3. *For all  $\alpha, \beta \in \{0, 1\}^{s(|x|)}$ ,  $R_0(\alpha, \beta) = 1$  if both  $\alpha$  and  $\beta$  are legitimate and  $D$  on input  $x$  reaches  $\beta$  from  $\alpha$  in one step and  $R_0(\alpha, \beta) = 0$  otherwise.*

**Proof of Proposition 6.12** For simplicity we attach the subscripts  $\xi$  and  $\theta$  to the above  $s(|x|)$  variables to indicate that they are appearing in the  $\xi$  part and in the  $\theta$  part, respectively.

Define

$$Equal(b, c) = 2bc - b - c + 1.$$

Then, for all  $b, c \in \{0, 1\}$ ,  $Equal(b, c) = 1$  if  $b = c$  and  $Equal(b, c) = 0$  otherwise. For each integer  $k \geq 2$ , and  $k$  boolean variables  $y_1, \dots, y_k$ , define

$$Uniq[k](y_1, \dots, y_k) = \left( 1 - \prod_{1 \leq i \leq k} (1 - y_i) \right) \prod_{1 \leq i < j \leq k-1} (1 - y_i y_j).$$

Then, for every  $k \geq 2$ ,  $Uniq[k](y_1, \dots, y_k)$  is a polynomial of degree  $k$  in each variable. Also, for all  $k \geq 2$  and  $y_1, \dots, y_k \in \{0, 1\}$ ,  $Uniq[k](y_1, \dots, y_k)$  is equal to 1 if exactly one of  $y_1, \dots, y_k$  is 1 and is equal to 0 otherwise. Let  $S$  be the set of all possible transitions of  $D$ ; i.e.,

$$\begin{aligned} S = \{ (i, j, k, l, m, d) \mid & 1 \leq i \leq p(|x|) \wedge 1 \leq j \leq M \wedge 1 \leq k \leq N \wedge \\ & 1 \leq l \leq M \wedge 1 \leq m \leq N \wedge d \in \{+1, 0, -1\} \wedge 1 \leq i + d \leq p(|x|) \wedge \\ & ((q_j, a_k), (q_l, a_m, d)) \in \Delta \}. \end{aligned}$$

Define

$$R_0(\xi, \theta) = \lambda(\xi) \lambda(\theta) \sum_{\tau \in S} \rho(\xi, \theta, \tau),$$

Here

$$\begin{aligned} \lambda(\xi) = & Uniq[p(|x|)](pos_\xi[1], \dots, pos_\xi[p(|x|)]) Uniq[M](stt_\xi[1], \dots, stt_\xi[M]) \\ & \prod_{1 \leq i \leq p(|x|)} Uniq[N](sym_\xi[i, 1], \dots, sym_\xi[i, N]), \\ \lambda(\theta) = & Uniq[p(|x|)](pos_\theta[1], \dots, pos_\theta[p(|x|)]) Uniq[M](stt_\theta[1], \dots, stt_\theta[M]) \\ & \prod_{1 \leq i \leq p(|x|)} Uniq[N](sym_\theta[i, 1], \dots, sym_\theta[i, N]), \end{aligned}$$

and, for each  $\tau = (i_\tau, j_\tau, k_\tau, l_\tau, m_\tau, d_\tau) \in S$ ,

$$\begin{aligned} \rho(\xi, \theta, \tau) = & pos_\xi[i_\tau] pos_\theta[i_\tau + d_\tau] stt_\xi[j_\tau] stt_\theta[k_\tau] sym_\xi[i_\tau, l_\tau] sym_\theta[i_\tau + d_\tau, m_\tau] \\ & \prod_{t \in \{1, \dots, p(|x|)\} \setminus \{i_\tau\}} \prod_{1 \leq u \leq N} Equal(sym_\xi[t, u], sym_\theta[t, u]). \end{aligned}$$

Since  $Uniq$  checks that exactly one of the input variables is 1,  $\lambda(\xi) = 1$  if and only if  $\xi$  is legitimate. Similarly,  $\lambda(\theta) = 1$  if and only if  $\theta$  is legitimate. Also, for all 6-tuple  $\tau = (i, j, k, l, m, d) \in S$ , and  $\alpha, \beta \in \{0, 1\}^{s(|x|)}$ ,  $\lambda(\alpha) \lambda(\beta) \rho(\alpha, \beta, \tau) = 1$  if

(i) both  $\alpha$  and  $\beta$  are legitimate and

- (ii) the changes corresponding to the transition  $((q_j, a_k), (q_l, a_m, d))$  are made to the variables  $pos_i, pos_{i+d}, stt_j, stt_l, sym_{i,k}$ , and  $sym_{i,m}$ , and for other places  $\alpha$  and  $\beta$  are equal to each other;

otherwise,  $\lambda(\alpha)\lambda(\beta)\rho(\alpha, \beta, \tau) = 0$ . Thus, for all  $\alpha, \beta \in \{0, 1\}^{s(|x|)}$ ,  $\lambda(\alpha)\lambda(\beta)\rho(\alpha, \beta, \tau) = 1$  if both  $\alpha$  and  $\beta$  are legitimate and  $\beta$  is the configuration that results from  $\alpha$  in one step of  $D$  and  $\lambda(\alpha)\lambda(\beta)\rho(\alpha, \beta, \tau) = 0$  otherwise. For each variable  $y$  that is present in  $\lambda$ , the degree of  $y$  in  $\lambda$  is at most  $p(|x|)$  if  $y$  is a *pos* variable, at most  $M$  if it is a *stt* variable, and at most  $N$  if it is a *sym* variable. On the other hand, the degree of each variable in  $\rho$  is at most 1. So, the degree of each variable in  $R_0$  is at most  $\max\{p(|x|), M, N\} + 1$ . Since for every  $n \geq 0$ ,  $p(n) \geq \max\{M, N\}$ , the degree of each variable in  $R_0$  is at most  $p(|x|) + 1$ . The set  $S$  has only polynomially many elements. Thus, the expression for  $R_0$  can be computed in time polynomial in  $|x|$ . Now given  $Q \geq 2$  and  $\alpha, \beta \in (\mathbb{Z}_Q)^{s(|x|)}$ ,  $R_0(\alpha, \beta) \bmod Q$  can be evaluated in time polynomial in  $|x| + \log Q$ . This proves the proposition.  $\square$

Proposition 6.12

For each  $k \in \{1, \dots, r(|x|)\}$ , define the polynomial  $R_k(\xi, \theta)$  to be

$$\sum_{\gamma_1 \in \{0,1\}} \sum_{\gamma_{s(|x|)} \in \{0,1\}} R_{k-1}(\xi, \gamma_1, \dots, \gamma_{s(|x|)}) R_{k-1}(\gamma_1, \dots, \gamma_{s(|x|)}, \theta).$$

Then it is easy to see that the following proposition holds.

**Proposition 6.13**

1. For every  $k \in \{0, \dots, r(|x|)\}$ ,  $R_k$  has degree at most  $p(|x|) + 1$  in each variable.
2. For all  $k$ ,  $0 \leq k \leq r(|x|)$ , and  $\alpha, \beta \in \{0, 1\}^{s(|x|)}$ ,  $R_k(\alpha, \beta) = 1$  if both  $\alpha$  and  $\beta$  are legitimate and  $\beta$  is reachable from  $\alpha$  by  $D$  on input  $x$  in exactly  $2^k$  steps and  $R_k(\alpha, \beta) = 0$  otherwise.

Now, let  $C_{ini} \in \{0, 1\}^{s(|x|)}$  be the initial configuration of  $D$  on input  $x$  and  $C_{fin} \in \{0, 1\}^{s(|x|)}$  be the unique accepting configuration of  $D$  on any input of length  $|x|$ . Then

$$x \in L \iff R_{r(|x|)}(C_{ini}, C_{fin}) = 1.$$

We develop an interactive protocol for verifying that  $R_{r(|x|)}(C_{ini}, C_{fin}) = 1$ . To explain the protocol we need to define some notation. For each  $k \in \{1, \dots, r(|x|)\}$ ,  $l \in \{1, \dots, s(|x|)\}$ ,  $\alpha, \beta \in \mathbb{Z}^{s(|x|)}$ , and  $\gamma = (\gamma_1, \dots, \gamma_{l-1}) \in \mathbb{Z}^{l-1}$ , define

$$\begin{aligned} G[k, l, \alpha, \beta, \gamma](y) &= \sum_{e_{l+1} \in \{0,1\}} \sum_{e_{s(|x|)} \in \{0,1\}} \\ &R_{k-1}(\alpha, (\gamma_1, \dots, \gamma_{l-1}, y, e_{l+1}, \dots, e_{s(|x|)})) \\ &R_{k-1}((\gamma_1, \dots, \gamma_{l-1}, y, e_{l+1}, \dots, e_{s(|x|)}), \beta), \end{aligned}$$

and for each  $k \in \{1, \dots, r(|x|)\}$ ,  $l \in \{0, \dots, s(|x|)\}$ ,  $\alpha, \beta \in \mathbb{Z}^{s(|x|)}$ , and  $\gamma = (\gamma_1, \dots, \gamma_l) \in \mathbb{Z}^l$ , define

$$G'[k, l, \alpha, \beta, \gamma] = \sum_{e_{l+1} \in \{0,1\}} \sum_{e_{s(|x|)} \in \{0,1\}} R_{k-1}(\alpha, (\gamma_1, \dots, \gamma_l, e_{l+1}, \dots, e_{s(|x|)})) \\ R_{k-1}((\gamma_1, \dots, \gamma_l, e_{l+1}, \dots, e_{s(|x|)}), \beta).$$

Then, by Proposition 6.13, we have the following result.

**Proposition 6.14**

1. For all  $k \in \{1, \dots, r(|x|)\}$ ,  $l \in \{1, \dots, s(|x|)\}$ ,  $\alpha, \beta \in \mathbb{Z}^{s(|x|)}$ , and  $\gamma \in \mathbb{Z}^{l-1}$ ,  $G[k, l, \alpha, \beta, \gamma](y)$  is a polynomial in  $y$  of degree at most  $2p(|x|) + 2$ .
2. For all  $k \in \{1, \dots, r(|x|)\}$ ,  $l \in \{0, \dots, s(|x|) - 1\}$ ,  $\alpha, \beta \in \mathbb{Z}^{s(|x|)}$ , and  $\gamma \in \mathbb{Z}^l$ ,  $G'[k, l, \alpha, \beta, \gamma] = G[k, l + 1, \alpha, \beta, \gamma](0) + G[k, l + 1, \alpha, \beta, \gamma](1)$ .
3. For all  $k \in \{1, \dots, r(|x|)\}$ ,  $l \in \{1, \dots, s(|x|)\}$ ,  $\alpha, \beta \in \mathbb{Z}^{s(|x|)}$ ,  $\gamma = (\gamma_1, \dots, \gamma_{l-1}) \in \mathbb{Z}^{l-1}$ , and  $\gamma_l \in \mathbb{Z}$ ,  $G'[k, l, \alpha, \beta, \gamma'] = G[k, l, \alpha, \beta, \gamma](\gamma_l)$ , where  $\gamma' = (\gamma_1, \dots, \gamma_l)$ .
4. For all  $k \in \{1, \dots, r(|x|)\}$  and  $\alpha, \beta \in \mathbb{Z}^{s(|x|)}$ ,  $G'[k, 0, \alpha, \beta, \epsilon] = R_k(\alpha, \beta)$ , where  $\epsilon$  denotes the empty string.
5. For every  $k \in \{1, \dots, r(|x|)\}$  and  $\alpha, \beta, \gamma \in \mathbb{Z}^{s(|x|)}$ ,  $G'[k, s(|x|), \alpha, \beta, \gamma] = R_{k-1}(\alpha, \gamma)R_{k-1}(\gamma, \beta)$ .

For each  $k$ ,  $1 \leq k \leq r(|x|)$ , and  $\alpha, \beta, \gamma \in \mathbb{Z}^{s(|x|)}$ , define

$$H[k, \alpha, \beta, \gamma](y) = R_{k-1}((\gamma - \alpha)y + \alpha, (\beta - \gamma)y + \gamma).$$

Here  $(\gamma - \alpha)y + \alpha$  is shorthand for

$$((\gamma_1 - \alpha_1)y + \alpha_1, \dots, (\gamma_{s(|x|)} - \alpha_{s(|x|)})y + \alpha_{s(|x|)})$$

and  $(\beta - \gamma)y + \gamma$  is shorthand for

$$((\beta_1 - \gamma_1)y + \gamma_1, \dots, (\beta_{s(|x|)} - \gamma_{s(|x|)})y + \gamma_{s(|x|)}),$$

where for each  $i$ ,  $1 \leq i \leq s(|x|)$ ,  $\alpha_i$ ,  $\beta_i$ , and  $\gamma_i$  are respectively the  $i$ th component of  $\alpha$ , the  $i$ th component of  $\beta$ , and the  $i$ th component of  $\gamma$ . Then we have the following result.

**Proposition 6.15** For all  $k \in \{1, \dots, r(|x|)\}$ , and  $\alpha, \beta, \gamma \in \mathbb{Z}^{s(|x|)}$ ,

1.  $H[k, \alpha, \beta, \gamma](y)$  is a polynomial in  $y$  of degree at most  $p(|x|) + 2$ ,
2.  $H[k, \alpha, \beta, \gamma](0) = R_{k-1}(\alpha, \gamma)$ , and
3.  $H[k, \alpha, \beta, \gamma](1) = R_{k-1}(\gamma, \beta)$ .



This routine takes as input two integers  $M \geq 2$  and  $t \geq 1$  and outputs a number in  $\{0, \dots, M-1\}$  under uniform distribution with a sampling error occurring with probability less than or equal to  $2^{-t}$ . Let  $\ell$  be the smallest integer such that  $2^\ell \geq M$ . Repeat the following at most  $t$  times.

(\*) Use  $\ell$  fair coin tosses to select an integer  $Y$  between 0 and  $2^\ell - 1$ . If  $Y \leq M$ , then quit the loop and return  $Y$ .

If none of the trials are successful, then accept  $x$ .

**Fig. 6.4** The sampling algorithm

Define the polynomial  $m$  by  $m(n) = r(n)(2p(n) + 2)(s(n) + 1) + 3$ . The protocol uses the following sampling algorithm that, on input  $M$  and  $t$ , outputs an integer between 0 and  $M - 1$  uniformly at random, where a sampling error occurs with probability less than or equal to  $2^{-t}$  (the equality holds if and only if  $M$  is a power of two).

It is not hard to see that the sampling algorithm works as desired. The number  $\ell$  satisfies  $2^{\ell-1} < M \leq 2^\ell$ . So, in a single execution of the loop body, the probability that the number  $Y$  is greater than or equal to  $M$  is 0 if  $M$  is a power of two and less than  $\frac{1}{2}$  otherwise. Since at most  $t$  rounds will be executed to find an appropriate  $Y$ , the error probability is precisely 0 if  $M$  is a power of two and less than  $2^{-t}$  otherwise. For each execution of the loop body, and for each  $i$ ,  $0 \leq i \leq M - 1$ , the chances that the selection  $Y$  is equal to  $i$  is precisely  $2^{-\ell}$ . So, the resulting distribution is uniform.

We can assume that the prover has a binary encoding of an integer  $Q \in [2^{m(|x|)}, 2^{2m(|x|)}]$  which is supposedly a prime number and a certificate of its primality that can be verified in polynomial time. This assumption is valid. By the Prime Number Theorem (Theorem 6.8), such a prime exists and the following theorem, which we state without a proof, shows that every prime has a polynomial-time verifiable certificate having polynomial length.

**Theorem 6.16** *The set of all prime numbers written in binary belongs to NP.*

Now we are ready to present the protocol in Fig. 6.5.

We claim that this protocol witnesses that  $L \in \text{IP}$ . By definition  $m$ ,  $r$ , and  $s$  are all polynomials. So, the entire computation requires polynomially many steps. The probability that the sampling algorithm fails is less than  $2^{-m(|x|)}$  since the parameter  $t$  is set to  $m(|x|)$ . The total number of samples generated is  $r(|x|)(s(|x|) + 1)$ . Since  $m(|x|) = r(|x|)(2p(|x|) + 2)(s(|x|) + 1) + 3$ , the probability that a sampling error occurs during the execution of the entire protocol, regardless of whether  $x \in L$  or not, is less than  $\frac{1}{8}$ .

**Claim 6.17** *The protocol is complete.*

**Step 1** Obtain from the prover a prime number  $Q$  in the interval  $[2^{m(|x|)}, 2^{2m(|x|)}]$  and a short certificate of its primality. Set  $v_{r(|x|),0} = 1$ ,  $\alpha = C_{ini}$ , and  $\beta = C_{fin}$ .

**Step 2** Repeat the following for  $k = r(|x|), \dots, 1$ :

- (a) Repeat the following for  $l = 1, \dots, s(|x|)$ :
  - (i) Obtain from the oracle a polynomial  $g \in \mathbb{Z}_Q[y]$  of degree at most  $2p(|x|) + 2$ , which the oracle claims is  $G[k, l, \alpha, \beta, \gamma_1 \dots \gamma_{l-1}](y)$ .
  - (ii) Test whether  $v_{k,l-1} \equiv g(0) + g(1) \pmod{Q}$ . If the test fails, immediately reject  $x$ . Otherwise, proceed to the part (iii) of Step 2(a).
  - (iii) Randomly sample a number  $\gamma_l \in \mathbb{Z}_Q$  by running the Sampling Algorithm in Fig. 6.4 with  $M = Q$  and  $t = m(|x|)$ . Set  $v_{k,l} = g(\gamma_l) \pmod{Q}$ .
- (b) Let  $\gamma = \gamma_1 \dots \gamma_{s(|x|)}$ . Obtain from the oracle a polynomial  $h \in \mathbb{Z}_Q[y]$  of degree at most  $p(|x|) + 1$ , which the oracle claims is  $H[k, \alpha, \beta, \gamma](y) \pmod{Q}$ . Test whether  $v_{k,s(|x|)} \equiv h(0)h(1) \pmod{Q}$ . If the test fails, immediately reject  $x$ .
- (c) Randomly sample a number  $\rho \in \mathbb{Z}_Q$  by running the Sampling Algorithm in Fig. 6.4 with  $M = Q$  and  $t = m(|x|)$ . Set  $v_{k-1,0} = h(\rho) \pmod{Q}$ . Set  $\alpha$  to  $(\gamma - \alpha)\rho + \alpha \pmod{Q}$  and  $\beta$  to  $(\beta - \gamma)\rho + \gamma \pmod{Q}$ .

**Step 3** Test whether  $v_{0,0} \equiv R_0(\alpha, \beta) \pmod{Q}$ . If the test succeeds, accept  $x$ . Otherwise, reject  $x$ .

**Fig. 6.5** Interactive protocol for PSPACE

**Proof of Claim 6.17** Suppose that  $x \in L$ . Then  $v_{r(|x|),0} = G'[r(|x|), 0, C_{ini}, C_{fin}, \epsilon] = 1$ . By Proposition 6.14, for all  $k \in \{1, \dots, r(|x|)\}$ ,  $l \in \{0, \dots, s(|x|) - 1\}$ ,  $\alpha, \beta \in (\mathbb{Z}_Q)^{s(|x|)}$ , and  $\gamma = (\gamma_1, \dots, \gamma_l) \in (\mathbb{Z}_Q)^l$ , if  $v_{k,l} \equiv G'[k, l, \alpha, \beta, \gamma] \pmod{Q}$ , the prover can provide a polynomial  $g \in \mathbb{Z}_Q[y]$  such that  $v_{k,l} \equiv g(0) + g(1) \pmod{Q}$  and such that, for all  $\gamma_{l+1} \in \mathbb{Z}_Q$ ,  $g(\gamma_{l+1}) \equiv G'[k, l+1, \alpha, \beta, \gamma'] \pmod{Q}$ , where  $\gamma' = (\gamma_1, \dots, \gamma_{l+1})$ .

Furthermore, by Proposition 6.15, for all  $k \in \{1, \dots, r(|x|)\}$ , and  $\alpha, \beta, \gamma \in (\mathbb{Z}_Q)^{s(|x|)}$ , if  $v_{k,s(|x|)} \equiv G'[k, s(|x|), \alpha, \beta, \gamma] \pmod{Q}$ , then the prover can provide a polynomial  $h \in \mathbb{Z}_Q[y]$  such that  $v_{k,s(|x|)} \equiv h(0)h(1) \pmod{Q}$  and such that, for all  $r \in \mathbb{Z}_Q$ ,  $h(r) \equiv G'[k-1, 0, \alpha', \beta', \epsilon] \pmod{Q}$ , where  $\alpha' = (\gamma - \alpha)r + \alpha \pmod{Q}$  and  $\beta' = (\beta - \gamma)r + \gamma \pmod{Q}$ .

By the above observations, there is a prover  $\tilde{P}$  such that the verifier accepts  $x$  with probability one through interactions with  $\tilde{P}$  assuming that a sampling error never occurs. Since the verifier accepts when a sampling error occurs, the probability that the verifier accepts through interactions with  $\tilde{P}$  is 1. Thus, the protocol is complete.  $\square$  Claim 6.17

**Claim 6.18** *The protocol is sound.*

**Proof of Claim 6.18** Suppose that  $x \notin L$ . Then  $G'[r(|x|), 0, C_{ini}, C_{fin}, \epsilon] = R_{r(|x|)}(C_{ini}, C_{fin}) = 0$ . So,  $v_{r(|x|),0} \neq G'[r(|x|), 0, C_{ini}, C_{fin}, \epsilon]$ . By Proposition 6.14, for all  $k \in \{1, \dots, r(|x|)\}$ ,  $l \in \{0, \dots, s(|x|) - 1\}$ ,

$\alpha, \beta \in (\mathbb{Z}_Q)^{s(|x|)}$ ,  $\gamma = (\gamma_1, \dots, \gamma_l) \in (\mathbb{Z}_Q)^l$ , and all polynomials  $g(y) \in \mathbb{Z}[y]$  of degree at most  $2p(|x|) + 2$ , if  $v_{k,l} \not\equiv G'[k, l, \alpha, \beta, \gamma] \pmod{Q}$  and  $v_{k,l} \equiv g(0) + g(1) \pmod{Q}$ , then  $g(y) \not\equiv G[k, l, \alpha, \beta, \gamma](y) \pmod{Q}$ , and thus, by Lemma 6.2, there are at most  $2p(|x|) + 2$  values of  $\gamma_{l+1}$  such that  $v_{k,l+1} \equiv G'[k, l, \alpha, \beta, \gamma'] \pmod{Q}$ , where  $\gamma' = (\gamma_1, \dots, \gamma_{l+1})$ . This implies that for all  $k \in \{1, \dots, r(|x|)\}$ ,  $l \in \{0, \dots, s(|x|) - 1\}$ ,  $\alpha, \beta \in (\mathbb{Z}_Q)^{s(|x|)}$ ,  $\gamma = (\gamma_1, \dots, \gamma_l) \in (\mathbb{Z}_Q)^l$ , and all polynomials  $g(y) \in \mathbb{Z}_Q[y]$  of degree at most  $2p(|x|) + 2$ , if  $v_{k,l} \not\equiv G'[k, l, \alpha, \beta, \gamma] \pmod{Q}$  and  $g$  passes the part (ii) of Step 2(a), then, for  $\gamma_{l+1}$  chosen uniformly at random in  $\mathbb{Z}_Q$ , with probability at most  $\frac{2p(|x|)+4}{Q}$ ,  $v_{k,l+1} \equiv G'[k, l, \alpha, \beta, \gamma'] \pmod{Q}$ , where  $\gamma' = (\gamma_1, \dots, \gamma_{l+1})$ .

Furthermore, by Proposition 6.15, for all  $k \in \{1, \dots, r(|x|)\}$ ,  $\alpha, \beta, \gamma \in (\mathbb{Z}_Q)^{s(|x|)}$ , and all polynomials  $h \in \mathbb{Z}_Q[y]$  of degree at most  $p(|x|) + 1$ , if  $v_{k,s(|x|)} \not\equiv G'[k, s(|x|), \alpha, \beta, \gamma] \pmod{Q}$  and  $v_{k,s(|x|)} \equiv h(0)h(1) \pmod{Q}$ , then  $h \not\equiv H[k, \alpha, \beta, \gamma]$ . Thus, for all  $k \in \{1, \dots, r(|x|)\}$ ,  $\alpha, \beta, \gamma \in (\mathbb{Z}_Q)^{s(|x|)}$ , and all polynomials  $h \in \mathbb{Z}_Q[y]$  of degree at most  $p(|x|) + 1$ , if  $v_{k,s(|x|)} \not\equiv G'[k, s(|x|), \alpha, \beta, \gamma] \pmod{Q}$  and  $h$  passes the test in Step 2(b), then, for  $\rho$  chosen uniformly at random in  $\mathbb{Z}_Q$ , with probability at most  $\frac{p(|x|)+2}{Q}$ ,  $v_{k-1,0} \equiv G'[k-1, 0, \alpha', \beta', \epsilon] \pmod{Q}$ , where  $\alpha' = (\gamma - \alpha)\rho + \alpha \pmod{Q}$  and  $\beta' = (\beta - \gamma)\rho + \gamma \pmod{Q}$ .

Finally, if  $v_{0,0} \not\equiv G'[0, 0, \alpha, \beta, \epsilon] \pmod{Q}$ , then the verifier deterministically rejects  $x$  in Step 3.

By the above observations, the probability that the verifier accepts  $x$  is at most  $\frac{(2p(|x|)+2)r(|x|)(s(|x|)+1)}{Q}$ . Since  $m(n) = r(n)(2p(n) + 2)(s(n) + 1) + 3$  and  $Q \geq 2^{m(|x|)}$ , this is at most  $\frac{1}{8}$ . Thus, the probability that the verifier accepts  $x$  is at most  $\frac{1}{8} + \frac{1}{8} = \frac{1}{4}$ . Hence, the protocol is sound.

□ Claim 6.18

□ Lemma 6.11

## 6.4 MIP = NEXP

In this section we study the power of multiprover interactive proof systems. We stipulate that the provers here do not talk among themselves; otherwise, one prover could simulate all the other provers, and thus, the computational power of the system is the same as that of PSPACE. We will show that there is a big jump in the computational power when one extra prover is added to the system. Namely, we will show in this section that the *two-prover* interactive proof systems recognize precisely those languages in nondeterministic exponential time. We also show that with more than two provers the verifier can recognize all languages in nondeterministic exponential time.

**Theorem 6.19** MIP = NEXP.

### 6.4.1 Probabilistic Oracle Protocols and $\text{MIP} \subseteq \text{NEXP}$

In order to prove Theorem 6.19 we introduce the concept of probabilistic oracle protocols.

**Definition 6.20** *A language  $L$  has a probabilistic oracle protocol if there exists a probabilistic polynomial time-bounded oracle Turing machine  $M$  such that, for every  $x \in \Sigma^*$ , the following conditions hold:*

1. **(Completeness)** *If  $x \in L$ , then there exists some oracle  $H$  such that  $M$  on input  $x$  relative to  $H$  accepts with probability greater than  $\frac{3}{4}$ .*
2. **(Soundness)** *If  $x \notin L$ , then for every oracle  $H$   $M$  on input  $x$  relative to  $H$  accepts with probability less than  $\frac{1}{4}$ .*

The difference between the above definition and Definition 6.1 is that here only one prover is involved and the unique prover behaves as an oracle.

**Theorem 6.21** *For every language  $L$ ,  $L$  is in  $\text{MIP}$  if and only if  $L$  has a probabilistic oracle protocol.*

**Proof** For the “only-if” part, suppose that  $L$  is a language in  $\text{MIP}$ . Take a verifier  $V$  witnessing  $L \in \text{MIP}$ . Let  $k \geq 1$  be the number of provers that  $V$  communicates with. Without loss of generality, we may assume that provers provide single-bit answers. By following an argument similar to the one we had in the proof of Lemma 6.10 on page 123, we can assume that these provers are deterministic.

We modify the queries of  $V$ . Let  $x$  be an input to the system. Suppose that  $V$  is about to make a query, say  $y$ . Let  $i$ ,  $1 \leq i \leq k$ , and  $j \geq 1$  be such that the query  $y$  that  $V$  is about to make is to a query to  $P_i$  and  $V$  on input  $x$  has made  $j-1$  queries to  $P_i$  so far. Then we replace this query by  $\langle x, i, j, y, W \rangle$ , where  $x$  is the input to the system and  $W$  is the history of communication between  $V$  and  $P_i$ . More precisely,  $W = y_1 \# b_1 y_2 \# b_2 \dots y_{j-1} \# b_{j-1}$ , where for each  $\ell$ ,  $1 \leq \ell \leq j-1$ ,  $y_\ell$  is the fourth component (the  $y$  part) of the  $\ell$ th query of  $V$  to  $P_i$  and  $b_\ell$  is the answer that  $P_i$  provided to that query. This modification makes the queries of  $V$  unique, in the sense that no queries are made twice. This modification does not change the probability of acceptance since the additional four pieces of information, i.e.,  $x$ ,  $i$ ,  $j$ , and  $W$ , are already known to the prover.

As the provers are deterministic and the queries are unique, we can turn  $V$  into an oracle Turing machine  $N$  by replacing the provers with a single oracle. Then for every  $x \in \Sigma^*$ , the largest probability that  $N$  on input  $x$  accepts with any oracle is equal to the largest probability that  $V$  on input  $x$  accepts with any set of  $k$  provers. So, the two conditions in Definition 6.20 hold. Thus,  $L$  has a probabilistic oracle protocol. This proves the “only-if” part.

For the “if” part, suppose that there is a polynomial time-bounded probabilistic oracle Turing machine  $N$  witnessing that  $L$  has a probabilistic oracle

**Step 1**  $V$  executes the following  $10p(|x|)$  times:

- (a)  $V$  simulates  $N$  on input  $x$  using  $P_1$  as the oracle.
- (b)  $V$  calls the Sampling Algorithm (in Fig. 6.4) with  $M = p(|x|)$  and  $t = p(|x|)$  and uses it to select one query  $y_i$  that is made during this round of simulation and asks  $y_i$  to  $P_2$ .

**Step 2**  $V$  accepts  $x$  if for all  $i$ ,  $1 \leq i \leq 10p(|x|)$ , the answer of  $P_2$  to  $y_i$  is equal to that of  $P_1$  and the number of simulations of  $N$  on input  $x$  that accepted is at least  $5p(|x|)$ . Otherwise,  $V$  rejects  $x$ .

**Fig. 6.6** The two-prover protocol

protocol. We may assume that there is a polynomial  $p$  such that, for every  $x \in \Sigma^*$ ,  $N$  on input  $x$  always makes exactly  $p(|x|)$  queries. We may also assume that the oracle provides single-bit answers. Let  $V$  be a verifier with two provers,  $P_1$  and  $P_2$ , that, on input  $x$ , executes the algorithm in Fig. 6.6

**6.4.1.1 Completeness of the Protocol.** We claim that this is a complete two-prover interactive proof system for  $L$ . Obviously,  $V$  is polynomial time-bounded. We need to show that the protocol is complete, and sound. We first show that the protocol is complete. Let  $x$  be any member of  $L$ . The following lemma, called Chebyshev's Inequality, which we state without a proof, is well known and useful for our analysis.

**Lemma 6.22 (Chebyshev's Inequality)** *Let  $X$  be a random variable with expectation  $\alpha$  and variance  $\sigma$ . Then for all  $\delta > 0$*

$$\Pr[|X - \alpha| \geq \delta] \leq \frac{\sigma}{\delta^2}.$$

Since  $x \in L$ , there exists an oracle  $H$  relative to which  $N$  on input  $x$  accepts with probability  $\rho > \frac{3}{4}$ . Suppose that both  $P_1$  and  $P_2$  provide answers as if they were  $H$ . Then the consistency tests between  $P_1$  and  $P_2$  all succeed. So, the probability that  $V$  accepts is equal to the probability that more than half of the simulations in Step 1 accept. Since the coin flips of  $V$  are independent, the expectation  $\alpha$  of the number of accepting computations that  $V$  finds is  $\rho(10p) > \frac{15p}{2}$  and the variance  $\sigma$  of this number is  $\rho(1-\rho)(10p) < \frac{15p}{8}$ , where  $p$  is shorthand for  $p(|x|)$ . Then, by Lemma 6.22, the probability that the number of accepting paths is less than or equal to  $5p$  is less than or equal to

$$\frac{\sigma}{|5p - \alpha|^2} < \frac{\frac{15p}{8}}{|5p - \frac{15p}{2}|^2} \leq \frac{3}{10p}.$$

This quantity is less than  $\frac{3}{20} < \frac{1}{4}$  for  $p(n) \geq 2$ . As we can make the polynomial  $p$  arbitrarily large, the probability of acceptance is greater than  $\frac{3}{4}$  for  $x$ . Hence, the protocol is complete.

**6.4.1.2 Soundness of the Protocol.** Next we prove that the protocol is sound. Let  $x$  be an arbitrary member of  $\bar{L}$ . Let  $n = |x|$  and  $m = 10p(n)$ . We need to show that, for every pair of provers  $(P_1, P_2)$ , the probability that  $V$  on input  $x$  accepts through interactions with  $P_1$  and  $P_2$  is strictly less than  $\frac{1}{4}$ . The number of times that the sampling algorithm is executed is  $10p(|x|)$ . The probability that the sampling algorithm fails is less than  $2^{-p(|x|)}$ . So, the probability that a sampling error occurs during the execution of the entire protocol is  $\frac{10p(|x|)}{2^{p(|x|)}}$ . This is at most  $\frac{1}{10}$  for  $p(n) \geq 10$ .

We claim that a pair of deterministic provers  $(P_1, P_2)$  can achieve the highest acceptance probability. To see why, note that we can assume that the goal of  $(P_1, P_2)$  is to maximize the acceptance probability of  $V$  on input  $x$  through interactions with them. Also, note that an additional role of  $P_2$  is correctly to guess the answer that  $P_1$  provided. Upon receiving a query,  $P_2$  can use its unlimited computational power to calculate the probability that  $P_1$  provided a 0 as an answer. Then  $P_2$  can maximize the probability by answering with a 0 if the calculated probability is greater than or equal to  $\frac{1}{2}$  and with a 1 otherwise. So, the strategy of  $P_2$  can be deterministic. On the other hand, suppose that the very last query of  $V$  on input  $x$  is given to  $P_1$ . Having unlimited computational power and knowing the protocol of  $P_2$ ,  $P_1$  can calculate the probability of acceptance of  $V$  on input  $x$  in the case when it answers with a 0 and the probability of acceptance of  $V$  on input  $x$  when it answers with a 1. Call these probabilities,  $\alpha_0$  and  $\alpha_1$ , respectively. To maximize the acceptance probability of  $V$  on input  $x$ ,  $P_1$  can deterministically select its answer as follows: answer with a 0 if  $\alpha_0 \geq \alpha_1$  and with a 1 otherwise. Thus,  $P_1$  can answer the very last query deterministically without decreasing the probability of acceptance of  $V$ . By repeating this argument for the penultimate query, we can argue that  $P_1$  can answer that query deterministically without decreasing the acceptance probability of  $V$ . By repeating this argument over and over again, we can argue that  $P_2$  can answer every query deterministically, without making it less likely that  $V$  will accept.

Since  $P_2$  is deterministic and  $V$  makes only one query to  $P_2$  at each round, for every  $i$ ,  $1 \leq i \leq m$ , the function of  $P_2$  at round  $i$  can be viewed as that of an oracle. We can assume that at the beginning of each round,  $P_1$  decides its strategy for that round. This selection determines the probability that  $P_1$  provides an answer that is different from that which  $P_2$  would provide. So, the overall strategy of  $P_1$  can be parameterized using  $m$  real numbers in the interval  $[0, 1]$ . For each  $\alpha = (\alpha_1, \dots, \alpha_m) \in [0, 1]^m$ , we say that  $P_1$  takes an  $\alpha$ -strategy if for all  $i$ ,  $1 \leq i \leq m$ , the strategy of  $P_1$  in round  $i$  is to provide an answer different from what  $P_2$  would provide with probability  $\alpha_i$ . We will show below that there exists some  $n_0 > 0$  such that, for all  $n \geq n_0$ , and for all  $\alpha \in [0, 1]^m$ , the probability that  $V$  accepts  $x$  in the case when  $P_1$  takes an  $\alpha$ -strategy is strictly less than  $\frac{1}{4}$ .

Let  $\alpha_1, \dots, \alpha_m \in [0, 1]^m$ . Assume that  $P_1$  takes an  $\alpha$ -strategy. Since  $x \notin L$ , for every oracle  $Q$ , the probability that  $N$  accepts  $x$  with  $Q$  as the oracle is less than  $\frac{1}{4}$ . Thus, for every  $i$ ,  $1 \leq i \leq m$ , the probability that the simulation of  $N$  on input  $x$  in round  $i$  through interactions with  $P_1$  is accepting is strictly less than

$$\min \left\{ 1, \frac{1}{4} + \alpha_i \right\} \leq \frac{1}{4} + \alpha_i.$$

Thus, the expected number of accepting simulations that are generated is less than  $\frac{m}{4} + \sum_{1 \leq i \leq m} \alpha_i$ . Also, the variance of the number of accepting simulations that are generated is less than  $\sum_{1 \leq i \leq m} (\frac{1}{4} + \alpha_i)(\frac{3}{4} - \alpha_i) \leq \frac{m}{4} + \sum_{1 \leq i \leq m} \alpha_i$ . Now we claim that under the assumption (\*), the probability that  $V$  accepts  $x$  is at most  $\frac{1}{4}$ . To prove the claim, first suppose that  $\sum_{1 \leq i \leq m} \alpha_i \leq \frac{9m}{40}$ . Then, by Chebyshev's Inequality (Lemma 6.22), the probability that the number of accepting simulations that are generated is at least  $\frac{m}{2}$  is less than

$$\frac{\frac{m}{4} + \frac{9m}{40}}{\left(\frac{m}{2} - \frac{m}{4} - \frac{9m}{40}\right)^2} = \frac{7600}{m} = \frac{760}{p}.$$

This is less than  $\frac{1}{8}$  for  $p(n) \geq 6081$ . Thus, the probability that  $V$  accepts  $x$  when there is no sampling error is less than  $\frac{1}{8}$ .

Next suppose that  $\sum_{1 \leq i \leq m} \alpha_i > \frac{9m}{40}$ . Then, under the assumption (\*), the expected number of simulations in which  $P_1$  disagrees with  $P_2$  on at least one query is  $\frac{9m}{40}$  while the variance of the number is less than  $\frac{9m}{40}$ . Then, by Chebyshev's Inequality (Lemma 6.22), the probability that there are no more than  $\frac{m}{5} = 2p(|x|)$  simulation rounds in which  $P_1$  disagrees with  $P_2$  is less than

$$\frac{\frac{9m}{40}}{\left(\frac{9m}{40} - \frac{m}{5}\right)^2} = \frac{3600}{m} = \frac{360}{p(|x|)}.$$

This is less than  $\frac{1}{16}$  for  $p(n) \geq 5761$ . Furthermore, if there are  $2p(|x|)$  simulation rounds in which  $P_1$  disagrees with  $P_2$ , then the probability that the disagreement is not discovered is at most

$$\left(1 - \frac{1}{p(|x|)}\right)^{2p(|x|)}$$

This is less than  $\frac{1}{16}$  for  $p(n) \geq 5761$ . Thus, for all  $n$  sufficiently large, the probability that  $V$  accepts  $x$  when there is no sampling error is less than  $\frac{1}{8}$ . Since a sampling error makes  $V$  accept and occurs with probability less than  $\frac{1}{8}$ , the probability that  $V$  accepts  $x$  is less than  $\frac{1}{4}$ . Hence, the protocol is sound. This proves the theorem.  $\square$  Theorem 6.21

Now we turn to the proof of Theorem 6.19. We first show that every language in MIP is indeed in NEXP.

**Theorem 6.23**  $\text{MIP} \subseteq \text{NEXP}$ .

**Proof** Let  $L \in \text{MIP}$ . By Theorem 6.21, there is a polynomial time-bounded probabilistic oracle Turing machine  $M$  satisfying the completeness and the soundness conditions in Definition 6.20. Then, for every  $x \in \Sigma^*$ ,  $x \in L$  if and only if there is an oracle relative to which  $M$  on input  $x$  accepts with probability strictly greater than  $\frac{3}{4}$ . Let  $p$  be a polynomial bounding the runtime of  $M$ . Then for every  $x \in \Sigma^*$  the queries of  $M$  on input  $x$  are of length at most  $p(|x|)$ . Then, for every  $x \in \Sigma^*$ ,  $x \in L$  if and only if there is some  $B \subseteq (\Sigma^*)^{\leq p(|x|)}$  such that  $M$  on  $x$  relative to  $B$  accepts with probability greater than  $\frac{3}{4}$ . Without loss of generality, we may assume that at each computational step,  $M$  has at most two possible moves and that all possible moves that  $M$  can make are linearly ordered. Define  $N$  to be the nondeterministic Turing machine that, on input  $x$ , executes the following:

- For each  $y \in \Sigma^*$  having length at most  $p(|x|)$   $N$  guesses a bit  $b(y)$ .  $N$  sets  $B$  to the set of all strings  $y \in \Sigma^*$  having length at most  $p(|x|)$  such that  $b(y) = 1$ .
- $N$  sets count  $S$  to 0. Then, for each  $w \in \Sigma^{p(|x|)}$ ,  $N$  does the following:
  - $N$  deterministically simulates  $M$  on input  $x$  with oracle  $B$  along path  $w$  as follows: For each  $i$ ,  $1 \leq i \leq p(|x|)$ , if there are two possible moves that  $M$  on input  $x$  with oracle  $B$  can make at step  $i$  in the current simulation, then  $N$  picks the move with the lower order if  $w_i = 0$  and the one with the higher order if  $w_i = 1$ , where  $w_i$  is the  $i$ th symbol of  $w$ .
  - If  $M$  on input  $x$  with oracle  $B$  accepts along path  $w$ , then  $N$  increments  $S$  by 1.
- $N$  accepts  $x$  if  $S/2^{p(|x|)} > \frac{3}{4}$  and rejects otherwise.

It is easy to see that for every  $x \in \Sigma^*$ ,  $N$  on input  $x$  accepts if and only if there is some  $B \subseteq (\Sigma^*)^{\leq p(|x|)}$  such that  $M$  on  $x$  relative to  $B$  accepts with probability greater than  $\frac{3}{4}$ . So,  $N$  decides  $L$ . The runtime of  $N$  is  $2^{cp(n)}$  for some constant  $c > 0$ . Thus,  $L \in \text{NEXP}$ .  $\square$

### 6.4.2 $\text{NEXP} \subseteq \text{MIP}$

The rest of the section proves the other inclusion:

**Theorem 6.24**  $\text{NEXP} \subseteq \text{MIP}$ .

Let  $L$  be a language in  $\text{NEXP}$  and let  $N_L$  be a one-tape  $\text{NEXP}$  machine that decides  $L$ . By Theorem 6.21, we have only to show that there is a probabilistic polynomial time-bounded oracle Turing machine  $M$  witnessing that  $L$  has a probabilistic oracle protocol. Our proof is in three phases:

- Phase 1** conversion of the membership question in  $L$  into a logical expression;
- Phase 2** conversion of the logical expression into an arithmetic expression; and
- Phase 3** development of an interactive oracle protocol for verifying the value of the arithmetic expression.



**6.4.2.1 Encoding Membership Questions.** In the first phase we convert the membership questions in  $L$  to logical expressions. By the tableau method, there is a polynomial  $p$  such that, for every  $x \in \Sigma^*$ , there is a 3CNF formula  $\varphi_x$  of  $2^{p(|x|)}$  variables such that  $x \in L$  if and only if  $\varphi_x$  is satisfiable. For a variable  $X$ , we write  $X = 1$  to denote the literal  $X$  and  $x = 0$  to denote the literal  $\bar{X}$ . Then, we can assume that each clause in  $\varphi_x$  is of the form

$$(X(n_1) = b_1) \vee (X(n_2) = b_2) \vee (X(n_3) = b_3)$$

for some  $n_1, n_2, n_3 \in \{0, \dots, 2^{p(|x|)} - 1\}$  and some  $b_1, b_2, b_3 \in \{0, 1\}$ . For each  $x \in \Sigma^*$ , let the binary strings of length  $p(|x|)$  encode the numbers in  $\{0, \dots, 2^{p(|x|)} - 1\}$ . Define the polynomial  $p'$  by:  $p'(n) = 3p(n) + 3$ . Then, for every  $x \in \Sigma^*$ , the set of all  $p'(|x|)$ -bit binary strings  $y = n_1 n_2 n_3 b_1 b_2 b_3$  is isomorphic to the set of all clauses that potentially appear in  $\varphi_x$ . There are exponentially many clauses in  $\varphi_x$ , so no polynomial-time algorithm can compute the entire description of  $\varphi_x$ . However, a polynomial-time algorithm *can* check, given  $x \in \Sigma^*$  and a potential clause  $y$ , whether  $y$  actually appears in  $\varphi_x$ . Define

$$B = \{x\#y \mid x \in \Sigma^* \wedge y \in \Sigma^{p'(|x|)} \wedge y \text{ appears in } \varphi_x\}.$$

Then  $B \in P$ . Furthermore, for every  $x \in \Sigma^*$ ,  $x \in L$  if and only if

(\\$) there is an assignment  $\mathcal{A}$  such that, for every potential clause  $y = n_1 n_2 n_3 b_1 b_2 b_3$ , if  $y$  is actually a clause in  $\varphi_x$ , then  $\mathcal{A}$  satisfies  $y$ .

By viewing  $\mathcal{A}$  as a mapping from  $\{0, 1\}^{p(|x|)}$  to  $\{0, 1\}$ , (\$) can be rewritten as

$$\begin{aligned} & (\exists \mathcal{A} : \{0, 1\}^{p(|x|)} \rightarrow \{0, 1\}) \\ & (\forall y = n_1 n_2 n_3 b_1 b_2 b_3 \in \{0, 1\}^{p'(|x|)}) \\ & [x\#y \in B \implies \\ & (\mathcal{A}(n_1) = b_1) \vee (\mathcal{A}(n_2) = b_2) \vee (\mathcal{A}(n_3) = b_3)]. \end{aligned} \tag{6.2}$$

Noting that  $B \in P$ , consider a nondeterministic polynomial time-bounded machine  $N_B$  that, on input  $x \in \Sigma^*$ , guesses a clause  $y$ , and then accepts if and only if  $x\#y \in B$ . Since for all  $x, x', y, y' \in \Sigma^*$ , if  $|x| = |x'|$  and  $|y| = |y'|$ , then  $|x\#y| = |x'\#y'|$ , by applying the tableau method to  $N_B$  and by adding some dummy variables and some dummy clauses, we obtain in polynomial time, for each  $x \in \Sigma^*$ , a generic 3CNF formula  $\zeta_x$  with the following properties:

- There is a polynomial  $q$  depending only on  $L$  such that  $\zeta_x$  has  $p'(|x|) + q(|x|)$  variables.
- There is a polynomial  $m$  depending only on  $L$  such that  $\zeta_x$  has  $m(|x|)$  clauses  $C_1, \dots, C_{m(|x|)}$ .
- For every  $y \in \Sigma^{p'(|x|)}$ ,  $x\#y \in B$  if and only if there is a satisfying assignment  $a$  of  $\zeta_x$  such that  $y$  is the length  $p'(|x|)$  prefix of  $a$ .

Then we can rewrite equation 6.2 as

$$\begin{aligned} & (\exists \mathcal{A} : \{0, 1\}^{p(|x|)} \rightarrow \{0, 1\}) \\ & (\forall y = n_1 n_2 n_3 b_1 b_2 b_3 \in \{0, 1\}^{p'(|x|)} (\forall z \in \{0, 1\}^{q(|x|)}) \\ & [\zeta_x(yz) \implies [(\mathcal{A}(n_1) = b_1) \vee (\mathcal{A}(n_2) = b_2) \vee (\mathcal{A}(n_3) = b_3)])]. \end{aligned} \quad (6.3)$$

The condition inside the outer brackets  $[\ ]$  of the formula, i.e.,

$$\zeta_x(yz) \implies [(\mathcal{A}(n_1) = b_1) \vee (\mathcal{A}(n_2) = b_2) \vee (\mathcal{A}(n_3) = b_3)]$$

is equivalent to

$$\neg \zeta_x(yz) \vee (\mathcal{A}(n_1) = b_1) \vee (\mathcal{A}(n_2) = b_2) \vee (\mathcal{A}(n_3) = b_3),$$

and thus, is equivalent to

$$\begin{aligned} & \neg C_1(yz) \vee \dots \vee \neg C_{m(|x|)}(yz) \vee \\ & (\mathcal{A}(n_1) = b_1) \vee (\mathcal{A}(n_2) = b_2) \vee (\mathcal{A}(n_3) = b_3). \end{aligned} \quad (6.4)$$

Define  $p''(n) = p'(n) + q(n)$ . Then equation 6.3 is equivalent to

$$(\exists \mathcal{A} : \{0, 1\}^{p(|x|)} \rightarrow \{0, 1\}) (\forall w \in \{0, 1\}^{p''(|x|)}) [Z(\mathcal{A}; w) = 1], \quad (6.5)$$

where

$$\begin{aligned} Z(\mathcal{A}, w) &= [\neg C_1(w) \vee \dots \vee \neg C_{m(|x|)}(w) \vee \\ & (\mathcal{A}(n_1) = b_1) \vee (\mathcal{A}(n_2) = b_2) \vee (\mathcal{A}(n_3) = b_3)] \end{aligned}$$

and  $w$  is of the form  $n_1 n_2 n_3 b_1 b_2 b_3 z$  such that  $|n_1| = |n_2| = |n_3| = p(|x|)$ ,  $b_1, b_2, b_3 \in \{0, 1\}$ , and  $|z| = q(|x|)$ .

**6.4.2.2 Converting the Logical Expression to Arithmetic Expressions.** In the second phase we obtain an arithmetic expression of equation 6.5. For simplicity, in the following discussion we fix a string  $x$  whose membership in  $L$  we are testing. Let  $m = m(|x|)$ ,  $p = p(|x|)$ , and  $p'' = p''(|x|)$ .

To construct our arithmetic form, we first replace equation 6.4 by

$$f_x(\mathcal{A}; \xi_1, \dots, \xi_{p''}) = \beta_1 \beta_2 \beta_3 \prod_{1 \leq i \leq m} \alpha_i. \quad (6.6)$$

Here

$$\begin{aligned} \beta_1 &= (\mathcal{A}(\xi_1, \dots, \xi_p) - \xi_{3p+1})^2, \\ \beta_2 &= (\mathcal{A}(\xi_{p+1}, \dots, \xi_{2p}) - \xi_{3p+2})^2, \\ \beta_3 &= (\mathcal{A}(\xi_{2p+1}, \dots, \xi_{3p}) - \xi_{3p+3})^2, \end{aligned}$$

and, for each  $i$ ,  $1 \leq i \leq m$ ,

$$\alpha_i = (\xi_{k_{i,1}} - c_{i,1})^2 + (\xi_{k_{i,2}} - c_{i,2})^2 + (\xi_{k_{i,3}} - c_{i,3})^2,$$

where  $C_i(yz) = (\xi_{k_{i,1}} = (1 - c_{i,1})) \vee (\xi_{k_{i,2}} = (1 - c_{i,2})) \vee (\xi_{k_{i,3}} = (1 - c_{i,3}))$ . Then, for every  $\mathcal{A} : \{0, 1\}^p \rightarrow \{0, 1\}$ , the following conditions hold:

- If  $\mathcal{A}$  is a polynomial of total degree at most  $d$ , then  $f_x$  is a polynomial of total degree at most  $6d + 2m$ .

- For every  $a_1, \dots, a_{p''} \in \{0, 1\}^{p''}$ ,  $0 \leq f_x(\mathcal{A}; a_1, \dots, a_{p''}) \leq 3^m$ .
- $f_x(\mathcal{A}; a_1, \dots, a_{p''}) = 0$  if and only if equation 6.4 holds.

Then, for every  $\mathcal{A} : \{0, 1\}^p \rightarrow \{0, 1\}$ ,

$$(\forall w \in \{0, 1\}^{p''(|x|)})[Z(\mathcal{A}; w) = 1]$$

if and only if

$$(\forall (a_1, \dots, a_{p''}) \in \{0, 1\}^{p''}) [f_x(\mathcal{A}; a_1, \dots, a_{p''}) = 0]. \quad (6.7)$$

To determine whether  $x \in L$  we test whether there exists a function  $\mathcal{A} : \{0, 1\}^p \rightarrow \{0, 1\}$  for which equation 6.7 holds.

Let  $F$  be a field. Suppose that  $\mathcal{A}$  is obtained from some  $\mathcal{B} : F^p \rightarrow F$  by restricting its domain to  $\{0, 1\}^p$ . Then,  $x \in L$  if and only if there exists some  $\mathcal{B} : F^p \rightarrow F$  that satisfies the following two conditions:

- For every  $(a_1, \dots, a_p) \in \{0, 1\}^p$ ,  $\mathcal{B}(a_1, \dots, a_p) \in \{0, 1\}$ .
- For every  $(a_1, \dots, a_{p''}) \in \{0, 1\}^{p''}$ ,  $f_x(\mathcal{B}; a_1, \dots, a_{p''}) = 0$ .

What field  $F$  and what kind of function  $\mathcal{B}$  satisfy these conditions in the case when  $x \in L$ ? The following fact shows that a polynomial of total degree at most  $p$  is sufficient.

**Fact 6.25** *Let  $Q$  be an arbitrary prime number greater than  $3^m$ . Then  $x \in L$  if and only if there exists a  $\mathcal{B} : \mathbb{Z}_Q^p \rightarrow \mathbb{Z}_Q$  such that*

1.  $\mathcal{B}$  is a polynomial of total degree at most  $p$ ,
2. for every  $(a_1, \dots, a_p) \in \{0, 1\}^p$ ,  $\mathcal{B}(a_1, \dots, a_p) \bmod Q \in \{0, 1\}$ , and
3. for every  $(a_1, \dots, a_{p''}) \in \{0, 1\}^{p''}$ ,  $f_x(\mathcal{B}; a_1, \dots, a_{p''}) \equiv 0 \pmod{Q}$ .

**Proof of Fact 6.25** Suppose that there exists an oracle  $\mathcal{A}$  that satisfies equation 6.7. Define a multivariate polynomial  $\mathcal{B}$  by:

$$\mathcal{B}(\xi_1, \dots, \xi_p) = \sum_{(c_1, \dots, c_p) \in \{0, 1\}^p} \mathcal{A}(c_1, \dots, c_p) \prod_{1 \leq i \leq p} \ell_{c_i}(\xi_i),$$

where  $\ell_0(\xi) = 1 - \xi$  and  $\ell_1(\xi) = \xi$ . Then  $\mathcal{B}$  is a multilinear polynomial in  $\xi_1, \dots, \xi_p$ , i.e., a polynomial that is linear in each of the variables  $\xi_1, \dots, \xi_p$ . This implies that  $\mathcal{B}$  is a polynomial of total degree at most  $p$ . Thus, property 1 holds. For all  $c = (c_1, \dots, c_p)$  and  $a = (a_1, \dots, a_p) \in \{0, 1\}^p$ ,  $\prod_{1 \leq i \leq p} \ell_{c_i}(a_i)$  is equal to 1 if  $c = a$  and is equal to 0 otherwise. Then, for every  $(a_1, \dots, a_p) \in \{0, 1\}^p$ ,  $\mathcal{A}(a_1, \dots, a_p) = \mathcal{B}(a_1, \dots, a_p)$ . Since the values of  $\mathcal{A}$  are restricted to 0 and 1, property 2 holds. This implies that for every  $(a_1, \dots, a_{p''}) \in \{0, 1\}^{p''}$ ,  $f_x(\mathcal{B}; a_1, \dots, a_{p''}) \in \{0, 1\}$  and that for every  $(a_1, \dots, a_{p''}) \in \{0, 1\}^{p''}$ ,  $f_x(\mathcal{B}; a_1, \dots, a_{p''}) = 0 \iff f_x(\mathcal{B}; a_1, \dots, a_{p''}) \equiv 0 \pmod{Q}$ . Thus, property 3 holds.

On the other hand, suppose that there is a polynomial  $\mathcal{B}$  of total degree at most  $p$  that satisfies properties 1, 2, and 3. Let  $\mathcal{B}'$  denote the function

constructed from  $\mathcal{B}$  by restricting its domain to  $\{0, 1\}^p$  and by taking modulo  $Q$  of the value. By property 2, we can treat  $\mathcal{B}'$  as if its values were restricted to 0 and 1. Recall that for every oracle  $\mathcal{A}$ , and every  $(a_1, \dots, a_{p''}) \in \{0, 1\}^{p''}$ ,

$$0 \leq f_x(\mathcal{A}; a_1, \dots, a_{p''}) \leq 3^m.$$

Since  $Q > 3^m$ , for every  $(a_1, \dots, a_{p''}) \in \{0, 1\}^{p''}$ ,  $f_x(\mathcal{B}'; a_1, \dots, a_{p''}) = 0$  if and only if  $f_x(\mathcal{B}'; a_1, \dots, a_{p''}) \equiv 0 \pmod{Q}$ . Thus, equation 6.7 holds. This implies that  $x \in L$ .  $\square$  Fact 6.25

**6.4.2.3 Developing the Probabilistic Oracle Protocol.** Now we develop a probabilistic oracle Turing machine  $M$  that tests the membership of  $x$  in  $L$  based on Fact 6.25.

We assume that the part of the oracle providing information about the membership of  $x$  in  $L$  has a binary encoding of an integer  $Q \in [2^{2m}, 2^{4m}]$ , which is supposedly a prime number, and a certificate of its primality that can be verified in polynomial time. This assumption is valid. See the discussion in the proof of Lemma 6.11 on page 131.

We also assume that the part of the oracle corresponding to  $x$  has information about the function  $\widehat{\mathcal{B}} : \mathbb{Z}_Q^p \rightarrow \mathbb{Z}_Q$ , which the oracle claims has the properties in the statement of Fact 6.25.

We define a family of polynomials  $\mathcal{B}_0, \dots, \mathcal{B}_p$ . Define for all  $\xi_1, \dots, \xi_p \in \mathbb{Z}_Q$ ,  $\mathcal{B}_0 : \mathbb{Z}_Q^p \rightarrow \mathbb{Z}_Q$  by

$$\mathcal{B}_0(\xi_1, \dots, \xi_p) = \widehat{\mathcal{B}}(\xi_1, \dots, \xi_p)(1 - \widehat{\mathcal{B}}(\xi_1, \dots, \xi_p)). \quad (6.8)$$

For each  $i$ ,  $1 \leq i \leq p$ , define

$$\begin{aligned} \mathcal{B}_i(\xi_1, \dots, \xi_p) = & \\ & \mathcal{B}_{i-1}(\xi_1, \dots, \xi_{i-1}, 0, \xi_{i+1}, \dots, \xi_p) \\ & + \mathcal{B}_{i-1}(\xi_1, \dots, \xi_{i-1}, 1, \xi_{i+1}, \dots, \xi_p)\xi_i. \end{aligned} \quad (6.9)$$

The functions  $\widehat{\mathcal{B}}, \mathcal{B}_0, \dots, \mathcal{B}_p$  have the following properties.

**Fact 6.26**

1. If  $\widehat{\mathcal{B}}$  is a polynomial of total degree at most  $p$ , then, for every  $i$ ,  $0 \leq i \leq p$ ,  $\mathcal{B}_i$  is a polynomial of total degree at most  $2p + i$ .
2. For every  $i$ ,  $0 \leq i \leq p$ ,

$$\mathcal{B}_i(\xi_1, \dots, \xi_p) = \sum_{c_1, \dots, c_i \in \{0, 1\}} \mathcal{B}_0(c_1, \dots, c_i, \xi_{i+1}, \dots, \xi_p) \prod_{1 \leq j \leq i} \xi_j^{c_j}.$$

3. If for all  $(\xi_1, \dots, \xi_p) \in \{0, 1\}^p$  it holds that  $\widehat{\mathcal{B}}(\xi_1, \dots, \xi_p) \equiv 0 \pmod{Q}$ , then, for all  $(\xi_1, \dots, \xi_p) \in \mathbb{Z}_Q^p$ ,  $\mathcal{B}_p \equiv 0 \pmod{Q}$ .

**Proof** If  $\widehat{\mathcal{B}}$  is a polynomial of total degree  $p$ , then  $\mathcal{B}_0$  is a polynomial of total degree  $2p$ . For every  $i$ ,  $1 \leq i \leq p$ , if  $\mathcal{B}_{i-1}$  is a polynomial of total degree

at most  $2p + i - 1$ , then  $\mathcal{B}_i$  is a polynomial of total degree at most  $2p + i$ . So, part 1 holds.

We prove part 2 by induction on  $i$ . For the base case, the equality trivially holds. For the induction step, suppose that  $i = i_0$  for some  $1 \leq i_0 \leq p$  and that the claim holds for all values of  $i$  that are greater than or equal to 0 and less than or equal to  $i_0 - 1$ . By the induction hypothesis,

$$\mathcal{B}_{i-1}(\xi_1, \dots, \xi_p) = \sum_{c_1, \dots, c_{i-1} \in \{0,1\}} \mathcal{B}_0(c_1, \dots, c_{i-1}, \xi_i, \dots, \xi_p) \prod_{1 \leq j \leq i-1} \xi_j^{c_j},$$

and by definition

$$\begin{aligned} \mathcal{B}_i(\xi_1, \dots, \xi_p) &= \mathcal{B}_{i-1}(\xi_1, \dots, \xi_{i-1}, 0, \xi_{i+1}, \dots, \xi_p) \\ &\quad + \mathcal{B}_{i-1}(\xi_1, \dots, \xi_{i-1}, 1, \xi_{i+1}, \dots, \xi_p) \xi_i. \end{aligned}$$

By combining the two equalities, we have

$$\begin{aligned} \mathcal{B}_i(\xi_1, \dots, \xi_p) &= \sum_{c_1, \dots, c_{i-1} \in \{0,1\}} \mathcal{B}_0(c_1, \dots, c_{i-1}, 0, \xi_{i+1}, \dots, \xi_p) \prod_{1 \leq j \leq i-1} \xi_j^{c_j} \\ &\quad + \sum_{c_1, \dots, c_{i-1} \in \{0,1\}} \prod_{1 \leq j \leq i-1} [\mathcal{B}_0(c_1, \dots, c_{i-1}, 1, \xi_{i+1}, \dots, \xi_p) \xi_j^{c_j}] \xi_i \\ &= \sum_{c_1, \dots, c_i \in \{0,1\}} \mathcal{B}_0(c_1, \dots, c_i, \xi_{i+1}, \dots, \xi_p) \prod_{1 \leq j \leq i} \xi_j^{c_j}. \end{aligned}$$

Thus, part 2 holds.

To prove part 3, note that

$$\mathcal{B}_p = \sum_{c_1, \dots, c_p \in \{0,1\}} \prod_{1 \leq j \leq p} \xi_j^{c_j} \mathcal{B}_0(c_1, \dots, c_p).$$

If for all  $(\xi_1, \dots, \xi_p) \in \{0,1\}^p$  it holds that  $\mathcal{B}_0(\xi_1, \dots, \xi_p) \equiv 0 \pmod{Q}$ , then for all  $(c_1, \dots, c_p) \in \{0,1\}^p$   $\mathcal{B}(c_1, \dots, c_p) \equiv 0 \pmod{Q}$ . This implies that  $\mathcal{B}_p \equiv 0 \pmod{Q}$ .  $\square$

We define a family of polynomials  $\mathcal{C}_0, \dots, \mathcal{C}_{p''}$ . Let  $\mathcal{C}_0 = f_x(\widehat{\mathcal{B}}; a_1, \dots, a_{p''})$ . For each  $i$ ,  $1 \leq i \leq p''$ , define

$$\begin{aligned} \mathcal{C}_i(\xi_1, \dots, \xi_{p''}) &= \\ \mathcal{C}_{i-1}(x_1, \dots, x_{i-1}, 0, \xi_{i+1}, \dots, \xi_{p''}) & \\ + \mathcal{C}_{i-1}(x_1, \dots, x_{i-1}, 1, \xi_{i+1}, \dots, \xi_{p''}) \xi_i. & \end{aligned} \tag{6.10}$$

The functions  $\mathcal{C}_0, \dots, \mathcal{C}_{p''}$  have the following properties.

**Fact 6.27**

1. If  $\widehat{\mathcal{B}}$  is a polynomial of total degree at most  $p$ , then for every  $i$ ,  $0 \leq i \leq p''$ ,  $\mathcal{C}_i$  is a polynomial of total degree at most  $(6p + 2m) + i$ .

2. For every  $i$ ,  $1 \leq i \leq p''$ ,

$$C_i = \sum_{c_1, \dots, c_i \in \{0,1\}} \prod_{1 \leq j \leq i} C(c_1, \dots, c_i, \xi_{i+1}, \dots, \xi_{p''})^{\xi_j^{c_j}}.$$

3. If for all  $(\xi_1, \dots, \xi_{p''}) \in \{0,1\}^{p''}$  it holds that  $C_0(\xi_1, \dots, \xi_{p''}) \equiv 0 \pmod{Q}$ , then, for all  $(\xi_1, \dots, \xi_{p''}) \in \mathbb{Z}_Q^{p''}$ ,  $C_{p''} \equiv 0 \pmod{Q}$ .

**Proof** Suppose that  $\widehat{B}$  is a polynomial of total degree at most  $p$ . Then  $C_0$  is a polynomial of total degree at most  $6p + 2m$ . For every  $i$ ,  $1 \leq i \leq p''$ , and every  $d \geq 1$ , if  $C_{i-1}$  is a polynomial of total degree at most  $d$ , then  $C_i$  is a polynomial of total degree at most  $d + 1$ . Thus, part 1 holds.

Parts 2 and 3 can be proven exactly the same way as we proved the corresponding parts of Fact 6.26.  $\square$

We assume that the oracle has functions  $\widehat{G}, \mathcal{G}_0, \dots, \mathcal{G}_p, \mathcal{H}_0, \dots, \mathcal{H}_{p''}$  which the oracle claims and  $\widehat{B}, \mathcal{B}_0, \dots, \mathcal{B}_p, C_0, \dots, C_{p''}$ , respectively, and that the oracle tries to convince  $M$  that both  $\mathcal{G}_p$  and  $\mathcal{H}_{p''}$  are zero functions modulo  $Q$ . To test the claim by the oracle, the machine  $M$  executes a sequence of protocols. The following conditions are tested by the sequence of protocols.

- (A)  $\widehat{G}$  is a polynomial of total degree at most  $p$ .
- (B) For each  $i$ ,  $0 \leq i \leq p$ ,  $\mathcal{G}_i$  is a polynomial of total degree at most  $2p + i$ .
- (C) For each  $i$ ,  $0 \leq i \leq p''$ ,  $\mathcal{H}_i$  is a polynomial of total degree at most  $6p + 2m + i$ .
- (D) Equation 6.8 holds with  $\mathcal{G}_0$  and  $\widehat{G}$  in place of  $\mathcal{B}_0$  and  $\widehat{B}$ , respectively.
- (E) For each  $i$ ,  $1 \leq i \leq p$ , equation 6.9 holds with  $\mathcal{G}_i$  and  $\mathcal{G}_{i-1}$  in place of  $\mathcal{B}_i$  and  $\mathcal{B}_{i-1}$ , respectively.
- (F) For each  $i$ ,  $1 \leq i \leq p''$ , equation 6.10 holds with  $\mathcal{H}_i$  and  $\mathcal{H}_{i-1}$  in place of  $C_i$  and  $C_{i-1}$ , respectively.
- (G)  $\mathcal{H}_0 \equiv f_x(\mathcal{G}_0; k, \xi_1, \dots, \xi_{p''-1}) \pmod{Q}$ .
- (H)  $\mathcal{G}_p \equiv 0 \pmod{Q}$ .
- (I)  $\mathcal{H}_{p''} \equiv 0 \pmod{Q}$ .

The machine  $M$  executes the following to test these conditions.

- To test (A),  $M$  executes the **Low-Degree Test** in Fig. 6.7 with  $s = p$ ,  $d = p$ , and  $\mathcal{U} = \widehat{G}$ .
- To test (B), for each  $i$ ,  $0 \leq i \leq p$ ,  $M$  executes the **Low-Degree Test** in Fig. 6.7 with  $s = p$ ,  $d = 2p + i$ , and  $\mathcal{U} = \mathcal{G}_i$ .
- To test (C), for each  $i$ ,  $0 \leq i \leq p''$ ,  $M$  executes the **Low-Degree Test** in Fig. 6.7 with  $s = p''$ ,  $d = 6p + 2m + i$ , and  $\mathcal{U} = \mathcal{H}_i$ .
- To test (D),  $M$  executes the **G-Equality Test** in Fig. 6.8.
- To test (E), for each  $i$ ,  $1 \leq i \leq p$ ,  $M$  executes the **Self-Correcting Polynomial Equality Test** in Fig. 6.9 with  $s = p$ ,  $d = p + i - 1$ ,  $\mathcal{U} = \mathcal{G}_{i-1}$ , and  $\mathcal{V} = \mathcal{G}_i$ .

- To test (F), for each  $i$ ,  $1 \leq i \leq p''$ ,  $M$  executes the **Self-Correcting Polynomial Equality Test** in Fig. 6.9 with  $s = p''$ ,  $d = 6p + 2m + i - 1$ ,  $\mathcal{U} = \mathcal{H}_{i-1}$ , and  $\mathcal{V} = \mathcal{H}_i$ .
- To test (G),  $M$  executes the  **$\mathcal{H}_0$ - $f_x$  Equality Test** in Fig. 6.10.
- To test (H),  $M$  executes the **Zero Polynomial Test** in Fig. 6.11 with  $s = p$  and  $\mathcal{U} = \mathcal{G}_p$ .
- To test (I),  $M$  executes the **Zero Polynomial Test** in Fig. 6.11 with  $s = p''$  and  $\mathcal{U} = \mathcal{H}_{p''}$ .

If all the tests succeed,  $M$  accepts  $x$ .

We will now prove that  $M$  witnesses that  $L \in \text{NEXP}$  by proving that  $M$  can be polynomial time-bounded, that the protocol is complete (i.e., if  $x \in L$ ,  $M$  accepts with probability more than  $\frac{3}{4}$ ), and that the protocol is sound (i.e., if  $x \notin L$ ,  $M$  rejects with probability more than  $\frac{3}{4}$ ).

**6.4.2.4 Running-Time Analysis of the Protocol.** Since  $Q \leq 2^{4m}$ , the sampling algorithm runs in time polynomial in  $|x|$ . Let  $g(s, d) = 8s(d+2)^2$ . For a single run of the Low-Degree Test with parameters  $s$  and  $d$ ,  $8s(d+2)^2$  samples are used. Then, the number of samples used to test (A) is  $8p(p+2)^2 = \mathcal{O}(p^3)$ , the total number of samples used to test (B) is

$$\sum_{2p \leq d \leq 3p} (8p(d+2)^2) = \mathcal{O}(p^4),$$

and the total number of samples used to test (C) is

$$\sum_{6p+2m \leq d \leq 6p+2m+p''} 8p''(d+2)^2 = \mathcal{O}((p+m+p'')^4).$$

For the  $G$ -Equality Test, the number of samples used is  $p$ . For a single run of the Self-Correcting Polynomial Equality Test with parameters  $s$  and  $d$ , the number of samples used is  $s + d + 1$ . Then, the total number of samples used to test (E) is

$$\sum_{p \leq d \leq 2p} (p + d + 1) = \mathcal{O}(p^2),$$

and the total number of samples used to test (F) is

$$\sum_{6p+2m \leq d \leq 6p+2m+p''} (p'' + d + 1) = \mathcal{O}((p+m+p'')^2).$$

The numbers of samples used to test (G), (H), and (I) are  $p''$ ,  $p$ , and  $p''$ , respectively. Thus, the grand total of the number of samples used is  $\mathcal{O}((p+m+p'')^4)$ . This is  $\mathcal{O}(n^k)$  for some fixed constant  $k \geq 1$ . All the other arithmetic operations required in the protocol can be done in polynomial time. Thus,  $M$  can be polynomial time bounded.

Repeat the following  $6(d+2)^2$  times.

- Step 1** Select  $y, z \in (\mathbb{Z}_Q)^s$  independently and uniformly at random. To select an entry use the Sampling Algorithm in Fig. 6.4 with  $M = Q$  and  $t = p$ .  
**Step 2** For each  $i$ ,  $0 \leq i \leq d+1$ , obtain from the oracle the value of  $\mathcal{U}(y + iz)$ .  
**Step 3** Test whether  $\sum_{0 \leq i \leq d+1} \alpha_{d,i} \mathcal{U}(y + iz) \equiv 0 \pmod{Q}$ , where for every  $d \geq 0$  and every  $i$ ,  $0 \leq i \leq d+1$ ,  $\alpha_{d,i} = (-1)^{i+1} \binom{d}{i}$ .  
 If the test fails then immediately reject the input  $x$ .

**Fig. 6.7** The low-degree test

- Step 1** Using the sampling algorithm select  $y_1, \dots, y_p$  from  $\mathbb{Z}_Q$ .  
**Step 2** Use the oracle to obtain  $u = \widehat{\mathcal{G}}(y_1, \dots, y_p)$  and  $v = \mathcal{G}_0(y_1, \dots, y_p)$ .  
**Step 3** Test whether  $u(1-u) \equiv v \pmod{Q}$ . If the test fails, then reject  $x$  immediately.

**Fig. 6.8** The  $G$ -equality test

- Step 1** Use the sampling algorithm to select  $y_1, \dots, y_s, z_0, \dots, z_d$  from  $\mathbb{Z}_Q$ .  
**Step 2** Test whether  $z_0, \dots, z_d$  are pairwise distinct. If the test fails then accept  $x$ .  
**Step 3** Set  $\tilde{w}$  to  $(y_1, \dots, y_s)$  and for each  $j$ ,  $0 \leq j \leq d$ , set  $w_j$  to  $\tilde{w}$  with the  $i$ th entry replaced by  $z_j$ . Obtain from the oracle  $\tilde{u} = \mathcal{V}(\tilde{w})$  and for each  $i$ ,  $0 \leq i \leq d$ ,  $u_i = \mathcal{U}(w_i)$ .  
**Step 4** Set  $A$  to the  $(d+1) \times (d+1)$  matrix

$$A = \begin{pmatrix} 1 & z_0 & z_0^2 & \cdots & z_0^d \\ 1 & z_1 & z_1^2 & \cdots & z_1^d \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & z_d & z_d^2 & \cdots & z_d^d \end{pmatrix}.$$

Use some polynomial-time algorithm (e.g., Gaussian elimination) to compute  $A^{-1}$ .

- Step 5** Compute  $c_0, \dots, c_d$  by

$$\begin{pmatrix} c_0 \\ \vdots \\ c_d \end{pmatrix} = A^{-1} \begin{pmatrix} u_0 \\ \vdots \\ u_d \end{pmatrix}.$$

and set  $t(\theta)$  to the polynomial  $c_0 + c_1\theta + \cdots + c_d\theta^d$ .

- Step 6** Compute  $v_0$  as  $t(0) \bmod Q$  and  $v_1$  as  $t(1) \bmod Q$ . Test whether  $v_0 + v_1 y_i \equiv \tilde{u} \pmod{Q}$ . If the test fails then reject  $x$  immediately.

**Fig. 6.9** The self-correcting polynomial equality test



- Step 1** Using the sampling algorithm select  $y_1, \dots, y_{p''}$  from  $\mathbb{Z}_Q$ .
- Step 2** Use the oracle to obtain  $u_0 = \mathcal{H}_0(y_1, \dots, y_{p''})$ ,  $u_1 = \widehat{\mathcal{G}}(y_1, \dots, y_p)$ ,  $u_2 = \widehat{\mathcal{G}}(y_{p+1}, \dots, y_{2p})$ , and  $u_3 = \widehat{\mathcal{G}}(y_{2p+1}, \dots, y_{3p})$ .
- Step 3** Evaluate  $\prod_{1 \leq i \leq m} \alpha_i(y_1, \dots, y_{p''})$  modulo  $Q$ , where  $\alpha_i$  is the polynomial appearing in the definition of  $f_x$ .
- Step 4** Test whether  $u_0 \equiv (u_1 - y_{3p+1})^2 (u_2 - y_{3p+2})^2 (u_3 - y_{3p+3})^2 u_4 \pmod{Q}$ . If the test fails then reject  $x$  immediately.

Fig. 6.10 The  $\mathcal{H}_0$ - $f_x$  equality test

- Step 1** Use the sampling algorithm to select  $y \in (\mathbb{Z}_Q)^s$ .
- Step 2** Obtain from the oracle  $u = \mathcal{U}(y)$ . If  $u \not\equiv 0 \pmod{Q}$  reject  $x$  immediately.

Fig. 6.11 The zero polynomial test

**6.4.2.5 Completeness of the Protocol.** Next we show that the protocol is complete. Suppose that  $x \in L$ . Then there exists some polynomial  $\widehat{B}$  of total degree at most  $p$  that satisfies all the conditions in Fact 6.25. Take the oracle functions  $\widehat{\mathcal{G}}, \mathcal{G}_0, \dots, \mathcal{G}_p, \mathcal{H}_0, \dots, \mathcal{H}_{p''}$  that are equal to  $\widehat{B}, \mathcal{B}_0, \dots, \mathcal{B}_p, \mathcal{C}_0, \dots, \mathcal{C}_{p''}$ , respectively. Then, by Facts 6.26 and 6.27, the oracle functions satisfy the following conditions:

- $\mathcal{B}_0(\xi_1, \dots, \xi_p) = \widehat{B}_0(\xi_1, \dots, \xi_p)(1 - \widehat{B}_0(\xi_1, \dots, \xi_p))$ .
- For every  $i$ ,  $1 \leq i \leq p$ ,

$$\begin{aligned} \mathcal{G}_i(\xi_1, \dots, \xi_p) = & \\ & \mathcal{G}_{i-1}(x_1, \dots, x_{i-1}, 0, \xi_{i+1}, \dots, \xi_p) \\ & + \mathcal{G}_{i-1}(x_1, \dots, x_{i-1}, 1, \xi_{i+1}, \dots, \xi_p) \xi_i. \end{aligned}$$

- For every  $i$ ,  $0 \leq i \leq p$ ,  $\mathcal{G}_i$  is a polynomial of total degree at most  $2p + i$ .
- For all  $(\xi_1, \dots, \xi_p) \in \mathbb{Z}_Q^p$ ,  $\mathcal{G}_p(\xi_1, \dots, \xi_p) \equiv 0 \pmod{Q}$ .
- For every  $i$ ,  $1 \leq i \leq p''$ ,

$$\begin{aligned} \mathcal{H}_i(\xi_1, \dots, \xi_{p''}) = & \\ & \mathcal{H}_{i-1}(x_1, \dots, x_{i-1}, 0, \xi_{i+1}, \dots, \xi_{p''}) \\ & + \mathcal{H}_{i-1}(x_1, \dots, x_{i-1}, 1, \xi_{i+1}, \dots, \xi_{p''}) \xi_i. \end{aligned}$$

- $\mathcal{H}_0(x_1, \dots, \xi_{p''}) = f_x(\mathcal{B}; \xi_1, \dots, \xi_{p''})$ .
- For every  $i$ ,  $0 \leq i \leq p''$ ,  $\mathcal{H}_i$  is a polynomial of total degree at most  $6p + 6m + i$ .
- For all  $(\xi_1, \dots, \xi_{p''}) \in \mathbb{Z}_Q^{p''}$ ,  $\mathcal{H}_{p''}(\xi_1, \dots, \xi_{p''}) \equiv 0 \pmod{Q}$ .

Since the protocol is designed to accept on encountering a sampling error and we need to prove here that the probability that  $M$  accepts is greater

than  $\frac{3}{4}$ , we can assume that there is no sampling error. We claim that with probability one all the tests either succeed or force  $M$  to stop computation instantly by accepting  $x$ . Clearly, the Zero Polynomial Test succeeds with probability one for both  $\mathcal{G}_p$  and  $\mathcal{H}_{p''}$ . The Low-Degree Test succeeds, too (again, given that there is no sampling error), which follows from the lemma below. We will give its proof in Sect. 6.4.3.

**Lemma 6.28 (The Low-Degree Polynomial Characterization Lemma)** *Let  $d, s$  be positive integers. Let  $F = \mathbb{Z}_R$  for some prime number  $R$ . For every function  $h : F^s \rightarrow F$ ,  $h$  is a polynomial of total degree at most  $d$  if and only if for all  $y, z \in F^s$ , it holds that*

$$\sum_{0 \leq i \leq d+1} \gamma_i h(y + iz) \equiv 0 \pmod{R},$$

where for every  $i$ ,  $0 \leq i \leq d+1$ ,  $\gamma_i = \binom{d+1}{i}(-1)^i$ .

Lemma 6.28 assures that the Low-Degree Test succeeds with probability one for  $\mathcal{G}_0, \dots, \mathcal{G}_p, \mathcal{H}_0, \dots, \mathcal{H}_{p''}$ . Both the  $G$ -Equality Test and the  $\mathcal{H}_0$ - $f_x$  Equality Test pass with probability one.

Furthermore, we claim that the Self-Correcting Polynomial Equality Test succeeds with probability one each time it is called. To prove the claim, suppose that the test is called for  $i$ ,  $1 \leq i \leq p$ ,  $s = p$ ,  $d = 2p + i - 1$ ,  $\mathcal{U} = \mathcal{G}_{i-1}$ , and  $\mathcal{V} = \mathcal{G}_i$ . Let  $y_1, \dots, y_p, z_0, \dots, z_d \in \mathbb{Z}_Q$ . If  $z_0, \dots, z_d$  are not pairwise distinct, then  $M$  accepts, so suppose that they are pairwise distinct. Let

$$g(\theta) = \mathcal{U}(y_1, \dots, y_{i-1}, 0, y_{i+1}, \dots, y_p) + \mathcal{U}(y_1, \dots, y_{i-1}, 1, y_{i+1}, \dots, y_p)\theta$$

and

$$t(\theta) = \mathcal{U}(y_1, \dots, y_{i-1}, t, y_{i+1}, \dots, y_p).$$

Since  $\mathcal{U}$  is a polynomial of total degree at most  $d = 2p + i - 1$ ,  $t$  is a polynomial in  $\theta$  of degree at most  $d$ . Then there exist unique  $c_0, \dots, c_d \in \mathbb{Z}_Q$  such that  $t(\theta) = \sum_{0 \leq j \leq d} c_j \theta^j$ . For each  $j$ ,  $0 \leq j \leq d$ , let  $w_j = (y_1, \dots, y_{i-1}, z_j, y_{i+1}, \dots, y_p)$  and  $u_j = \mathcal{U}(w_j)$ . Let  $\mathbf{c} = (c_0, \dots, c_d)^T$  and  $\mathbf{u} = (u_0, \dots, u_d)^T$ . Let  $A$  be the matrix specified in the protocol, namely,

$$A = \begin{pmatrix} 1 & z_0 & z_0^2 & \dots & z_0^d \\ 1 & z_1 & z_1^2 & \dots & z_1^d \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & z_d & z_d^2 & \dots & z_d^d \end{pmatrix}.$$

This type of matrix is called a *Vandermonde Matrix*. The following proposition, which we state without a proof, shows that the determinant of a Vandermonde matrix has a simple formula.

**Proposition 6.29** *Let  $d \geq 2$ . Let  $A$  be a Vandermonde matrix of dimension  $d$ , i.e., for some  $a_1, \dots, a_d$ ,  $A$  is of the form*

$$\begin{pmatrix} 1 & a_1 & a_1^2 & \dots & a_1^{d-1} \\ 1 & a_2 & a_2^2 & \dots & a_2^{d-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & a_d & a_d^2 & \dots & a_d^{d-1} \end{pmatrix}.$$

*Then the determinant of  $A$  is  $\prod_{1 \leq i < j \leq d} (a_i - a_j)$ .*

By Proposition 6.29, the determinant of  $A$  is  $\prod_{0 \leq i < j \leq d} (z_i - z_j)$ . Since  $z_0, \dots, z_d$  are pairwise distinct and  $\mathbb{Z}_Q$  is a field, the determinant of  $A$  is nonzero, so  $A$  is nonsingular. Thus,  $A^{-1}$  exists, and so we can set  $\mathbf{c} = A^{-1}\mathbf{u}$ . For all  $\theta \in \mathbb{Z}_Q$ ,  $\mathcal{U}(y_1, \dots, y_{i-1}, \theta, y_{i+1}, \dots, y_p) = t(\theta)$ . Thus, for all  $\theta \in \mathbb{Z}_Q$ ,  $g(\theta) = t(0) + t(1)\theta$ . In particular,  $g(y_i) = t(0) + t(1)y_i$ . By our assumption,  $g(y_i) = \mathcal{V}(y_1, \dots, y_p)$ . The right-hand side is  $\tilde{u}$  of the protocol. Since the protocol accepts if and only if  $\tilde{u} \equiv t(0) + t(1)y_i \pmod{Q}$ , the probability that the test succeeds is 1.

By following a similar argument, we can show that the protocol accepts when the test is called for  $i$ ,  $1 \leq i \leq p''$ ,  $s = p$ ,  $d = 6p + 2m + i - 1$ ,  $\mathcal{U} = \mathcal{H}_{i-1}$ , and  $\mathcal{V} = \mathcal{H}_i$ .

Hence, the probability that  $M$  accepts  $x$  is one. This proves the completeness of the protocol.

**6.4.2.6 Soundness of the Protocol.** Next we show that the protocol is sound. Suppose that  $x \notin L$ . A sampling error occurs in a single run of the sampling algorithm with probability less than  $2^{-p}$ . Since the total number of samples generated in the entire protocol is  $\mathcal{O}((p + m + p'')^4)$ , the probability that a sampling error occurs during the execution of the entire protocol is

$$\mathcal{O}\left(\frac{(p + p'' + m)^4}{2^p}\right).$$

This is less than  $\frac{1}{8}$  for all  $x$  sufficiently large. That is, the probability that  $M$  accepts  $x$  due to a sampling error is less than  $\frac{1}{8}$ . We will show that for every oracle the probability that  $M$  accepts  $x$ , provided that no sampling error occurs, is less than  $\frac{1}{8}$ . Then for every oracle the probability that  $M$  accepts  $x$  is less than  $\frac{1}{8} + \frac{1}{8} = \frac{1}{4}$  as desired. For simplicity, in the following discussion, assume that no sampling error occurs. Let  $\hat{\mathcal{G}}, \mathcal{G}_0, \dots, \mathcal{G}_p : (\mathbb{Z}_Q)^p \rightarrow \mathbb{Z}_Q$ ,  $\mathcal{H}_0, \dots, \mathcal{H}_{p''} : (\mathbb{Z}_Q)^{p''} \rightarrow \mathbb{Z}_Q$  be the oracle functions. We analyze the tests that  $M$  conducts.

**6.4.2.6.1 The Low-Degree Test.** To analyze the Low-Degree Test, we need to define the concept of *closeness*.

**Definition 6.30** *Let  $s \geq 1$  be an integer. Let  $f, g : (\mathbb{Z}_Q)^s \rightarrow \mathbb{Z}_Q$  be functions. Let  $0 \leq \epsilon \leq 1$ . Then  $f$  and  $g$  are said to be  $\epsilon$ -close if the proportion of  $x \in \mathbb{Z}_Q$  such that  $f(x) \neq g(x)$  is at most  $\epsilon$ .*

Then we have the following lemma.

**Lemma 6.31 (The Low-Degree Polynomial Closeness Lemma)** *Let  $s, d \geq 1$  be integers. Let  $\delta = \frac{1}{2(d+2)^2}$ . Let  $f$  be a function from  $(\mathbb{Z}_Q)^s$  to  $\mathbb{Z}_Q$ . Suppose that  $f$  is not  $2\delta$ -close to any polynomial of total degree at most  $d$ . Then  $f$  survives the Low-Degree Test with probability less than  $\frac{1}{8}$ .*

The proof of Lemma 6.31 is long; we defer its proof to Sect. 6.4.4. For now, assume that the lemma is correct.

Let  $\hat{d} = p$  and  $\hat{\delta} = \frac{1}{2(p+2)^2}$ . For each  $i$ ,  $0 \leq i \leq p$ , let  $d_i = 2p + i$  and  $\delta_i = \frac{1}{2(2p+i+2)^2}$ . Also, for each  $i$ ,  $0 \leq i \leq p''$ , let  $d'_i = 6p + 2m + i$  and  $\delta'_i = \frac{1}{2(6p+2m+i+2)^2}$ . Then, by calling the Low-Degree Test,  $M$  checks whether the following conditions are all satisfied:

- $\hat{\mathcal{G}}$  is  $\hat{\delta}$ -close to a polynomial of total degree at most  $\hat{d}$ .
- For all  $i$ ,  $0 \leq i \leq p$ ,  $\mathcal{G}_i$  is  $\delta_i$ -close to a polynomial of total degree at most  $d_i$ .
- For all  $i$ ,  $0 \leq i \leq p''$ ,  $\mathcal{H}_i$  is  $\delta'_i$ -close to a polynomial of total degree at most  $d'_i$ .

Then, by Lemma 6.31, if one of these conditions is not satisfied,  $M$  on input  $x$  rejects with probability more than  $\frac{7}{8}$ . So, in the following discussion, let us assume that these conditions are all satisfied, i.e.,

- (\*)  $\hat{\mathcal{G}}$  is  $\hat{\delta}$ -close to a polynomial  $\hat{g}$  of total degree at most  $\hat{d}$ ;  
for each  $i$ ,  $0 \leq i \leq p$ , there is a polynomial  $g_i$  of total degree at most  $d_i$  that is  $\delta_i$ -close to  $\mathcal{G}_i$ ; and  
for each  $i$ ,  $0 \leq i \leq p''$ , there is a polynomial  $h_i$  of total degree at most  $d'_i$  that is  $\delta'_i$ -close to  $\mathcal{H}_i$ .

The polynomials  $g_0, \dots, g_p, h_0, \dots, h_{p''}$  are uniquely determined, due to the following lemma.

**Lemma 6.32** *Let  $d, s \geq 1$  be integers. Let  $F$  be a finite field and let  $N = ||F||$ . Let  $u : F^s \rightarrow F$  be a nonzero polynomial of total degree at most  $d$ . Then the proportion of  $y \in F^s$  for which  $u(y) = 0$  is at most  $\frac{d}{N}$ .*

**Proof of Lemma 6.32** Let  $d, s, F, N$ , and  $u$  be as in the hypothesis. Let  $T$  be the roots of  $u$  in  $F^s$ , i.e.,  $T = \{y \in F^s \mid u(y) = 0\}$ . To prove the lemma, it suffices to show that the cardinality of  $T$  is at most  $dN^{s-1}$ . We prove this by induction on  $s$ . For the base case, suppose that  $s = 1$ . Then  $u$  is a univariate polynomial of degree at most  $d$ .  $u$  has at most  $d$  distinct roots, so  $||T|| \leq d = dN^{s-1}$  as desired. Thus, the claim holds for  $s = 1$ .

For the induction step, suppose that  $s = s_0$  for some  $s_0 \geq 2$  and that the claim holds for all values of  $s$  that are less than  $s_0$  and greater than or equal to 1. For some  $e$ ,  $0 \leq e \leq d$ ,  $u$  can be written as

$$\sum_{0 \leq i \leq e} v_i(\xi_2, \dots, \xi_s) \xi_1^i,$$

where for every  $i$ ,  $0 \leq i \leq e$ ,  $v_i$  is a polynomial in  $\xi_2, \dots, \xi_s$  of total degree at most  $(d-i)$ , and  $v_e$  is not the zero polynomial. Then, for all  $y = (y_1, \dots, y_s) \in F^s$ ,  $y$  is a root of  $u$  if and only if one of the following conditions (i) and (ii) holds:

- (i)  $(y_2, \dots, y_s)$  is a root of each of  $v_0, \dots, v_e$ .
- (ii) For some  $i$ ,  $0 \leq i \leq e$ ,  $(y_2, \dots, y_p)$  is not a root of  $v_i$ , and  $y_1$  is a root of the nonzero univariate polynomial

$$\sum_{0 \leq i \leq e} v_i(y_2, \dots, y_s) \xi_1^i.$$

By our induction hypothesis, the number of  $y$  for which (i) holds is at most  $((d-e)N^{s-2})N = (d-e)N^{s-1}$  and the number of  $y$  for which (ii) holds is at most  $(N^{s-1})e$ . Thus, the total number of roots of  $u$  is at most  $dN^{s-1}$  as desired. This proves the lemma.  $\square$  Lemma 6.32

Now, the reason that the polynomials  $\hat{g}, g_0, \dots, g_p$  and  $h_0, \dots, h_{p''}$  are unique can be explained as follows: Let  $u$  and  $v$  be two distinct polynomials of total degree  $d$ . Suppose that both  $u$  and  $v$  are  $\rho$ -close to a function  $f: F^s \rightarrow F$ , where  $F$  is a field of size  $N$ . Let  $w = u - v$ . Since  $u \neq v$ ,  $w \neq 0$ . Since  $u$  and  $v$  are  $\rho$ -close to  $f$ ,  $u$  is  $2\rho$ -close to  $v$ . So, the proportion of  $y \in F^s$  such that  $w(y) = 0$  is at least  $1 - 2\rho$ . By Lemma 6.32, the proportion of  $y \in F^s$  such that  $w(y) = 0$  is at most  $\frac{d}{N}$ . This implies  $\frac{d}{N} \geq 1 - 2\rho$ , i.e.,  $\frac{d}{N} + 2\rho \geq 1$ . We claim that this inequality holds for none of  $\hat{g}, g_0, \dots, g_p, h_0, \dots, h_{p''}$ . Since  $N = Q > 2^m$ , the largest value of  $d$  is  $6p + 2m + p''$ , and the largest value of  $\rho$  is  $\frac{1}{2(p+2)^2}$ , and thus,  $\frac{d}{N} + 2\rho$  is at most  $\frac{6p+2m+p''}{2^m} + \frac{1}{(p+2)^2}$ . This is less than 1 for all  $x$  sufficiently large. Hence,  $\hat{g}, g_0, \dots, g_p, h_0, \dots, h_{p''}$  are uniquely defined.

Now that each of  $\hat{\mathcal{G}}, \mathcal{G}_0, \dots, \mathcal{G}_p, \mathcal{H}_0, \dots, \mathcal{H}_{p''}$  is close to a polynomial, we think of the other test as checking the properties (D) through (I) with each of these polynomial replacing its corresponding oracle function, i.e., (D) through (I) are modified as follows:

- (D') Equation 6.8 holds with  $g_0$  and  $\hat{g}$  in place of  $\mathcal{B}_0$  and  $\hat{\mathcal{B}}$ , respectively.
- (E') For each  $i$ ,  $1 \leq i \leq p$ , equation 6.9 holds for  $g_i$  and  $g_{i-1}$  in place of  $\mathcal{B}_i$  and  $\mathcal{B}_{i-1}$ , respectively.
- (F') For each  $i$ ,  $1 \leq i \leq p''$ , equation 6.10 holds for  $h_i$  and  $h_{i-1}$  in place of  $\mathcal{C}_i$  and  $\mathcal{C}_{i-1}$ , respectively.
- (G')  $h_0 \equiv f_x(g_0; k, \xi_1, \dots, \xi_{p''-1}) \pmod{Q}$ .
- (H')  $g_p \equiv 0 \pmod{Q}$ .
- (I')  $h_{p''} \equiv 0 \pmod{Q}$ .

Of course, the machine  $M$  does not have direct access to any of the polynomials  $g_0, \dots, g_p, h_0, \dots, h_{p''}$ , but each of them is close to the corresponding

oracle function, so with high probability a randomly selected point in the domain hits one at which the oracle function and the polynomial agree.

**6.4.2.6.2 The Zero Polynomial Test and Equality Tests.** We next analyze the Zero polynomial Test and the Equality Tests. We analyze the effect of the Zero Polynomial Test first. Suppose that the test is called with  $s = p$  and  $\mathcal{U} = \mathcal{G}_p$ . Suppose that  $g_p$  is not the zero-polynomial. Since  $g_p$  is a polynomial of total degree  $d_p$ , by Lemma 6.32, the proportion of  $y = (y_1, \dots, y_p)$  such that  $g_p(y) \not\equiv 0 \pmod{Q}$  is at least  $1 - \frac{d_p}{Q}$ . On the other hand, since  $\mathcal{G}_p$  is  $\delta_p$ -close to  $g_p$ , the proportion of  $y = (y_1, \dots, y_p)$  such that  $g_p(y) \not\equiv \mathcal{G}_p(y) \pmod{Q}$  is at most  $\delta_p$ . Thus, the proportion of  $y = (y_1, \dots, y_p)$  such that  $\mathcal{G}_p(y) \not\equiv 0 \pmod{Q}$  is at least  $1 - \frac{d_p}{Q} - \delta_p = 1 - \frac{3p}{2^m} - \frac{1}{2(3p+2)^2}$ . This is greater than  $\frac{7}{8}$  for all  $x$  sufficiently large. Thus, we have the following fact.

**Fact 6.33** *Assuming (\*), if  $g_p$  is not zero, then  $g_p$  survives the Zero Polynomial Test with probability less than  $\frac{1}{8}$ .*

By following a similar discussion we can prove the following fact.

**Fact 6.34** *Assuming (\*), if  $h_{p''}$  is not zero, then  $h_{p''}$  survives the Zero Polynomial Test with probability less than  $\frac{1}{8}$ .*

The  $G$ -Equality Test can be analyzed similarly. Suppose that  $\hat{g}(y)(1 - \hat{g}(y)) \not\equiv g_0(y) \pmod{Q}$ . Since  $g_0$  is a polynomial of total degree at most  $d_0 = 2p$ , the proportion of  $y = (y_1, \dots, y_p) \in (\mathbb{Z}_Q)^p$  such that  $\hat{g}(y)(1 - \hat{g}(y)) \not\equiv g_0(y) \pmod{Q}$  is, by Lemma 6.32, at least  $1 - \frac{2p}{Q}$ . Since  $\hat{\mathcal{G}}$  is  $\hat{\delta}$ -close to  $\hat{g}$ , the proportion of  $y = (y_1, \dots, y_p) \in (\mathbb{Z}_Q)^p$  such that  $\hat{\mathcal{G}}(y)(1 - \hat{\mathcal{G}}(y)) \not\equiv \hat{g}(y)(1 - \hat{g}(y)) \pmod{Q}$  is at most  $\hat{\delta}$ . Since  $g_0$  is  $\delta_0$ -close to  $\mathcal{G}_0$ , the proportion of  $y = (y_1, \dots, y_p)$  such that  $\mathcal{G}_0(y) \not\equiv g_0(y) \pmod{Q}$  is at most  $\delta_0$ . So, the proportion of  $y = (y_1, \dots, y_p)$  such that  $\hat{\mathcal{G}}(y)(1 - \hat{\mathcal{G}}(y)) \not\equiv \mathcal{G}_0(y) \pmod{Q}$  is at least  $1 - \frac{2p}{Q} - \hat{\delta} - \delta_0 = 1 - \frac{2p}{Q} - \frac{1}{2(p+2)^2} - \frac{1}{2(2p+2)^2}$ . This is greater than  $\frac{7}{8}$  for all  $x$  sufficiently large.

Thus, we have the following result.

**Fact 6.35** *Assuming (\*), if  $\hat{g}(y)(1 - \hat{g}(y)) \not\equiv g_0(y) \pmod{Q}$ , then the  $G$ -Equality Test fails with probability greater than  $\frac{7}{8}$ .*

The analysis of  $\mathcal{H}_0$ - $f_x$  Equality Test is similar. Let  $\mathcal{F} = f_x(\mathcal{G}_0; \xi_1, \dots, \xi_{p''})$  and  $\mathcal{F}' = f_x(g_0; \xi_1, \dots, \xi_{p''})$ . Suppose that  $\mathcal{F}' \not\equiv h_0 \pmod{Q}$ . Since  $h_0$  is a polynomial of total degree at most  $d'_0$ , by Lemma 6.32, the proportion of  $y = (y_1, \dots, y_{p''})$  such that  $\mathcal{F}'(y) \not\equiv h_0(y) \pmod{Q}$  is at least  $1 - \frac{d'_0}{Q}$ . Since  $h_0$  is  $\delta'_0$ -close to  $\mathcal{H}_0$ , the proportion of  $y = (y_1, \dots, y_{p''})$  such that  $\mathcal{H}_0(y) \not\equiv h_0(y) \pmod{Q}$  is at most  $\delta'_0$ . To evaluate  $\mathcal{F}'$  on  $y = (y_1, \dots, y_{p''})$ , the first, second, and third blocks of  $p$  entries are given to  $\mathcal{G}_0$ , since  $g_0$  is directly accessible. Since these three blocks do not intersect with each other,

the proportion of  $y = (y_1, \dots, y_{p''})$  for which at least one of the queries to  $\mathcal{G}_0$  returns a value different from that of  $g_0$  is at most  $1 - (1 - \delta_0)^3 \leq 3\delta_0$ . So, the proportion of  $y = (y_1, \dots, y_{p''})$  such that  $\mathcal{F}(y) \not\equiv \mathcal{H}_0(y) \pmod{Q}$  is at least  $1 - \frac{d'_0}{Q} - \delta'_0 - 3\delta_0 \geq 1 - \frac{6p+2m}{2^m} - \frac{1}{2(6p+2m+2)^2} - \frac{3}{2(p+2)^2}$ . This is greater than  $\frac{7}{8}$  for all  $x$  sufficiently large. Thus, we have the following result.

**Fact 6.36** *Assuming (\*), if  $h_0 \not\equiv f_x(g_0; \xi_1, \dots, \xi_{p''}) \pmod{Q}$ , then the  $\mathcal{H}_0$ - $f_x$  Equality Test fails with probability greater than  $\frac{7}{8}$ .*

**6.4.2.6.3 The Effect of the Self-Correcting Polynomial Equality Test.** Now we analyze the Self-Correcting Polynomial Equality Test.

**Fact 6.37** *Assume (\*). Suppose that there is some  $i$ ,  $1 \leq i \leq p$ , such that equation 6.9 does not hold with  $g_i$  in place of  $\mathcal{B}_i$  and  $g_{i-1}$  in place of  $\mathcal{B}_{i-1}$ ; i.e.,*

$$g_i(\xi_1, \dots, \xi_p) \not\equiv g_{i-1}(\xi_1, \dots, \xi_{i-1}, 0, \xi_{i+1}, \dots, \xi_p) \\ + g_{i-1}(\xi_1, \dots, \xi_{i-1}, 1, \xi_{i+1}, \dots, \xi_p)\xi_i \pmod{Q}.$$

*Then the probability that  $M$  rejects  $x$  during the execution of the Self-Correcting Polynomial Equality Test with  $s = p$ ,  $d = d_{i-1}$ ,  $\mathcal{U} = \mathcal{G}_{i-1}$ , and  $\mathcal{V} = \mathcal{G}_i$  is greater than  $\frac{7}{8}$ .*

**Proof of Fact 6.37** Assume (\*). Suppose that there is some  $i$ ,  $1 \leq i \leq p$ , such that

$$g_i(\xi_1, \dots, \xi_p) \not\equiv g_{i-1}(\xi_1, \dots, \xi_{i-1}, 0, \xi_{i+1}, \dots, \xi_p) \\ + g_{i-1}(\xi_1, \dots, \xi_{i-1}, 1, \xi_{i+1}, \dots, \xi_p)\xi_i \pmod{Q}.$$

Let  $i$  be such an  $i$ . Consider the execution of the Self-Correcting Polynomial Equality Test with  $s = p$ ,  $d = d_i$ ,  $\mathcal{U} = \mathcal{G}_{i-1}$ , and  $\mathcal{V} = \mathcal{G}_i$ . Suppose that  $y_1, \dots, y_{i-1}, y_{i+1}, \dots, y_p \in \mathbb{Z}_Q$  have been fixed and  $y_i, z_0, \dots, z_d$  are yet to be picked. Let  $Y$  denote the  $(p-1)$ -tuple  $(y_1, \dots, y_{i-1}, y_{i+1}, \dots, y_p)$  and for each  $\theta \in \mathbb{Z}_Q$ , let  $Y[\theta]$  denote  $(y_1, \dots, y_{i-1}, \theta, y_{i+1}, \dots, y_p)$ . The protocol rejects  $x$  if and only if the following two conditions hold:

- $z_0, \dots, z_d$  are pairwise distinct, and
- $t(0) + t(1)y_i \not\equiv \mathcal{V}(y_1, \dots, y_p) \pmod{Q}$ ,

where  $t$  is the polynomial whose coefficients are given by  $A^{-1}(\mathcal{U}(Y[z_0]), \dots, \mathcal{U}(Y[z_d]))^T$  and  $A$  is the  $(d+1) \times (d+1)$  Vandermonde matrix such that for all integers  $j$  and  $k$ ,  $0 \leq j, k \leq d$ , the  $(j+1, k+1)$ th entry of  $A$  is  $z_j^k$ . Since  $g_{i-1}$  is a polynomial of total degree at most  $d$ , if  $z_0, \dots, z_d$  are pairwise distinct and, for all  $j$ ,  $0 \leq j \leq d$ ,  $\mathcal{U}(Y[z_j]) \equiv g_{i-1}([Y[z_j]]) \pmod{Q}$ , then for all  $\theta \in \mathbb{Z}_Q$   $t(\theta) \equiv g_{i-1}(Y[\theta]) \pmod{Q}$ . So,  $M$  rejects  $x$  if

- $z_0, \dots, z_d$  are pairwise distinct and for every  $j$ ,  $0 \leq j \leq d$ ,  $\mathcal{U}(Y[z_j]) \equiv g_{i-1}([Y[z_j]]) \pmod{Q}$ ,

- (ii)  $g_{i-1}(Y[0]) + g_{i-1}(Y[1])y_i \not\equiv g_i(y_1, \dots, y_p) \pmod{Q}$ , and  
 (iii)  $\mathcal{V}(Y[y_i]) \equiv g_i(Y[y_i]) \pmod{Q}$ .

We estimate the probability that all these conditions hold. Let  $S_1[Y]$  be the set of all  $\theta \in \mathbb{Z}_Q$  such that  $g_{i-1}(Y[\theta]) \equiv g_{i-1}(Y[\theta]) \pmod{Q}$  and  $S'_1[Y] = \mathbb{Z}_Q - S_1[Y]$ . To estimate the probability that (i) holds, first suppose that  $S_1[Y]$  has at least  $d+1$  elements. Then the probability that (i) holds is

$$\frac{||S_1[Y]|| (||S_1[Y]|| - 1) \cdots (||S_1[Y]|| - d)}{Q^{d+1}}.$$

This is at least

$$\begin{aligned} & \left( \frac{||S_1[Y]|| - d}{Q} \right)^{d+1} \\ &= \left( \frac{Q - ||S'_1[Y]|| - d}{Q} \right)^{d+1} \\ &\geq 1 - \frac{(d+1)(||S'_1[Y]|| + d)}{Q}. \end{aligned}$$

Next suppose that  $S_1[Y]$  has at most  $d$  elements. Then the probability that (i) holds is 0. Since  $||S'_1[Y]|| + ||S_1[Y]|| = Q$ ,  $1 - \frac{(d+1)(||S'_1[Y]|| + d)}{Q} < 0$ . So, regardless of the cardinality of  $S_1[Y]$ , the probability that (i) holds is at least  $1 - \frac{(d+1)(||S'_1[Y]|| + d)}{Q}$ .

On the other hand, to estimate the probability that both (ii) and (iii) hold let  $S_2[Y]$  be the set of all  $\theta \in \mathbb{Z}_Q$  such that  $g_{i-1}(Y[0]) + g_{i-1}(Y[1])y_i \not\equiv g_i(y_1, \dots, y_p) \pmod{Q}$  and let  $S'_2[Y] = \mathbb{Z}_Q - S_2[Y]$ . Also, let  $S_3[Y]$  be the set of all  $\theta \in \mathbb{Z}_Q$  such that  $\mathcal{V}(Y[y_i]) \equiv g_i(Y[y_i]) \pmod{Q}$  and let  $S'_3[Y] = \mathbb{Z}_Q - S_3[Y]$ . Then the probability that both (ii) and (iii) hold is  $\frac{||S_2[Y] \cap S_3[Y]||}{Q}$ .

Now the probability that (i), (ii), and (iii) all hold is

$$\left( 1 - \frac{(d+1)(||S'_1[Y]|| + d)}{Q} \right) \frac{||S_2[Y] \cap S_3[Y]||}{Q}.$$

This is at least

$$\frac{||S_2[Y] \cap S_3[Y]||}{Q} - (d+1) \frac{||S'_1[Y]||}{Q} - \frac{(d+1)d}{Q}.$$

Since  $S_2[Y] \cap S_3[Y] = \mathbb{Z}_Q - (S'_2[Y] \cup S'_3[Y])$ ,  $||S_2[Y] \cap S_3[Y]|| \geq Q - ||S'_2[Y]|| - ||S'_3[Y]||$ . So, the probability that (i), (ii), and (iii) hold is at least

$$1 - \frac{||S'_2[Y]||}{Q} - \frac{||S'_3[Y]||}{Q} - (d+1) \frac{||S'_1[Y]||}{Q} - \frac{(d+1)d}{Q},$$

and the probability that  $M$  rejects  $x$  is the average of this amount where  $Y$  is chosen uniformly at random.



By hypothesis,  $g_{i-1}(\xi_1, \dots, \xi_{i-1}, 0, \xi_{i+1}, \dots, \xi_p) + g_{i-1}(\xi_1, \dots, \xi_{i-1}, 1, \xi_{i+1}, \dots, \xi_p)\xi_i \not\equiv g_i(\xi_1, \dots, \xi_p) \pmod{Q}$ . Since  $g_i$  is a polynomial of total degree  $d_i = 2p + i$ , by Lemma 6.32, the average of  $\frac{\|S'_2[Y]\|}{Q}$  is at most  $\frac{2p+i}{Q}$ . Since  $\mathcal{V}$  is  $\delta_i$ -close to  $g_i$ , the average of  $\frac{\|S'_3[Y]\|}{Q}$  is at most  $\frac{1}{2(p+i+2)^2}$ . Since  $\mathcal{U}$  is  $\delta_{i-1}$ -close to  $g_{i-1}$ , the average of  $\frac{\|S'_1[Y]\|}{Q}$  is at most  $\frac{1}{2(p+i+1)^2}$ . So, the probability that  $M$  rejects is at least

$$1 - \frac{2p+i}{Q} - \frac{1}{2(2p+i+2)^2} - \frac{1}{2(2p+i+1)^2} - \frac{(d+1)d}{Q},$$

and this is greater than  $\frac{7}{8}$  for all  $x$  sufficiently large. This proves the proposition.  $\square$  Fact 6.37

By a similar analysis we can show that the following fact holds.

**Fact 6.38** *Suppose that there is some  $i$ ,  $1 \leq i \leq p$ , such that equation 6.10 does not hold with  $h_i$  in place of  $C_i$  and  $h_{i-1}$  in place of  $C_{i-1}$ ; i.e.,*

$$\begin{aligned} h_i(\xi_1, \dots, \xi_{p''}) &\not\equiv h_{i-1}(\xi_1, \dots, \xi_{i-1}, 0, \xi_{i+1}, \dots, \xi_{p''}) \\ &\quad + h_{i-1}(\xi_1, \dots, \xi_{i-1}, 1, \xi_{i+1}, \dots, \xi_{p''})\xi_i \pmod{Q}. \end{aligned}$$

*Then the probability that  $M$  rejects  $x$  during the execution of the Self-Correcting Polynomial Equality Test with  $s = p''$ ,  $d = d'_{i-1}$ ,  $\mathcal{U} = \mathcal{H}_{i-1}$ , and  $\mathcal{V} = \mathcal{H}_i$  is greater than  $\frac{7}{8}$ .*

**6.4.2.6.4 Putting the Pieces Together.** Now we put all the pieces together. Assume that there is no sampling error and that (\*) holds, i.e., each of the oracle functions is close to a polynomial of desired degree with a desired distance. Also, assume that the conditions (D') through (I') are all satisfied. Then,  $\hat{g}$  is a polynomial of total degree at most  $p$  such that, for all  $(\xi_1, \dots, \xi_p) \in \{0, 1\}^p$ ,  $\hat{g}(\xi_1, \dots, \xi_p) \pmod{Q} \in \{0, 1\}$ , and such that, for all  $(\xi_1, \dots, \xi_{p''}) \in \{0, 1\}^p$ ,  $f_x(\hat{g}; \xi_1, \dots, \xi_{p''}) \equiv 0 \pmod{Q}$ . This implies that  $x \in L$ , a contradiction. So, either (\*) does not hold or at least one of (D') through (I') fails to hold, and hence,  $M$  rejects  $x$  with probability at least  $\frac{3}{4}$ . This proves the soundness of the protocol.

Now the only remaining task is to prove the Low-Degree Polynomial Characterization Lemma (Lemma 6.28) and the Low-Degree Polynomial Closeness Lemma (Lemma 6.31).

### 6.4.3 Proof of the Low-Degree Polynomial Characterization Lemma (Lemma 6.28)

This lemma states the following: Let  $d$  and  $s$  be positive integers and  $R$  be a prime number. Let  $h$  be a mapping from  $(\mathbb{Z}_R)^s$  to  $\mathbb{Z}_R$ . Then  $h$  is a polynomial of total degree at most  $d$  if and only if

$$(\forall y, z \in F^s) \left[ \sum_{0 \leq i \leq d+1} \gamma_i h(y + iz) \equiv 0 \pmod{R} \right],$$

where for every  $i$ ,  $0 \leq i \leq d+1$ ,  $\gamma_i = \binom{d+1}{i} (-1)^i$ . The lemma can be proven by simply combining two propositions.

**Proposition 6.39** *Let  $d$  and  $s$  be positive integers. Let  $R$  be a prime number. Let  $h$  be a mapping from  $(\mathbb{Z}_R)^s$  to  $\mathbb{Z}_R$ . Then  $h$  is a multivariate polynomial over  $\mathbb{Z}_R$  of degree at most  $d$  if and only if for all  $y, z \in (\mathbb{Z}_R)^s$  the function  $h'_{y,z}(i) = h(y + iz)$  is a polynomial in  $i$  of degree at most  $d$  over  $\mathbb{Z}_R$ .*

**Proof of Proposition 6.39** Let  $d$ ,  $s$ ,  $R$ , and  $h$  be as in the hypothesis of the proposition. We first show that  $h$  is a polynomial of degree at most  $d$ . We first show that  $h$  is a polynomial. For each  $y = (y_1, \dots, y_s) \in (\mathbb{Z}_R)^s$ , let  $Q_y(\xi_1, \dots, \xi_s)$  be the polynomial

$$\prod_{1 \leq i \leq s} \prod_{z \in \mathbb{Z}_R \setminus \{y_i\}} (\xi_i - z)(y_i - z)^{-1}.$$

Then  $Q_y$  is an  $s$ -variate polynomial and for every  $a = (a_1, \dots, a_s) \in (\mathbb{Z}_R)^s$ ,  $Q_y(a) \equiv 1 \pmod{R}$  if  $a = y$  and  $Q_y(a) \equiv 0 \pmod{R}$  otherwise. Define

$$\hat{h}(\xi_1, \dots, \xi_s) = \sum_{y \in (\mathbb{Z}_R)^s} Q_y(\xi_1, \dots, \xi_s) h(\xi_1, \dots, \xi_s).$$

Then  $\hat{h}$  is an  $s$ -variate polynomial and  $\hat{h} \equiv h \pmod{R}$ . Thus,  $h$  is a polynomial.

Now what we need to show is that the degree of  $h$  is at most  $d$  if and only if for all  $y$  and  $z$ ,  $y, z \in (\mathbb{Z}_R)^s$ ,  $h'_{y,z}(i)$  is a polynomial in  $i$  of degree at most  $d$ . Suppose that  $h$  has total degree at most  $d$ . Then  $h$  can be expressed as the sum of monomials, each of the form  $c \xi_{i_1}^{e_1} \cdots \xi_{i_m}^{e_m}$  such that  $c \in \mathbb{Z}_R \setminus \{0\}$ ,  $1 \leq i_1 < \cdots < i_m \leq s$ ,  $e_1, \dots, e_m \geq 1$ , and  $e_1 + \cdots + e_m \leq d$ . Let  $t$  be such a monomial. Let  $y$  and  $z$  be arbitrary elements of  $(\mathbb{Z}_R)^s$ . Then,  $t(y + iz)$  is a polynomial in  $i$  of degree  $e_1 + \cdots + e_m \leq d$ . Thus, the degree of  $h'_{y,z}(i)$  is at most  $d$ . This proves the direction from left to right.

To prove the other direction, suppose that for all  $y, z \in (\mathbb{Z}_R)^s$ , the function  $h'_{y,z}(i) = h(y + iz)$  is a polynomial in  $i$  of degree at most  $d$ . Assume that the total degree of  $h$  is some  $k > d$ . Divide  $h$  into  $u_1$  and  $u_2$ , where  $u_1$  consists of all the monomials of  $h$  having degree exactly  $k$  and  $u_2$  consists of all the monomials of  $h$  having degree less than  $k$ . Take  $y$  to be  $(0, \dots, 0)$ . Then for all  $z \in (\mathbb{Z}_R)^s$ ,  $u_1(iz) = u_1(z)i^k$  and  $u_2(iz)$  is a polynomial in  $i$  having degree at most  $k-1$ . By our assumption, for all  $z \in (\mathbb{Z}_R)^s$ ,  $u(iz)$  is a polynomial in  $i$  of degree at most  $d$ . Since  $k > d$ , this implies that, for all  $z \in (\mathbb{Z}_R)^s$ ,  $u_1(z) \equiv 0 \pmod{R}$ . Thus,  $h \equiv u_2 \pmod{R}$ , and thus, the degree of  $h$  is less than  $k$ , a contradiction. Hence,  $h$  is a polynomial having degree at most  $d$ . This completes the proof of the proposition.  $\square$  Proposition 6.39

**Proposition 6.40** *Let  $d$  be a positive integer, let  $R \geq d + 1$  be a prime number, and  $h$  be a mapping from  $\mathbb{Z}_R$  to  $\mathbb{Z}_R$ . Then  $h$  is a polynomial of degree at most  $d$  if and only if for every  $y, z \in \mathbb{Z}_R$ ,*

$$\sum_{0 \leq i \leq d+1} \gamma_i h(y + iz) \equiv 0 \pmod{R},$$

where for every  $i$ ,  $0 \leq i \leq d + 1$ ,  $\gamma_i = \binom{d+1}{i}(-1)^i$ .

**Proof of Proposition 6.40** Let  $d$ ,  $R$ , and  $h$  be as in the hypothesis of the proposition. Suppose that  $h$  is a polynomial of degree at most  $d$ . Since two polynomials of degree  $\leq d$  can agree on at most  $d$  points, specifying the value of  $h$  at  $d + 1$  distinct points will give a unique specification of  $h$ . Let  $a_1, \dots, a_{d+1}$  be distinct elements of  $R$ . Then,  $h$  can be interpolated from the values of  $h$  at  $a_1, \dots, a_{d+1}$ , i.e.,

$$h(\xi) = \sum_{1 \leq i \leq d+1} h(a_i) \left( \prod_{j \in \{1, \dots, d+1\} \setminus \{i\}} (\xi - a_j) \right) \left( \prod_{j \in \{1, \dots, d+1\} \setminus \{i\}} (a_i - a_j) \right)^{-1}$$

holds. Let  $y, z$  be arbitrary elements of  $\mathbb{Z}_R$ . If  $z = 0$ , then  $\sum_{0 \leq i \leq d+1} \gamma_i h(y + iz) = h(y) \sum_{0 \leq i \leq d+1} \gamma_i$ . Note that, for all  $u$ ,  $\sum_{0 \leq i \leq d+1} \gamma_i u^i = (1 - u)^{d+1}$ . So,  $\sum_{0 \leq i \leq d+1} \gamma_i = (1 - 1)^{d+1} = 0$ . Thus, if  $z = 0$ , then  $\sum_{0 \leq i \leq d+1} \gamma_i h(y + iz) = 0 \equiv 0 \pmod{R}$  as desired. So, suppose that  $z \neq 0$ . Let  $\xi = y$  and for each  $i$ ,  $0 \leq i \leq d + 1$ , let  $a_i = y + iz$ . Then, since  $z \neq 0$  and  $d + 1 \leq R$ ,  $a_1, \dots, a_{d+1}$  are pairwise distinct. So, we apply the above formula. For every  $i$ ,  $1 \leq i \leq d + 1$ ,

$$\prod_{j \in \{1, \dots, d+1\} \setminus \{i\}} (\xi - a_j) = (-1)^d z^d \frac{(d+1)!}{i}$$

and

$$\prod_{j \in \{1, \dots, d+1\} \setminus \{i\}} (a_i - a_j) = (-1)^{d+1+i} (i-1)! (d+1-i)! z^d.$$

So,

$$h(y) = \sum_{1 \leq i \leq d+1} \binom{d+1}{i} (-1)^{i+1} h(y + iz).$$

Since for every  $i$ ,  $0 \leq i \leq d + 1$ ,  $\gamma_i = \binom{d+1}{i}(-1)^i$ , we have

$$\sum_{0 \leq i \leq d+1} \gamma_i h(y + iz) \equiv 0 \pmod{R}$$

as desired.

On the other hand, suppose that for all  $y, z \in \mathbb{Z}_R$ , it holds that  $\sum_{0 \leq i \leq d+1} \gamma_i h(y + iz) \equiv 0 \pmod{R}$ . For each  $n \geq 0$ , let  $a_n = h(n \bmod R)$ . It suffices to show that there is a degree- $d$  polynomial  $q(n)$  such that

$$(\forall n \geq 0)[a_n \equiv q(n) \pmod{R}].$$

For each  $k$ ,  $1 \leq k \leq d$ , and each  $n \geq 0$ , let

$$s_{k,n} = \sum_{0 \leq j \leq k} \alpha_{k,j} a_{n+j}.$$

Then we have the following fact.

**Fact 6.41** *For each  $k$ ,  $0 \leq k \leq d$ , there exists a polynomial  $q_k$  of degree  $d - k$  such that, for all  $n \geq 0$ ,  $s_{k,n} \equiv q_k(n) \pmod{R}$ .*

**Proof of Fact 6.41** The proof is by induction on  $k$ , going down from  $k = d$  to  $k = 0$ . For each  $k \geq 0$  and each  $j$ ,  $0 \leq j \leq k$ , let  $\alpha_{k,j} = (-1)^j \binom{k}{j}$ . For the base case, let  $k = d$ . We are assuming that for all  $y, z \in \mathbb{Z}_R$ ,  $\sum_{0 \leq j \leq d+1} \gamma_j h(y + iz) \equiv 0 \pmod{R}$ . For all  $j$ ,  $0 \leq j \leq d+1$ ,  $\alpha_{d+1,j} = \gamma_j$ . So, it holds that  $\sum_{0 \leq j \leq d+1} \alpha_{d+1,j} a_{y+jz} \equiv 0 \pmod{R}$ . Replace  $y$  by  $n$  and take  $z$  to be 1. Then, for all  $n \geq 0$ ,

$$\sum_{0 \leq j \leq d+1} \alpha_{d+1,j} a_{n+j} \equiv 0 \pmod{R}. \quad (6.11)$$

Note that for all  $m \geq 1$ ,  $\binom{m}{m} = \binom{m}{0} = 1$  and for all  $i$ ,  $0 \leq i \leq m-1$ ,  $\binom{m}{i} + \binom{m}{i+1} = \binom{m+1}{i+1}$ . So, equation 6.11 can be rewritten as

$$\left( \sum_{0 \leq j \leq d} \alpha_{d,j} a_{n+j} \right) - \left( \sum_{0 \leq j \leq d} \alpha_{d,j} a_{(n+1)+j} \right) \equiv 0 \pmod{R}.$$

By definition, the first term is  $s_{d,n}$  and the second term is  $s_{d,n+1}$ . So, we have  $s_{d,n} - s_{d,n+1} \equiv 0 \pmod{R}$ . Let  $q_d(n)$  be the constant polynomial  $s_{d,0}$ , which is equal to  $\sum_{0 \leq j \leq d} \alpha_{d,j} a_j$ . Then, for all  $n \geq 0$ ,  $s_{d,n} \equiv q_d(n) \pmod{R}$ . Thus, the claim holds for  $k = d$ .

For the induction step, let  $k = k_0$ ,  $0 \leq k_0 < d$  and suppose that the claim holds for all values of  $k$  greater than  $k_0$  and less than or equal to  $d$ . In particular, since the claim holds for  $k = k_0 + 1$ , for all  $n \geq 0$ ,

$$\sum_{0 \leq j \leq k_0+1} \alpha_{k_0+1,j} a_{n+j} \equiv q_{k_0+1}(n) \pmod{R},$$

where  $q_{k_0+1}$  is a polynomial of degree  $d - k_0 - 1$ . As in the previous paragraph, the sum can be rewritten as

$$\left( \sum_{0 \leq j \leq k_0} \alpha_{k_0, j} a_{n+j} \right) - \left( \sum_{0 \leq j \leq k_0} \alpha_{k_0, j} a_{(n+1)+j} \right).$$

So, for all  $n \geq 0$ ,  $s_{k_0, n} - s_{k_0, n+1} \equiv q_{k_0+1}(n) \pmod{R}$ . Let  $\beta_{k_0} = s_{k_0, 0} = \sum_{0 \leq j \leq k_0} \alpha_{k_0, j} a_j$ . Then, for all  $n \geq 1$ ,

$$s_{k_0, n} \equiv \beta_{k_0} - \sum_{1 \leq j \leq n} q_{k_0+1}(j) \pmod{R}.$$

Since  $q_{k_0+1}$  is a polynomial of degree  $d - k_0 - 1$ , the summation on the right hand side is a polynomial of degree  $d - k_0$ . To see why, let  $\ell \geq 0$  be an integer. Let us suppose that the sum  $\sum_{1 \leq j \leq n} j^\ell$  is a degree- $(\ell + 1)$  polynomial of the form  $c_0 + c_1 n + \dots + c_{\ell+1} n^{\ell+1}$ . Since  $\sum_{1 \leq j \leq n} j^\ell = 0$ ,  $c_0$ . The condition that these coefficients have to satisfy is

$$(c_1 n + \dots + c_{\ell+1} n^{\ell+1}) \equiv (c_1(n-1) + \dots + c_{\ell+1}(n-1)^{\ell+1}) + n^\ell \pmod{R}.$$

By rearranging terms, this condition is equivalent to

$$c_1(n - (n-1)) + \dots + c_{\ell+1}(n^{\ell+1} - (n-1)^{\ell+1}) + n^\ell \equiv 0 \pmod{R}.$$

Note that for every  $m \geq 1$ ,  $n^m - (n-1)^m = \sum_{0 \leq j \leq m-1} \alpha_{m, j} n^j$ . So, the above condition can be written as

$$\begin{pmatrix} -\alpha_{\ell+1, \ell} & 0 & 0 & 0 \\ -\alpha_{\ell+1, \ell-1} & -\alpha_{\ell, \ell-1} & 0 & 0 \\ -\alpha_{\ell+1, \ell-2} & -\alpha_{\ell, \ell-2} & -\alpha_{\ell-1, \ell-2} & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ -\alpha_{\ell+1, 0} & -\alpha_{\ell, 0} & -\alpha_{\ell-1, 0} & \dots & -\alpha_{1, 0} \end{pmatrix} \begin{pmatrix} c_{\ell+1} \\ c_\ell \\ c_{\ell-1} \\ \vdots \\ c_1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

where the arithmetic is over  $\mathbb{Z}_R$ . The  $(\ell + 1) \times (\ell + 1)$ -matrix is lower triangular and none of its diagonal entries are zero, so its inverse exists. Thus,  $c_1, \dots, c_{\ell+1}$  can be uniquely determined in  $\mathbb{Z}_R$ . Thus, the sum of the right-hand side is a polynomial of degree  $d - k$ . Thus the claim holds for this  $k$ , too. Hence, the fact holds.  $\square$  Fact 6.41

Now note that  $s_{0, n} = a_n$ . Since  $s_{0, n}$  is a polynomial of degree  $d$ ,  $h$  is a polynomial of degree  $d$ . This proves the proposition.  $\square$  Proposition 6.40

This proves Lemma 6.28.  $\square$

#### 6.4.4 Proof of the Low-Degree Polynomial Closeness Lemma (Lemma 6.31)

This lemma states the following: Let  $s$  and  $d$  be positive integers and let  $\delta$  be a real number such that  $\delta \leq \frac{1}{2(d+2)^2}$ . Let  $f$  be a mapping from  $(\mathbb{Z}_Q)^s$  to

$\mathbb{Z}_Q$  that is not  $2\delta$ -close to any polynomial of total degree at most  $d$ . Then  $f$  survives the Low-Degree Test with probability less than  $\frac{1}{4}$ .

The lemma follows from Theorem 6.42 below.

**Theorem 6.42** *Let  $d$  and  $s$  be positive integers. Let  $R \geq d + 2$  be a prime number. Let  $C$  be a mapping from  $(\mathbb{Z}_R)^s$  to  $\mathbb{Z}_R$ . Let  $\epsilon_0 = \frac{1}{2(d+2)^2}$ . Let  $\epsilon$  be the probability that*

$$\sum_{0 \leq i \leq d+1} \gamma_i C(y + iz) \not\equiv 0 \pmod{R},$$

*where  $y$  and  $z$  are chosen independently and uniformly at random from  $(\mathbb{Z}_R)^s$ . Suppose that  $\epsilon \leq \epsilon_0$ . Then  $C$  is  $2\epsilon$ -close to a polynomial of total degree at most  $d$  over  $\mathbb{Z}_R$ .*

**Proof of Lemma 6.31** Let  $R = Q$ ,  $d = p$ ,  $s = p$ ,  $C = \mathcal{B} \bmod \mathbb{Z}_R$ , and  $\epsilon_0 = \delta$ . Suppose that  $\mathcal{B} \bmod R$  is  $2\delta$ -close to no polynomial of total degree at most  $p$ . By taking the contrapositive of the statement of the theorem, we have that  $\epsilon > \epsilon_0 = \frac{1}{2(p+2)^2}$ . Since the equivalence is tested  $6(p+2)^2$  times, the probability that the Low-Degree Test succeeds is at most

$$\left(1 - \frac{1}{2(p+2)^2}\right)^{6(p+2)^2} < 2^{-3} = \frac{1}{8}.$$

This proves the lemma.  $\square$

Lemma 6.31

Now let us turn to proving Theorem 6.42. Let  $d$ ,  $s$ ,  $R$ ,  $C$ ,  $\epsilon$ , and  $\epsilon_0$  be as in the statement of the lemma and suppose that  $\epsilon \leq \epsilon_0$ . For each  $y \in (\mathbb{Z}_R)^s$ , define  $h(y)$  to be the most frequently occurring value in the multiset  $\{(\sum_{1 \leq i \leq d+1} \gamma_i C(y + iz)) \bmod R \mid z \in (\mathbb{Z}_R)^s\}$ , where ties are broken arbitrarily.

**Fact 6.43** *If  $y$  is chosen uniformly at random from  $(\mathbb{Z}_R)^s$ , then  $h(y) \equiv C(y) \bmod R$  with probability at least  $1 - 2\epsilon$ .*

**Proof** Let  $W = \{y \in (\mathbb{Z}_R)^s \mid h(y) \equiv C(y) \bmod R\}$ . Let  $\rho = |W|/R^s$ . Since ties can be broken arbitrarily to determine the value of  $h$ , for every  $y \in W$ , for at least half of  $z \in (\mathbb{Z}_R)^s$  it holds that  $C(y) \not\equiv \sum_{1 \leq i \leq d+1} \gamma_i C(y + iz) \pmod{R}$ . Then  $\epsilon$ , the proportion of  $(y, z) \in (\mathbb{Z}_R)^s \times (\mathbb{Z}_R)^s$  such that  $C(y) \not\equiv \sum_{1 \leq i \leq d+1} \gamma_i C(y + iz) \pmod{R}$ , is at least  $\rho/2$ . Thus,  $\rho \geq 2\epsilon$ . This proves the fact.  $\square$

Fact 6.43

**Fact 6.44** *For all  $y \in (\mathbb{Z}_R)^s$ , if  $z$  is chosen uniformly at random from  $(\mathbb{Z}_R)^s$ , then the probability that  $h(y) \equiv \sum_{1 \leq i \leq d+1} \gamma_i C(y + iz) \pmod{R}$  is at least  $1 - 2(d+1)\epsilon$ .*

**Proof** Let  $y \in (\mathbb{Z}_R)^s$  be fixed. Let  $i$  be any integer between 1 and  $d+1$ . Suppose that we select  $z \in (\mathbb{Z}_R)^s$  uniformly at random and output  $u = y + iz \bmod R$ . Since  $R \geq d+2$ , a multiplicative inverse of  $i$  in  $\mathbb{Z}_R$  exists, so,  $u$  is uniformly distributed over  $(\mathbb{Z}_R)^s$ . So, the probability that

$$\mathcal{C}(y + iz) \equiv \sum_{1 \leq j \leq d+1} \gamma_j \mathcal{C}(y + iz + jw) \pmod{R} \quad (6.12)$$

when  $y$  and  $z$  are chosen independently and uniformly at random from  $(\mathbb{Z}_R)^s$  is equal to the probability that

$$\mathcal{C}(u) \equiv \sum_{1 \leq j \leq d+1} \gamma_j \mathcal{C}(u + jv) \pmod{R} \quad (6.13)$$

when  $u$  and  $v$  are chosen independently and uniformly at random from  $(\mathbb{Z}_R)^s$ . This probability is equal to  $1 - \epsilon$  by our assumption. By exchanging the role of  $i$  and  $j$  as well as the role of  $z$  and  $w$ , we have that, for every  $j$ ,  $1 \leq j \leq d+1$ , the probability that

$$\mathcal{C}(y + jw) \equiv \sum_{1 \leq i \leq d+1} \gamma_i \mathcal{C}(y + iz + jw) \pmod{R}$$

when  $z$  and  $w$  are chosen independently and uniformly at random from  $(\mathbb{Z}_R)^s$  is  $1 - \epsilon$ . Let  $E_1[z, w]$  be the event

$$\sum_{1 \leq i \leq d+1} \sum_{1 \leq j \leq d+1} \gamma_i \gamma_j \mathcal{C}(y + iz + jw) = \sum_{1 \leq i \leq d+1} \gamma_i \mathcal{C}(y + iz),$$

$E_2[z, w]$  be the event

$$\sum_{1 \leq i \leq d+1} \sum_{1 \leq j \leq d+1} \gamma_i \gamma_j \mathcal{C}(y + iz + jw) = \sum_{1 \leq j \leq d+1} \gamma_j \mathcal{C}(y + jw),$$

and

$$E_0[z, w] = E_1[z, w] \wedge E_2[z, w].$$

Note that for all  $y, z \in (\mathbb{Z}_R)^s$  the following conditions hold:

- If for all  $i$ ,  $1 \leq i \leq d+1$ , equation 6.12 holds, then  $E_1[z, w]$  holds.
- If for all  $j$ ,  $1 \leq j \leq d+1$ , equation 6.13 holds, then  $E_2[z, w]$  holds.

Since both  $i$  and  $j$  range from 1 to  $d+1$ , if  $z$  and  $w$  are chosen independently and uniformly at random from  $(\mathbb{Z}_R)^s$ , then  $E_0[z, w]$  holds with probability at least  $1 - 2(d+1)\epsilon$ .

Let  $v_1, \dots, v_R$  be an enumeration of all the members of  $\mathbb{Z}_R$ . For each  $k$ ,  $1 \leq k \leq R$ , let  $\rho_k$  be the probability that  $v_k \equiv \sum_{1 \leq i \leq d+1} \gamma_i \mathcal{C}(y + iz) \pmod{Q}$  when  $z \in (\mathbb{Z}_R)^s$  is chosen uniformly at random. Assume, without loss of generality, that  $\rho_1 = \max\{\rho_1, \dots, \rho_k\}$ . Then  $E_0[z, w]$  occurs with probability at most  $\rho_1^2 + \dots + \rho_R^2 \leq \rho_1(\rho_1 + \dots + \rho_R) = \rho_1$ . Since the event  $E_0[z, w]$  occurs with probability at least  $1 - 2(d+1)\epsilon$ , we have  $\rho_1 \geq 1 - 2(d+1)\epsilon$ . Thus, the probability that  $\sum_{1 \leq i \leq d+1} \gamma_i \mathcal{C}(y + iz) \pmod{R}$  takes the most frequently occurring value, which is  $h(y)$ , is at least  $1 - 2(d+1)\epsilon$ . This proves the fact.  $\square$  Fact 6.44

**Fact 6.45** *If  $\epsilon \leq \epsilon_0$ , then for all  $y, z \in (\mathbb{Z}_R)^s$ ,*

$$\sum_{0 \leq i \leq d+1} \gamma_i h(y + iz) \equiv 0 \pmod{R}.$$

**Proof** Let  $u, v \in (\mathbb{Z}_R)^s$  be fixed. For every  $i$ ,  $0 \leq i \leq d+1$ , and every  $w \in (\mathbb{Z}_R)^s$ , the equation  $u + iv \equiv w \pmod{R}$  has precisely  $R^s$  solutions. So, for every  $i$ ,  $0 \leq i \leq d+1$ ,  $u + iv \pmod{R}$  is subject to the uniform distribution over  $(\mathbb{Z}_R)^s$  when  $u$  and  $v$  are chosen independently and uniformly at random from  $(\mathbb{Z}_R)^s$ . So, by Fact 6.44, for every  $i$ ,  $0 \leq i \leq d+1$ , the probability that

$$h(y + iz) \equiv \sum_{1 \leq j \leq d+1} \gamma_j \mathcal{C}((y + iz) + j(u + iv)) \pmod{R}$$

when  $u$  and  $v$  are chosen independently and uniformly at random from  $(\mathbb{Z}_R)^s$  is at least  $1 - 2(d+1)\epsilon$ . So, the event

$$\begin{aligned} \sum_{0 \leq i \leq d+1} \gamma_i h(y + iz) \equiv \\ \sum_{1 \leq j \leq d+1} \gamma_j \sum_{0 \leq i \leq d+1} \gamma_i \mathcal{C}((y + iz) + j(u + iv)) \pmod{R} \end{aligned}$$

has probability at least  $1 - 2(d+1)(d+2)\epsilon$  when  $u$  and  $v$  are chosen uniformly at random from  $(\mathbb{Z}_R)^s$ . By rearranging terms  $(y + iz) + j(u + iv)$  is equal to  $(y + ju) + i(z + jv)$ . If  $u$  is chosen uniformly at random from  $(\mathbb{Z}_R)^s$ , then  $y + ju \pmod{R}$  is uniformly distributed over  $(\mathbb{Z}_R)^s$ . Also, if  $v$  is chosen uniformly at random from  $(\mathbb{Z}_R)^s$ , then  $z + jv \pmod{R}$  is uniformly distributed over  $(\mathbb{Z}_R)^s$ . So, the probability that

$$\sum_{0 \leq i \leq d+1} \gamma_i \mathcal{C}((y + iz) + j(u + iv)) \equiv 0 \pmod{R}$$

is  $1 - \epsilon$  if  $u$  and  $v$  are chosen independently and uniformly at random from  $(\mathbb{Z}_R)^s$ . By combining the two observations, the probability that

$$\begin{aligned} \sum_{0 \leq i \leq d+1} \gamma_i h(y + iz) \equiv \\ \sum_{1 \leq j \leq d+1} \gamma_j \sum_{0 \leq i \leq d+1} \gamma_i \mathcal{C}((y + iz) + j(u + iv)) \equiv 0 \pmod{R} \end{aligned}$$

is positive. Since the event  $\sum_{0 \leq i \leq d+1} \gamma_i h(y + iz) = 0 \pmod{R}$  is independent of  $y$  and  $z$ ,  $\sum_{0 \leq i \leq d+1} \gamma_i h(y + iz) = 0 \pmod{R}$  holds.  $\square$  Fact 6.45

Combining Fact 6.45 and Lemma 6.28, we have the following.

**Fact 6.46** *If  $\epsilon \leq \epsilon_0$ , then  $h$  is a polynomial of total degree at most  $d$ .*

Now the theorem follows by combining Facts 6.43 and 6.46. This concludes the proof of Theorem 6.24.  $\square$



## 6.5 OPEN ISSUE: The Power of the Provers

What computational power must the provers possess to convince the verifier of membership? Following a discussion similar to that of Sect. 6.2, one can show that  $P^{\#P}$ -machines can serve as provers for  $P^{\#P}$ . Also, the proof of  $IP = PSPACE$  gives that  $PSPACE$ -provers are sufficient and necessary for  $PSPACE$ . Then how about  $NEXP$ ? Since the oracle has only to fix a satisfying assignment and the largest satisfying assignment in the lexicographic order of a formula with exponentially many variables can be computed in exponential time with an  $NP$  language as the oracle, the provers need only the computational power of  $EXP^{NP}$ . Note that for  $EXP$ , an  $EXP$  prover suffices (because an  $EXP$  machine can be viewed as a special  $NEXP$  machine which uses no nondeterminism). Can we lower the upper bound of  $EXP^{NP}$ ?

**Open Question 6.47** *Can we show a stronger upper bound on the power of the provers for  $NEXP$ ?*

## 6.6 Bibliographic Notes

Part 1 of Proposition 6.3 is due to Zankò [Zan91]. Theorem 6.4 is due to Lund et al. [LFKN92]. Definition 6.6, the notion of an enumerator (also known as an enumerative approximator) is due to Cai and Hemachandra [CH89]. Theorem 6.7 is due to Cai and Hemachandra ([CH91], see also [CH89]) and, independently, Amir, Beigel, and Gasarch [ABG00]. Lemma 6.10 is from Goldwasser and Sipser [GS89]. Theorems 6.21 and 6.23 are due to Fortnow, Rompel, and Sipser [FRS94]. Lemma 6.11 is due to Shamir [Sha92]. Our presentation of the proof of Lemma 6.11 is based on the idea of Hartmanis [Har91]. Savitch's Theorem, which forms the basis of the proof of Lemma 6.11, is due to Savitch [Sav70]. The tableau method is due to Cook [Coo71]. Theorem 6.8 follows as a corollary to Chebyshev's Theorem, which states that the number of primes not greater than  $x$  is  $x/\ln x$  (see [HW79, Theorem 7]). Theorem 6.16 is due to Pratt [Pra75]. Theorem 6.19 is due to Babai, Fortnow, and Lund [BFL91]. The low-degree test we used here was established by Rubinfeld and Sudan [RS96]. Lemmas 6.28 and 6.31 are due to them. The Self-Correcting Polynomial Equality Test is due to Sudan [Sud92]. Lemma 6.32 is due to DeMillo and Lipton [DL78], Schwartz [Sch80], and Zippel [Zip79]. Proposition 6.29 can be found in many linear algebra textbooks (See Lang [Lan87], for example). The interpolation formula used in the proof of Proposition 6.40 is called the Lagrange Interpolation Formula and can be found in such algebra textbooks, such as Van der Waerden's [vdW70]. Chebyshev's Inequality (Lemma 6.22) can be found in probability textbooks (see [Fel68] for example).

The idea of arithmetization first appeared in a paper by Beaver and Feigenbaum [BF90]. In some sense arithmetization is a very sophisticated

version of program checking by Blum and Kannan [BK95]. The interactive proof for  $P^{\#P}$  combines the self-testing procedure of Lipton [Lip91] and the downward self-reducibility by Blum, Luby, and Rubinfeld [BLR93].

To construct an oracle protocol for NEXP, Babai, Fortnow, and Lund [BFL91] stipulated that the polynomial held in the oracle is multilinear and developed a probabilistic oracle protocol for testing multilinearity. It is natural to ask whether the requirement that the polynomial should be multilinear be weakened so that only having a small degree is required. This question is studied in [BFLS91, FGL<sup>+</sup>96]. The goal of the Low-Degree Test in the proof of  $MIP = NEXP$  is to ensure that a given function is close to a low-degree polynomial. Once this has been done for all the functions involved, the other two tests can be carried out by simply assuming that the functions are all polynomials. The concept of hypothesizing that a given function is a low-degree polynomial, called *self-testing*, was introduced by Gemmell et al. [GLR<sup>+</sup>91] and was further explored by Rubinfeld and Sudan [RS96, RS96]. The problem of computing a value of function knowing that there is an oracle that is close to the function is called *self-correction*. Self-correction borrows an idea from random self-reducibility of Abadi, Feigenbaum and Kilian [AFK89] and was first formally studied by Blum and Kannan [BK95].

We note that the progress from Theorem 6.4 toward Theorem 6.23 was made in only five weeks. Email announcements of  $PH \subseteq IP$  by Fortnow,  $IP = PSPACE$  by Shamir, and  $MIP = NEXP$  by Fortnow again came out respectively on December 13, 1989, December 26, 1989, January 17, 1990. (For a detailed history, see an amusing survey by Babai [Bab90].)

The polynomial interpolation technique was received with great excitement and invigorated research on interactive proof systems. Babai and Fortnow [BF91] show a new characterization of  $\#P$  by straight-line programs, Cai, Condon, and Lipton [CCL94] show that every language in PSPACE has a bounded-round multiprover interactive proof systems, Lapidot and Shamir [LS97] show that a fully parallelized version of the protocol by Ben-Or and others [BOGKW88] yields a one-round “perfect zero-knowledge” protocol for each language in NEXP, and Feige and Lovász [FL92] show that two-prover one-round interactive proof systems exist for all languages in NEXP. We noted in Sect. 6.5 that to construct a multiprover protocol for a EXP language a prover in EXP is sufficient. In other words, the oracle of a probabilistic oracle protocol for EXP languages can be in EXP. Based on this observation, Babai et al. [BFNW93] show that if  $EXP \subseteq P/poly$  then EXP is included in MA, a class introduced by Babai [Bab85]. Note that one can prove  $EXP \subseteq P/poly \implies EXP = S_2^P$  by applying the proof of Theorem 1.16 to EXP in light of Sengupta’s observation (see the Bibliographic Notes of Chap. 1). However, the collapse shown by Babai et al. seems stronger since MA is known to be included in  $S_2^P$  [RS98].

The  $MIP = NEXP$  Theorem naturally raises the issue of translating the theorem to characterizations of NP. Feige et al. [FGL<sup>+</sup>96] and Babai et

al. [BFLS91] independently obtained two similar but incomparable results. Roughly speaking these two papers show that every language in NP has a probabilistic polynomial-time protocol for the verifier that uses *polylogarithmic* random coin tosses and communicates with its prover *polylogarithmic* bits of information. In addition to the “scaled-down” theorem, the former paper shows the following: If there exists a deterministic polynomial-time algorithm that approximates the size of the largest clique in a graph within a constant factor, then  $\text{NP} \subseteq \text{DTIME}[2^{\log n \log \log n}]$ . This was a remarkable achievement, because for decades researchers had been looking for results to shed light to the question of whether polynomial time approximation of the largest clique size within *any* factor between 2 and  $\frac{n}{\log^3 n}$  is possible. For the first time, strong evidence is given that approximation of the clique size within a constant factor is not possible under some reasonable assumption about NP. Feige et al. also show that if there exist constants  $\epsilon$ ,  $0 < \epsilon < 1$ , and  $d > 0$ , such that one can approximate the clique size within a factor of  $2^{\log^{1-\epsilon} n}$  using an algorithm that runs in time  $2^{\log^d n}$ , then  $\text{NP} \subseteq \bigcup_{k>0} \text{DTIME}[n^{\log^k n}]$ .

To describe the two results about NP, let  $\text{PCP}(r(n), q(n))$  (see [AS98]) denote the class of all languages for which there exists a probabilistic polynomial time oracle protocol with the following three properties: (i) the verifier flips  $r(n)$  coins and examines  $q(n)$  bits of the oracle on an input of length  $n$ , (ii) if the input belongs to the language, then there exists an oracle relative to which the verifier accepts with probability 1, and (iii) if the input does not belong to the language, then there is no oracle relative to which the verifier accepts with probability at least  $\frac{1}{2}$ . With this notation the  $\text{MIP} = \text{NEXP}$  Theorem can be restated as  $\text{NEXP} = \bigcup_{c>0} \text{PCP}(n^c, n^c)$ , the above result about NP by Feige et al. as  $\text{NP} \subseteq \bigcup_{c>0} \text{PCP}(c \log n \log \log n, c \log n \log \log n)$  and the one by Babai et al. as  $\text{NP} \subseteq \bigcup_{c>0} \text{PCP}(\log^c n, \log^c n)$ .

The two results about NP raised the question whether the polylogarithmic number of random bits and the communication bits are truly necessary. Arora and Safra [AS98] made significant progress towards that question and showed that  $\text{NP} \subseteq \text{PCP}(\mathcal{O}(\log n), \mathcal{O}(\sqrt{\log n}))$ . To prove this result, Arora and Safra proposed a technique of composing verifiers—verifying computation of a verifier by another verifier. Improving this technique further, Arora et al. [ALM<sup>+</sup>98] reduced the second amount to a constant, and obtained the so-called *PCP Theorem*:  $\text{NP} = \text{PCP}(\mathcal{O}(\log n), \mathcal{O}(1))$ . The PCP Theorem states that every language in NP has a probabilistic oracle protocol such that (i) the prover provides a proof of polynomial length, (ii) the verifier tosses  $\mathcal{O}(\log n)$  coins and examines only a constant number of bits of the proof, (iii) if the input belongs to the language, then there is a proof with which the verifier accepts with probability 1, and (iv) if the input does not belong to the language, then there is no proof with which the verifier accepts with probability at least  $\frac{1}{2}$ . This theorem is optimal in the sense that  $\text{NP} = \text{PCP}(o(\log n), o(\log n))$  implies  $\text{P} = \text{NP}$  [FGL<sup>+</sup>96]. In this model the verifier’s error is one-sided, in the sense that it accepts each member of the

language with probability one given a correct proof. An alternative model is the one in which the verifier is allowed to make an error for both members and nonmembers, but the error probability is bounded by a constant that is strictly less than  $\frac{1}{2}$ . When the amount of randomness is fixed to  $\mathcal{O}(\log n)$  an important question is how many bits of information have to be examined to achieve the desired error probabilities. For the one-sided-error PCP model, the current best known result is due to Guruswami et al. [GLST98]: For every constant  $\epsilon$ , every language in NP has a one-sided-error PCP protocol that uses  $\mathcal{O}(\log n)$  random bits, examines only three bits, and, for each nonmember, makes an error with probability at most  $\frac{1}{2} + \epsilon$ . For the two-sided-error PCP model, Samorodnitsky and Trevisan [ST00] show the following strong result: For all constants  $\epsilon > 0$  and for all positive integers  $q$ , every language in NP has a two-sided PCP protocol that uses  $\mathcal{O}(\log n)$  random bits, examines  $q$  bits, accepts each member with probability at least  $1 - \epsilon$  given a correct proof, and rejects each nonmember with probability at least  $1 - 2^{-q + \mathcal{O}(\sqrt{q})}$ . This is essentially the current best bound. Håstad and Wigderson [HW01] present a simpler analysis of the proof of Samorodnitsky and Trevisan, and show that the error probability in the soundness condition can be slightly improved.

The PCP Theorem also improves upon the nonapproximability result in [FGL<sup>+</sup>96] as follows: For every constant  $\epsilon > 0$ , it is NP-hard to approximate the size of the largest clique in a graph in polynomial time within a factor of  $n^{1-\epsilon}$ . The proof of the PCP Theorem is very complex and long, and thus, is beyond the coverage of the book. The reader may tackle the paper by Arora et al. [ALM<sup>+</sup>98] for a complete presentation. The PCP Theorem is a culmination of the research on interactive proof systems, and it opened up a new research subarea: NP-hardness of approximation based upon PCP characterizations of NP. There is a vast literature in this subarea. We refer the reader to two surveys, one by Arora and Lund [AL97], which discusses the basic results in the subarea, and the other by Bellare [Bel96], which discusses further development. Crescenzi and Kann ([CK], see also [ACG<sup>+</sup>99]) maintain a web site containing a compendium of NP optimization problems.

## 7. The Nonsolvable Group Technique

Bounded-width branching programs offer interesting connections between complexity classes and algebraic structures. Let  $k \geq 2$  and  $n \geq 1$  be integers. A width- $k$  branching program over  $n$ -bit inputs prescribes manipulation of a pebble placed on a track of  $k$  squares. First the pebble is placed on square 1. Then a sequence of instructions is executed. Each instruction is simple: it tells you to examine an input bit and then move the pebble to another (possibly the same) square, where to which square the pebble will be moved depends on the examined bit, the current location of the pebble, and the step of the computation. The program accepts the input if and only if the pebble is not on square 1 at the end.

How big is the class of things that are accepted by a family of bounded-width branching programs of polynomially many instructions? In the case where  $k = 2$ , since there are only two squares, the class does not seem large enough to contain many interesting languages other than the parity function (constructing such a program is easy). Then how about  $k = 3$ ? Again, 3 does not seem big enough for us to handle complicated membership criteria. Then how about 4, 5, or 6? Note that for every  $k \geq 2$  a width- $k$  branching program can be simulated by a bounded-fan-in circuit whose depth is proportional to the logarithm of the program size, i.e., the number of instructions. So we ask whether bounded-width branching programs can simulate every circuit in nonuniform-NC<sup>1</sup>.

**Pause to Ponder 7.1** *Can polynomial-size, bounded-width branching programs simulate nonuniform-NC<sup>1</sup>?*

Indeed, polynomial-size, bounded-width branching programs can simulate nonuniform-NC<sup>1</sup>? Interestingly, to simulate nonuniform-NC<sup>1</sup> the width of polynomial-size branching programs can be as small as 5. However, it is believed that the width cannot be smaller than that. A significant difference seems to exist between the computational power of width-4 programs and that of width-5 programs. Much to our surprise, the crucial difference lies in the fact that the permutation group over  $\{1, \dots, k\}$  is nonsolvable for  $k \geq 5$  while it is solvable for  $k = 1, 2, 3, 4$ . Recall that a group is solvable if its derived series,  $G_0, G_1, \dots$ , converges to the trivial group, where  $G_0 = G$  and for every  $i \geq 1$ ,  $G_i$  is the *commutator subgroup* of  $G_{i-1}$ , i.e.,  $G_i$  is the group

generated by the elements  $\{h_2^{-1} \circ h_1^{-1} \circ h_2 \circ h_1 \mid h_1, h_2 \in G_{i-1}\}$ . Here we call  $h_2^{-1} \circ h_1^{-1} \circ h_2 \circ h_1$  the *commutator* of  $h_1$  and  $h_2$ . Using nonsolvability of the permutation group over  $\{1, \dots, 5\}$ , one can design a family of polynomial-size, width-5 branching programs for each language in nonuniform-NC<sup>1</sup>.

In this chapter we study the power of polynomial-size, bounded-width branching programs and how such results translate into uniform complexity classes. For a formal definition of the classes that are discussed in this chapter see Sect. A.18.

## 7.1 GEM: Width-5 Branching Programs Capture Nonuniform-NC<sup>1</sup>

### 7.1.1 Equivalence Between Width-5 Branching Programs and NC<sup>1</sup>

As we have just mentioned, polynomial-size, width-5 branching programs capture nonuniform-NC<sup>1</sup>. In fact, the two computational models are equal.

**Theorem 7.2** 5-PBP = nonuniform-NC<sup>1</sup>.

We need to define some notions and notation. Recall that a monoid is a finite set  $S$  of objects with an associated binary operation  $\circ$  and an identity element. Let  $k \geq 2$  be an integer. By  $\mathcal{M}_k$  we denote the monoid consisting of all mappings of  $\{1, \dots, k\}$  to itself and by  $I_k$  we denote the identity mapping in  $\mathcal{M}_k$ . The binary operation  $\circ$  is defined as follows: For all  $\alpha, \beta \in \mathcal{M}_k$ ,  $\alpha \circ \beta$  is the mapping  $\gamma \in \mathcal{M}_5$  such that for all  $i$ ,  $1 \leq i \leq k$ ,  $\gamma(i) = \alpha(\beta(i))$ . The operation  $\circ$  is associative, i.e., for all  $\alpha, \beta, \gamma \in \mathcal{M}_k$ ,

$$\alpha \circ (\beta \circ \gamma) = (\alpha \circ \beta) \circ \gamma.$$

By  $S_k$  we denote the permutation group over  $\{1, \dots, k\}$ , i.e., the set of all bijections from  $\{1, \dots, k\}$  to itself.

Let  $n \geq 1$  and let  $P = \{(i_j, \mu_j^0, \mu_j^1)\}_{j=1}^m$  be a width- $k$  branching program for  $\Sigma^n$ . Then, by  $|P|$ , we denote its length,  $m$ . For each  $x \in \Sigma^n$ ,  $P[x]$  denotes the product

$$\mu_m^{x_{i_m}} \circ \dots \circ \mu_2^{x_{i_2}} \circ \mu_1^{x_{i_1}}.$$

Now we prove Theorem 7.2.

**Proof of Theorem 7.2** We first prove  $5\text{-PBP} \subseteq \text{nonuniform-NC}^1$ . The inclusion follows from a more general statement:

**Lemma 7.3** For all  $k \geq 2$ ,  $k\text{-PBP} \subseteq \text{nonuniform-NC}^1$ .

**Proof of Lemma 7.3** Let  $k \geq 2$ . Let  $L$  be a language in  $k\text{-PBP}$  and  $\mathcal{P} = \{P_n\}_{n \geq 1}$  be a family of width- $k$ , polynomial-size branching programs that decides  $L$ . For all  $n \geq 1$ , and for all  $x \in \Sigma^n$ ,

$$x \in L \iff P_n[x](1) \neq 1.$$

Let  $p$  be a polynomial bounding the size of  $\mathcal{P}$ , i.e., for all  $n \geq 1$ ,  $|P_n| \leq p(n)$ . We'll replace  $\mathcal{P}$  by a family of width- $k$ , polynomial-size branching programs,  $\mathcal{P}' = \{P'_n\}_{n \geq 1}$ , such that  $\mathcal{P}'$  decides  $L$  and, for all  $n \geq 1$ ,  $|P'_n|$  is a power of 2 and is at most  $2p(n)$ . Let  $n \geq 1$  and let  $m = |P_n|$ . If  $m$  is a power of 2, then  $P'_n = P_n$ . If  $m$  is not a power of 2, we construct  $P'_n$  as follows: Let  $t$  be the smallest integer such that  $2^t > m$ . Then  $2^t < 2m \leq 2^t p(n)$ . Let  $P'_n$  be the program of size  $2^t$  such that, for all  $j$ ,  $1 \leq j \leq m$ , the  $j$ th instruction of  $P'_n$  is equal to that of  $P_n$  and, for all  $j$ ,  $m+1 \leq j \leq 2^t$ , the  $j$ th instruction of  $P'_n$  is  $(1, I_k, I_k)$ . Then, for all  $x \in \Sigma^n$ ,  $P'_n[x] = P_n[x]$ . So,  $P'_n$  has the desired properties.

Let  $n \geq 1$  be fixed. Let  $P'_n = \{(i_j, \mu_j^0, \mu_j^1)\}_{j=1}^m$ . Let  $t$  be such that  $2^t = m$ . For all  $x \in \Sigma^n$  and for all integers  $r$  and  $s$  such that  $1 \leq r \leq s \leq m$ , define  $\pi(r, s)[x]$  define inductively as follows:

- If  $r = s$ , then  $\pi(r, s)[x] = \mu_s^{x_{i_j}}$ .
- If  $r > s$ , then  $\pi(r, s)[x] = \mu_s^{x_{i_j}} \circ \pi(r, s-1)[x]$ .

Clearly, for all  $x \in \Sigma^n$ ,  $P'_n[x] = \pi(1, m)[x]$ .

Since the monoid operation  $\circ$  is associative, for every  $x \in \Sigma^n$ , the expression  $\pi(1, m)[x]$  can be evaluate by a simple divide-and-conquer method:

- ★ Let  $r$  and  $s$  be integers such that  $1 \leq r < s \leq m$ . To evaluate  $\pi(r, s)[x]$ , evaluate  $\alpha = \pi(r, \lfloor (r+s)/2 \rfloor)[x]$  and  $\beta = \pi(\lfloor (r+s)/2 \rfloor + 1, s)[x]$  individually, and then set  $\pi(r, s)[x]$  to  $\beta \circ \alpha$ .

Since  $m$  is a power of 2, the divide-and-conquer evaluation method can be viewed as a full binary tree having height  $t$ , where for all  $d$ ,  $0 \leq d \leq t$ , and  $j$ ,  $1 \leq j \leq 2^d$ , the task at the  $j$ th node from right at depth  $d$  is to evaluate  $\pi((j-1)2^{t-d} + 1, j2^{t-d})[x]$ . Call this tree  $T_n[x]$ .

We construct a bounded-fan-in circuit  $C_n$  for  $L^n$  by transforming the tree  $T_n[x]$  into a circuit. To accomplish this, we need to fix a binary encoding of the mappings in  $\mathcal{M}_k$ . Let  $\ell = \lceil \log(k^k) \rceil$ . Since  $|\mathcal{M}_k| = k^k$ ,  $2^\ell \geq |\mathcal{M}_k|$ . We encode each element of  $\mathcal{M}_k$  as a  $2\ell$ -bit string as follows: Let  $g_1, \dots, g_{k^k}$  be an enumeration of the members of  $\mathcal{M}$ . Then for each  $i$ ,  $1 \leq i \leq k^k$ , the encoding of  $g_i$ , denoted by  $e(g_i)$ , is the  $2\ell$ -bit string  $y$  such that the first half of  $y$  has rank  $i$  in  $\Sigma^\ell$  and the second half of  $y$  is the bitwise complement of the first half of  $y$ . Let  $W = \{y \in \Sigma^{2\ell} \mid (\exists g \in \mathcal{M}_k)[y = e(g)]\}$ . Let  $Q : W \times W \rightarrow W$  be the function defined for all  $y, z \in W$  by

$$Q(y, z) = e(e^{-1}(z) \circ e^{-1}(y)).$$

In other words, the function  $Q$  takes two strings in  $W$  and computes the encoding of the product of the mappings encoded by the two strings. Also, let  $R : W \rightarrow \Sigma$  be the function defined for all  $y \in W$  by

$$R(y) = \begin{cases} 1 & \text{if } e^{-1}(y)(1) \neq 1, \\ 0 & \text{otherwise.} \end{cases}$$

In other words,  $R$  takes a string in  $W$  and tests whether the mapping encoded by  $W$  maps 1 to something other than 1. We will show in Fact 7.4 that  $Q$  and  $R$  can be computed by depth- $\mathcal{O}(k \log k)$  bounded-fan-in circuits. For now, let us assume the correctness of the fact and present how the circuits for  $Q$  and  $R$  are built.

Note that at the leaf level of  $T_n[x]$ , each component of the product  $P'_n[x]$  is evaluated depending on a single bit of  $x$ . For each  $j$ ,  $1 \leq j \leq m$ , there is a depth-0 circuit that computes  $e(\pi(j, j)[x])$ . Let  $j$ ,  $1 \leq j \leq m$ , be fixed. Let  $a_1 \cdots a_{2\ell} = e(\mu_j^0)$  and  $b_1 \cdots b_{2\ell} = e(\mu_j^1)$ . For each  $r$ ,  $1 \leq r \leq \ell$ , the  $r$ th output bit of  $e(\pi(i, j)[x])$  is

$$\begin{cases} 0 & \text{if } a_r = b_r = 0, \\ 1 & \text{if } a_r = b_r = 1, \\ x_{i_j} & \text{if } a_r = 0 \text{ and } b_r = 1, \\ \overline{x_{i_j}} & \text{if } a_r = 1 \text{ and } b_r = 0, \end{cases}$$

and the  $(\ell + r)$ th output bit of  $e(\pi(i, j)[x])$  is

$$\begin{cases} 1 & \text{if } a_r = b_r = 0, \\ 0 & \text{if } a_r = b_r = 1, \\ \overline{x_{i_j}} & \text{if } a_r = 0 \text{ and } b_r = 1, \\ x_{i_j} & \text{if } a_r = 1 \text{ and } b_r = 0. \end{cases}$$

Since this circuit computes by simply assigning input bits, the depth of the circuit is 0.

Note that at each nonleaf level of  $T_n[x]$ , divide-and-conquer is applied. So, for each  $d$ ,  $0 \leq d \leq t-1$ , and  $r$ ,  $1 \leq r \leq 2^d$ , we put the circuit for computing  $Q$  at the  $r$ th node from right at level  $d$ , where the first (respectively, the second)  $2\ell$  input bits of the circuit are the  $2\ell$  output bits of the circuit at the  $(2r-1)$ th position (respectively, at the  $2r$ th position) from right at level  $d+1$ .

The resulting circuit computes  $e(P'_n[x])$ . We feed the outputs of the circuit to the circuit for computing  $R$ . This is  $C_n$ . Then, for all  $x \in \Sigma^n$ ,  $C_n(x) = 1$  if and only if  $R(e(P'_n[x])) = 1$ , and thus,  $C_n(x) = 1$  if and only if  $P'_n[x](1) \neq 1$ . Clearly, the depth of the circuit  $C_n$  is  $\mathcal{O}((k \log k)t)$ , and this is  $\mathcal{O}(\log n)$  since  $k$  is fixed.

Now it remains to show that depth- $\mathcal{O}(k \log k)$  circuits exists for  $Q$  and  $R$ .

**Fact 7.4** *There is a depth- $\mathcal{O}(k \log k)$ , bounded-fan-in boolean circuit  $H$  that computes  $Q$  in the following sense: For all  $y, z \in \Sigma^{2\ell}$ , if  $y, z \in W$ , then  $H(yz) = Q(y, z)$ .*

*Also, there is a depth- $\mathcal{O}(k \log k)$ , bounded-fan-in boolean circuit  $H'$  that computes  $R$  in the following sense: For all  $y \in \Sigma^{2\ell}$ , if  $y \in W$ , then  $H'(y) = R(y)$ .*

**Proof of Fact 7.4** Let  $s$  be an integer such that  $1 \leq s \leq k^k$ . Note that  $e(g_s)$  had exactly  $\ell$  1s. Let  $r_1, \dots, r_\ell$  be an enumeration of the  $\ell$  positions at



which the bit of  $e(g_s)$  is a 1. Let  $F_s$  be a bounded-fan-in boolean circuit that takes  $y = y_1 \dots y_{2\ell} \in \Sigma^{2\ell}$ , and computes

$$y_{r_1} \wedge \dots \wedge y_{r_\ell}.$$

Since each string  $y \in W$  has exactly  $\ell$  1's, for all  $y \in W$ ,

$$F_s(y) = \begin{cases} 1 & \text{if } y = e(g_s), \\ 0 & \text{otherwise.} \end{cases}$$

Let  $s_1$  and  $s_2$  be integers such that  $1 \leq s_1, s_2 \leq k^k$ . Let  $G_{s_1, s_2}$  be a bounded-fan-in boolean circuit that takes a pair of  $2\ell$  bits strings  $y = y_1 \dots y_{2\ell}$  and  $z = z_1 \dots z_{2\ell}$  and outputs  $w = w_1 \dots w_{2\ell}$ , defined as follows: Let  $\gamma = \gamma_1 \dots \gamma_{2\ell}$  be  $e(e^{-1}(g_{s_2} \circ g_{s_1}))$ . Then, for each  $r$ ,  $1 \leq r \leq 2\ell$ ,  $w_r$  is given as

$$\gamma_r \wedge F_{s_1}(y) \wedge F_{s_2}(z).$$

Then, for all  $y, z \in W$ ,

$$G_{s_1, s_2}(yz) = \begin{cases} \gamma & \text{if } y = e(g_{s_1}) \text{ and } z = e(g_{s_2}), \\ 0^{2\ell} & \text{otherwise.} \end{cases}$$

Now, for each  $r$ ,  $1 \leq r \leq 2\ell$ , let the  $r$ th output bit of  $H(yz)$  be defined by

$$\bigvee_{1 \leq s_1, s_2 \leq k^k} G_{s_1, s_2}^{(r)}(yz),$$

where  $G_{s_1, s_2}^{(r)}(y, z)$  denotes the  $r$ th output bit of  $G_{s_1, s_2}(y, z)$ . Then, for all  $y, z \in W$ ,

$$H(yz) = e(e^{-1}(z) \circ e^{-1}(y)).$$

The depth of  $H$  can be  $\lceil \log \ell \rceil + 2 + \lceil \log k^{2k} \rceil$ . This is  $\mathcal{O}(k \log k)$ . Thus, the first claim of the fact holds.

To prove the second claim, let  $J = \{s \mid 1 \leq s \leq k^k \wedge g_s(1) \neq 1\}$ . Then, for all  $y \in W$ ,  $R(y) = 1 \iff (\exists s \in J)[y = g_s]$ . Define

$$H'(y) = \bigvee_{s \in J} F_s(y).$$

Then, for all  $y \in W$ ,  $H'(y) = R(y)$ . The depth of  $H'$  can be  $\lceil \ell \rceil + \lceil \log(k-1)^{k-1} \rceil$ . This is  $\mathcal{O}(k \log k)$ . Thus, the second claim of the fact holds.

□ Fact7.4

This proves the first part.

Next we prove the other part, i.e., nonuniform-NC<sup>1</sup>  $\subseteq$  5-PBP. Let  $L$  be a language in nonuniform-NC<sup>1</sup>. Let  $\mathcal{C} = \{C_n\}_{n \geq 1}$  be a family of bounded-fan-in, depth- $\mathcal{O}(\log n)$ , polynomial-size boolean circuits that decides  $L$ . Let  $c > 0$  be a constant such that for every  $n \geq 1$  it holds that  $\text{depth}(C_n) \leq c \log n$ .

For each 5-tuple of integers  $a, b, c, d, e$  such that  $\{a, b, c, d, e\} = \{1, 2, 3, 4, 5\}$ ,  $(a\ b\ c\ d\ e)$  denotes the permutation that maps  $a$  to  $b$ ,  $b$  to  $c$ ,  $c$  to  $d$ ,  $d$  to  $e$ , and  $e$  to  $a$ . Define the following permutations in  $S_5$ :

$$\alpha = (1\ 2\ 3\ 4\ 5), \beta = (1\ 3\ 5\ 4\ 2), \text{ and } \gamma = (1\ 3\ 2\ 5\ 4).$$

Then  $\gamma$  is the commutator of  $\alpha$  and  $\beta$ , i.e.,  $\gamma = \beta^{-1} \circ \alpha^{-1} \circ \beta \circ \alpha$ . Furthermore, define

$$\theta_0 = (1\ 2\ 5\ 3\ 4), \theta_1 = (1\ 4\ 2\ 5\ 3), \theta_2 = (1\ 4\ 3\ 2\ 5), \text{ and } \theta_3 = (1\ 5\ 3\ 2\ 4).$$

By inspection one can easily verify that the following fact holds.

**Fact 7.5**  $\theta_0^{-1} \circ \gamma \circ \theta_0 = \alpha$ ,  $\theta_1^{-1} \circ \gamma \circ \theta_1 = \beta$ ,  $\theta_2^{-1} \circ \gamma \circ \theta_2 = \alpha^{-1}$ , and  $\theta_3^{-1} \circ \gamma \circ \theta_3 = \beta^{-1}$ .

Let  $n \geq 1$  be fixed. Let  $\hat{u}$  be the output gate of  $C_n$ . For each gate  $f$  in  $C_n$  and each input  $x \in \{0, 1\}^n$ , let  $f(x)$  be the output of  $f$  on input  $x$ . Then, for all  $x \in \Sigma^n$ ,  $C_n(x) = \hat{u}(x)$ .

Let  $Q$  be a branching program on  $\{0, 1\}^n$ , let  $f$  be a gate in  $C_n$ , and let  $\theta$  and  $\xi$  be members of  $S_5$ . We say that  $Q$  is a  $(\theta, \xi)$  program for  $f$  if for every  $x \in \{0, 1\}^n$ ,  $Q[x]$ , the mapping induced by  $Q$  on input  $x$ , satisfies

$$Q[x] = \begin{cases} \theta, & \text{if } f(x) = 0, \\ \xi, & \text{if } f(x) = 1. \end{cases}$$

We will construct for each gate  $f$  in  $C_n$  its  $(I_5, \gamma)$  program  $P^f$ . Then, for all  $x \in \Sigma^n$ ,  $P^{\hat{u}}[x] = I_5$  if  $\hat{u}(x) = 0$  and  $P^{\hat{u}}[x] = \gamma$  if  $\hat{u}(x) = 1$ . Note that  $I_5(1) = 1$  and  $\gamma(1) = 3$ . So,  $P^{\hat{u}}$  fixes 1 and  $\gamma$  moves 1 to 3. So,  $P^{\hat{u}}$  is a width-5 branching program for  $L^n$ . The construction is inductive, proceeding from the input level toward the output level.

First, let  $f$  be any input gate. Define  $P^f$  as follows:

- If  $f$  is labeled by  $x_i$  for some  $i$ , then  $P^f = \{(i, I_5, \gamma)\}$ .
- If  $f$  is labeled by  $\bar{x}_i$  for some  $i$ , then  $P^f = \{(i, \gamma, I_5)\}$ .
- If  $f$  is labeled by 1, then  $P^f = \{(1, \gamma, \gamma)\}$ .
- If  $f$  is labeled by 0, then  $P^f = \{(1, I_5, I_5)\}$ .

Then clearly  $P^f$  is an  $(I_5, \gamma)$  program for  $f$ .

Next, let  $f$  be a gate at a non-input level. Let  $g$  and  $h$  be the gates providing inputs to  $f$ . Suppose that we have already obtained a size- $k$ ,  $(I_5, \gamma)$  program  $P^g$  for  $g$  and a size- $l$ ,  $(I_5, \gamma)$  program  $P^h$  for  $h$ . We consider two cases:  $f$  is an AND gate and  $f$  is an OR gate.

We first consider the case in which  $f$  is an AND gate. We construct a program  $T_0$  from  $P^g$  such that  $|T_0| = |P^g|$  and, for every  $x \in \{0, 1\}^n$ ,

$$T_0[x] = \theta_0^{-1} \circ P^g[x] \circ \theta_0.$$

This is done as follows: Let  $(i, s, t)$  be the first instruction of  $P^g$ . We replace this instruction by  $(i, s \circ \theta_0, t \circ \theta_0)$ . Let  $R$  be the resulting program. Let

$(j, u, v)$  be the last instruction of  $R$ . We replace this instruction by  $(j, \theta_0^{-1} \circ u, \theta_0^{-1} \circ v)$ . This is  $T_0$ . Then  $T_0$  has the desired properties. By Fact 7.5,  $\theta_0^{-1} \circ \gamma \circ \theta_0 = \alpha$  and  $\theta_0^{-1} \circ I_5 \circ \theta_0 = I_5$ . Since,  $P^g$  is an  $(I_5, \gamma)$  program for  $g$ ,  $T_0$  is an  $(I_5, \alpha)$  program for  $g$ .

Similarly, construct  $T_1$  from  $P^h$  using  $\theta_1$  in place of  $\theta_0$ ,  $T_2$  from  $P^g$  similarly with  $\theta_2$  in place of  $\theta_0$ , and  $T_3$  from  $P^h$  similarly with  $\theta_3$  in place of  $\theta_0$ . Then,  $T_1$  is a size- $l$ ,  $(I_5, \beta)$  program for  $h$ ,  $T_2$  is a size- $k$ ,  $(I_5, \alpha^{-1})$  program for  $g$ , and  $T_3$  is a size- $l$ ,  $(I_5, \beta^{-1})$  program for  $h$ .

Define  $P^f$  to be the program that executes  $T_0$ , then  $T_1$ , then  $T_2$ , and then  $T_3$ . Then, for every  $x \in \{0, 1\}^n$ , the following conditions hold:

- If both  $g(x) = h(x) = 1$ , then  $P^f[x] = \beta^{-1} \circ \alpha^{-1} \circ \beta \circ \alpha = \gamma$ .
- If  $g(x) = 1$  and  $h(x) = 0$ , then  $P^f[x] = I_5 \circ \alpha^{-1} \circ I_5 \circ \alpha = I_5$ .
- If  $g(x) = 0$  and  $h(x) = 1$ , then  $P^f[x] = \beta^{-1} \circ I_5 \circ \beta \circ I_5 = I_5$ .
- If  $g(x) = h(x) = 0$ , then  $P^f[x] = I_5 \circ I_5 \circ I_5 \circ I_5 = I_5$ .

So,  $P^f$  is an  $(I_5, \gamma)$  program for  $f$  and has size  $2(k + l) \leq 4 \max\{k, l\}$ .

Next we consider the case in which  $f$  is an OR gate. As in Case 1, we construct  $P^f$  from four programs  $T_0, \dots, T_3$ .  $T_0$  is constructed from  $P^g$  by inserting  $\gamma^{-1} \circ \theta_3 \circ \gamma$  before the first instruction and  $\theta_3^{-1}$  after the last instruction without increasing the program size. By Fact 7.5,  $\theta_3^{-1} \circ \gamma^{-1} \circ \theta_3 = \beta$ . Since,  $P^g$  is an  $(I_5, \gamma)$  program for  $g$ ,  $T_0$  is a  $(\beta \circ \gamma, \gamma)$  program for  $g$ . Thus,  $T_0$  is a size- $k$ ,  $(\beta \circ \gamma, \gamma)$  program for  $g$ .

$T_1$  is constructed from  $P^h$  by inserting  $\gamma^{-1} \circ \theta_2$  before the first instruction and  $\theta_2^{-1}$  after the last instruction. By Fact 7.5,  $\theta_2^{-1} \circ \gamma^{-1} \circ \theta_2 = \alpha$ . Since  $P^h$  is a size- $l$ ,  $(I_5, \gamma)$  program for  $h$ ,  $T_1$  is a size- $l$ ,  $(\alpha, I_5)$  program for  $h$ .

For  $T_2$  we use  $P^g$  and insert  $\gamma^{-1} \circ \theta_1$  before the first instruction and  $\theta_1^{-1}$  after the last. By Fact 7.5,  $\theta_1^{-1} \circ \gamma^{-1} \circ \theta_1 = \beta^{-1}$ . Since  $P^g$  is a size- $k$ ,  $(I_5, \gamma)$  program for  $g$ ,  $T_2$  is a size- $k$ ,  $(\beta^{-1}, I_5)$  program for  $g$ .

For  $T_3$  we use  $P^h$  and insert  $\gamma^{-1} \circ \theta_0$  before the first instruction and appending  $\theta_0^{-1}$  after the last. By Fact 7.5,  $\theta_0^{-1} \circ \gamma^{-1} \circ \theta_0 = \alpha^{-1}$ . Since  $P^h$  is a size- $l$ ,  $(I_5, \gamma)$  program for  $h$ ,  $T_3$  is a size- $l$ ,  $(\alpha^{-1}, I_5)$  program for  $h$ .

Now define  $P^f$  to be the program that executes  $T_0$ , then  $T_1$ , then  $T_2$ , and then  $T_3$ . Then for every  $x \in \{0, 1\}^n$ , the following conditions hold:

- If  $g(x) = h(x) = 0$ , then  $P^f[x] = \alpha^{-1} \circ \beta^{-1} \circ \alpha \circ (\beta \circ \gamma) = I_5$ .
- If  $g(x) = 0$  and  $h(x) = 1$ , then  $P^f[x] = I_5 \circ \beta^{-1} \circ I_5 \circ (\beta \circ \gamma) = \gamma$ .
- If  $g(x) = 1$  and  $h(x) = 0$ , then  $P^f[x] = \alpha^{-1} \circ I_5 \circ \alpha \circ \gamma = \gamma$ .
- If  $g(x) = h(x) = 1$ , then  $P^f[x] = I_5 \circ I_5 \circ I_5 \circ \gamma = \gamma$ .

So  $P^f$  is an  $(I_5, \gamma)$  program for  $f$  and has size  $2(k + l) \leq 4 \max\{k, l\}$ .

Define  $P_n = P^{\hat{u}}$ . Since  $\text{depth}(C_n) \leq c \log n$ ,  $|P_n| \leq 4^{\text{depth}(C_n)} \leq n^{2c}$ . Hence,  $L$  is recognized by a family of polynomial-size, width-5 branching programs  $\square$  Theorem 7.2

### 7.1.2 Programs over a Nonsolvable Group Capture NC<sup>1</sup>

We generalize the notion of branching programs. Let  $M$  be a finite monoid. A program over  $M$  for  $\Sigma^n$  is a sequence of instructions  $P = \{(i_j, s_j^0, s_j^1)\}_{j=1}^m$  such that for all  $j$ ,  $1 \leq j \leq m$ ,  $1 \leq i_j \leq n$  and  $s_j^0, s_j^1 \in M$ . For each string  $x \in \Sigma^n$ ,  $P[x]$  is defined as

$$s_m^{x_{i_m}} \circ \dots \circ s_2^{x_{i_2}} \circ s_1^{x_{i_1}}.$$

We say that  $P$  accepts  $x$  if  $P[x] \neq e$ , where  $e$  is the identity mapping of  $M$ . For  $W \subseteq \Sigma^n$ , we say that  $P$  decides  $W$  if for every  $x \in \Sigma^n$  it holds that  $x \in W \iff P$  accepts  $x$ . Let  $\mathcal{P} = \{P_n\}_{n \geq 1}$  be a family of programs over  $M$  such that, for every  $n \geq 1$ ,  $P_n$  is a program for  $\Sigma^n$ . We say that  $\mathcal{P}$  decides a language  $L$  if for all  $n \geq 1$   $P_n$  decides  $L^n$ . For a boolean function  $f$  over  $\{0, 1\}^n$ , and  $s_0, s_1 \in M$ , we say that a program  $P$  is an  $(s_0, s_1)$  program for  $f$  if for every  $x \in \{0, 1\}^n$  the product generated by the instructions on input  $x$  is  $s_0$  if  $f(x) = 0$  and  $s_1$  otherwise.

To prove that nonuniform-NC<sup>1</sup>  $\subseteq$  5-PBP, we showed that for all  $\omega \in \{\alpha, \beta, \alpha^{-1}, \beta^{-1}, \gamma\}$ , for all integers  $n \geq 1$  and  $d \geq 0$ , and for all depth- $d$ , bounded-fan-in circuits  $C$  with  $n$  inputs, there exists a length  $4^d$ ,  $(I_5, \omega)$  program for  $C$ . By generalizing this we can show that, for every nonsolvable group  $G$ , there is an integer  $B > 0$  such that, for all integers  $n \geq 1$  and  $d \geq 0$ , for all depth- $d$ , bounded-fan-in circuits  $C$  with  $n$  inputs, and for all  $s \in G$ , both  $C$  and its negation have a size- $B^d$ ,  $(e, s)$  program, where  $e$  is the identity of  $G$ .

**Theorem 7.6** *Let  $G$  be an arbitrary nonsolvable group. Let  $L$  be an arbitrary language in nonuniform-NC<sup>1</sup>. Then  $L$  is decided by a family of polynomial-size programs over  $G$ .*

**Proof** The proof is almost the same as that of Theorem 7.2. Let  $G$  be an arbitrary nonsolvable group. Since  $G$  is nonsolvable there exists some nontrivial subgroup  $H$  of  $G$  such that  $G$ 's derived series  $G_0, G_1, \dots$  converges to  $H$ . Let  $C$  be a circuit and let  $g$  be a gate in  $C$ . We define the height of  $g$  in  $C$  to be the length of the longest downward path from  $g$  to any input gate. Note that all input gates of  $C$  have height 0 and the output gate of  $C$  has height  $\text{depth}(C)$ .

**Lemma 7.7** *Let  $n \geq 1$ . Let  $C$  be a bounded-fan-in circuit with  $n$  inputs. Let  $H$  be an arbitrary nonsolvable group such that its commutator subgroup (the group generated by the commutators of  $H$ ) is identical to  $H$ . Let  $e$  be the identity element of  $H$  and let  $s$  be an arbitrary element in  $H$ . For every  $h$ ,  $0 \leq h \leq \text{depth}(C)$ , and for every gate  $g$  in  $C$  having height  $h$ ,  $g$  has an  $(e, s)$  program over  $H$  and an  $(s, e)$  program over  $H$ , both having size at most  $(4||H||)^h$ .*

**Proof of Lemma 7.7** Let  $n, C, H$ , and  $e$  be as defined in the hypothesis. Let  $B = 4||H||$ . Let  $s$  be an arbitrary element in  $H$ . If  $s = e$ , then the statement of the lemma trivially holds since for every gate  $g$  in  $C$ ,  $\{(1, e, e)\}$  is an  $(e, e)$  program for  $g$ . So, assume that  $s \neq e$ .

Note that in the boolean circuit model we use, the negation appears only at the input level. We show that for every gate  $g$  in  $C$  there is an  $(e, s)$  program,  $P^g$ , for  $g$  having length at most  $(4||H||)^h$ , where  $h$  is the height of  $g$ .

The proof is by induction on  $h$ . For the base case, let  $h = 0$ . Let  $g$  be a gate having height  $h$ . We define the program for  $g$  as follows:

- If  $g$  is labeled by  $x_i$  for some  $i$ , then  $P^g = \{(i, e, s)\}$ .
- If  $g$  is labeled by  $\overline{x_i}$  for some  $i$ , then  $P^g = \{(i, s, e)\}$ .
- If  $g$  is labeled by 1, then  $P^g = \{(1, s, s)\}$ .
- If  $g$  is labeled by 0, then  $P^g = \{(1, e, e)\}$ .

Then  $P^g$  is a desired  $(e, s)$  program for  $g$ .

For the induction step, let  $h = h_0 \geq 1$  and suppose that the claim holds for all values of  $h$  that are less than  $h_0$  and greater than or equal to 0. Let  $g$  be a gate having height  $h$ . Let  $g_1$  and  $g_2$  be inputs of  $g$ . Since the commutator subgroup of  $H$  is  $H$  itself, every element in  $H$  can be expressed as the product of commutators of  $H$ . The minimum number of commutators of  $H$  necessary to express  $s$  is at most  $||H||$ . To see why, suppose that there exists some  $s \in H$  such that the smallest number of commutators of  $H$  that are needed to express  $s$  is  $k > ||H||$ . Let  $t_1 \cdots t_k$  be commutators of  $H$  such that

$$s = t_k \cdots t_1.$$

For all  $i$ ,  $1 \leq i \leq k$ , the partial product  $t_i \cdots t_1$  is a member of  $H$ . Since  $k > ||H||$ , there exist a pair of indices  $(i, j)$  such that  $1 \leq i < j \leq k$  and  $t_i \cdots t_1 = t_j \cdots t_1$ . Then

$$s = t_k \cdots t_{j+1} t_i \cdots t_1.$$

This implies that a shorter expression for  $s$  exists, a contradiction. Thus, the length of expression for each element of  $H$  has length at most  $||H||$ . Let  $s$  be expressed as

$$\beta_k^{-1} \alpha_k^{-1} \beta_k \alpha_k \cdots \beta_1^{-1} \alpha_1^{-1} \beta_1 \alpha_1, \quad (7.1)$$

where  $1 \leq k \leq ||H||$  and  $\alpha_1, \dots, \alpha_k, \beta_1, \dots, \beta_k$  are commutators of  $H$ . The gates  $g_1$  and  $g_2$  are at height  $\leq h_0 - 1$ . Then by our induction hypothesis, for every  $i$ ,  $1 \leq i \leq k$ , there exist the following programs  $P_i, Q_i, R_i$ , and  $S_i$ :

- $P_i$  is an  $(e, \alpha_i)$  program for  $g_1$  and has length less than or equal to  $B^{h_0-1}$ .
- $Q_i$  is an  $(e, \beta_i)$  program for  $g_2$  and has length less than or equal to  $B^{h_0-1}$ .
- $R_i$  is an  $(e, \alpha_i^{-1})$  program for  $g_1$  and has length less than or equal to  $B^{h_0-1}$ .
- $S_i$  is an  $(e, \beta_i^{-1})$  program for  $g_2$  and has length less than or equal to  $B^{h_0-1}$ .

Let  $T$  be the program that executes  $P_1, Q_1, R_1, S_1, \dots, P_k, Q_k, R_k, S_k$  in that order. For all  $x \in \Sigma^n$ , if  $g_1(x) = g_2(x) = 1$ , then  $T[x] = s$ . Also, if either  $g_1(x) = 0$  or  $g_2(x) = 0$ , then  $T[x] = e$ . Thus,  $T$  is an  $(e, s)$  program for  $g$ . Since  $k \leq \|H\|$  and  $P_1, Q_1, R_1, S_1, \dots, P_k, Q_k, R_k, S_k$  all have length at most  $(4\|H\|)^{h_0-1}$ , the length of the program  $T$  is at most  $(4\|H\|)^{h_0}$  as desired.

To construct an  $(s, e)$  program for  $g$  having length at most  $(4\|H\|)^{h_0}$ , we follow the above construction with  $s^{-1}$  in place of  $s$  to define an  $(e, s^{-1})$  program for  $g$ . Let  $(i, \xi, \theta)$  be its first instruction. Then we replace this first instruction by  $(i, \xi \circ s, \theta \circ s)$ . The replacement turns the program into, while preserving the program size, an  $(s, e)$  program for  $g$ .

The construction in the case when  $g$  is an  $\vee$  gate uses the same idea. Let  $g' = \neg g$ ,  $g'_1 = \neg g_1$ , and  $g'_2 = \neg g_2$ . Then  $g = \neg g'$  and  $g' = g'_1 \wedge g'_2$ . For all gates  $g$  and all  $\xi, \theta \in H$ , a  $(\xi, \theta)$  program for  $g$  is a  $(\theta, \xi)$  program for  $\neg g$ . So, we obtain an  $(e, s^{-1})$  program for  $g'$  and then replace its first instruction, say  $(i, \xi, \theta)$ , by  $(i, \xi \circ s, \theta \circ s)$ . That turns the program into, while preserving the program size, an  $(s, e)$  program for  $g'$ , which is an  $(e, s)$  program for  $g$ .

□ Lemma 7.7

To complete the proof, note that  $\text{NC}^1$  circuits have depth  $\mathcal{O}(\log n)$  and that  $B^{\mathcal{O}(\log n)} = \mathcal{O}(n^{c \log B})$ . Thus, the resulting program has polynomial size. □

## 7.2 Width-5 Bottleneck Machines Capture PSPACE

Let  $k \geq 2$ . Recall that  $\text{SF}_k$  is the class of languages  $L$  for which there exists some polynomial  $p$  and some polynomial time computable function  $f : \Sigma^* \times \Sigma^* \rightarrow \mathcal{M}_k$  such that for every  $x \in \Sigma^*$  it holds that

$$x \in L \iff \left( f(x, 1^{p(|x|)}) \circ f(x, 1^{p(|x|)-1}0) \circ \dots \right. \\ \left. \circ f(x, 0^{p(|x|)-1}1) \circ f(x, 0^{p(|x|)}) \right) (1) = 1.$$

By translating Theorem 7.2 to polynomial space-bounded computation, we show that the power of  $\text{SF}_5$  is exactly that of PSPACE.

**Theorem 7.8**  $\text{SF}_5 = \text{PSPACE}$ .

In Theorem 7.2 branching programs are shown to be able to simulate  $\text{NC}^1$  circuits. In Theorem 7.8 rather than use a bottleneck machine to simulate a PSPACE machine, we construct a polynomial-depth circuit from an instance of QBF, a canonical complete language for PSPACE. We then show that a width-5 bottleneck machine can evaluate all such circuits and determine for each given instance, whether the instance is a member of QBF.

**Proof of Theorem 7.8**  $\text{SF}_5 \subseteq \text{PSPACE}$  holds because a polynomial space-bounded Turing machine can, by cycling through all counter values, compute the product of all the exponentially many mappings associated with

a given input  $x$ . For the other direction, we will show  $\text{QBF} \in \text{SF}_5$ . Then, since  $\text{SF}_5$  is closed under polynomial-time many-one reductions, it follows that  $\text{PSPACE} \subseteq \text{SF}_5$ .

Define  $\alpha, \beta, \gamma$ , and  $\theta_0, \dots, \theta_3$  as in the proof of Theorem 7.2. We will define a polynomial-time computable function  $f : \Sigma^* \times \Sigma^* \rightarrow \mathcal{S}_5$  such that for every  $n \geq 1$ , and for every fully quantified boolean formula  $\zeta$  of  $n$  variables,

$$f(\zeta, 1^{2n}) \circ \dots \circ f(\zeta, 0^{2n}) = \gamma \text{ if } \zeta \in \text{QBF} \text{ and } I_5 \text{ otherwise.}$$

This will establish that  $\overline{\text{QBF}} \in \text{SF}_5$ . Since  $\overline{\text{QBF}}$  is  $\leq_m^p$ -reducible to  $\text{QBF}$ , we will then have  $\text{QBF} \in \text{SF}_5$ . In order to construct such a function we will be scaling up the proof of Theorem 7.2, from logarithmic depth to polynomial depth. Let  $\zeta$  be a fully quantified boolean formula of the form

$$Q_1 x_1 \dots Q_n x_n \varphi(x_1, \dots, x_n).$$

$\zeta$  can be naturally viewed as a bounded-fan-in boolean circuit in the shape of a full binary tree having height  $n$  with  $2^n$  inputs, where the inputs of the circuit are  $\varphi(0, \dots, 0), \dots, \varphi(1, \dots, 1)$  and, for each  $i$ ,  $1 \leq i \leq n$ , the gates at level  $i$  (distance  $i$  from the input level) are AND gates if  $Q_{n+1-i} = \forall$  and are OR gates if  $Q_{n+1-i} = \exists$ . Call this circuit  $C_\zeta$ . Since it is a tree, each gate of  $C_\zeta$  can be specified uniquely by the downward path from the root (the output gate). For each  $y \in (\Sigma^*)^{\leq n}$ , the gate specified by  $y$  evaluates the following formula:

- If  $y$  is the empty string, the formula is  $\zeta$ .
- If  $1 \leq |y| \leq n-1$ , then the formula is

$$Q_{|y|+1} x_{|y|+1} \dots Q_n x_n \varphi(b_1, \dots, b_{|y|}, x_{|y|+1}, \dots, x_n),$$

where for every  $i$ ,  $1 \leq i \leq |y|$ ,  $b_i$  is the  $i$ th bit of  $y$ .

- If  $|y| = n$ , then the formula is  $\varphi(b_1, \dots, b_{|y|})$ , where for every  $i$ ,  $1 \leq i \leq n$ ,  $b_i$  is the  $i$ th bit of  $y$ .

We apply the construction of a branching program described in Theorem 7.2 to  $C_\zeta$  to build a directed graph,  $T_\zeta$ , in the shape of a full quaternary tree having height  $n$ . In  $T_\zeta$  each nonroot  $v$  is bidirectionally connected to its parent. For each nonleaf of  $T_\zeta$ , we assign numbers  $0, \dots, 3$  to its four children from right to left. Since the nodes of  $T_\zeta$  are laid out in a full quaternary tree, each node of  $T_\zeta$  can be specified by a unique downward path from the root. Written in binary, for every  $m$ ,  $0 \leq m \leq n$ , the length of the path for each node at depth  $m$  is  $2m$ . The empty string specifies the root and for each  $m$ ,  $1 \leq m \leq n$ , and for each  $u = b_1 \dots b_{2m} \in \{0, 1\}^{2m}$ , the string  $u$  specifies the node that is reached from the root by the downward path along which for each  $d$ ,  $1 \leq d \leq m$ , the edge towards the  $(b_{2d-1}b_{2d})$ th child is selected at depth  $d-1$ , where  $00, 01, 10$ , and  $11$  stand for  $0, 1, 2$ , and  $3$ , respectively.

We let each node of  $T_\zeta$  correspond to a fully quantified boolean formula. Let  $u$  be a binary string such that  $|u|$  is even and  $0 \leq |u| \leq 2n$ . The formula

corresponding to the node specified by  $u$ , denoted by  $F(u)$ , is determined as follows:

- If  $u$  is the empty string,  $F(u) = \zeta$ .
- If  $2 \leq |u| \leq 2(n-1)$ , then

$$F(u) = Q_{m+1}x_{m+1} \cdots Q_n x_n \varphi(b_2, b_4, \dots, b_{2m-2}, b_{2m}, x_{m+1}, \dots, x_n),$$

where for every  $i$ ,  $1 \leq i \leq 2m$ ,  $b_i$  is the  $i$ th bit of  $y$ .

- If  $|u| = 2n$ , then

$$F(u) = \varphi(b_2, b_4, \dots, b_{2n-2}, b_{2n}),$$

where for every  $i$ ,  $1 \leq i \leq 2n$ ,  $b_i$  is the  $i$ th bit of  $y$ .

Next we label each edge and each leaf of  $T_\zeta$  by an element of  $\mathcal{M}_5$ . For each edge  $e = (u, v)$ , write  $\lambda_E(u, v)$  to denote the label assigned to  $e$  and, for each leaf  $u$ , write  $\lambda_V(u)$  to denote the label assigned to  $u$ . Let  $u$  be any node of  $T_\zeta$ . Let  $P(u)$  denote the product of the labels, defined as follows:

- If  $u$  is a leaf, then  $P(u) = \lambda_V(u)$ .
- If  $u$  is not a leaf, let  $v_0, v_1, v_2, v_3$  be the four children of  $u$ , enumerated from right to left. For each  $i$ ,  $0 \leq i \leq 3$ , let  $\alpha_i = \lambda_E(u, v_i)$  and  $\beta_i = \lambda_E(v_i, u)$ . Then

$$\begin{aligned} P(u) = & \beta_3 \circ P(v_3) \circ \alpha_3 \circ \beta_2 \circ P(v_2) \circ \alpha_2 \circ \\ & \beta_1 \circ P(v_1) \circ \alpha_1 \circ \beta_0 \circ P(v_0) \circ \alpha_0. \end{aligned}$$

In other words,  $P(u)$  is the product of the all labels that are encountered during the in-order traversal of the subtree rooted at  $u$ , where at every nonleaf node, the children are visited from right to left.

We assign these labels are assigned so that, for all  $u$ , it holds that  $P(u) = \gamma$  if  $F(u) = \text{True}$  and  $P(u) = I_5$  otherwise. To accomplish this, we use the construction in the proof of Theorem 7.2. Recall that, to construct a program for an  $\wedge$ -gate or an  $\vee$ -gate, we concatenated four programs that were constructed recursively, and that we inserted into each of the four programs two constant mappings, one at the beginning and the other at the end. The four children of a nonleaf node correspond to the four components, so for each  $i$ ,  $0 \leq i \leq 3$ , the downward edge to the  $i$ th child of  $u$  is labeled by the constant mapping that is inserted at the very beginning of the  $i$ th component and the upward edge from that child is labeled by the one inserted at the very end.

More specifically we determine the labels as follows:

1. For every leaf  $u$ , it is labeled by  $\gamma$  if the formula corresponding to it evaluates to 1 and  $I_5$  otherwise.



2. For every  $d$ ,  $0 \leq d \leq n-1$ , such that  $Q_d = \forall$ , for every nonleaf  $u$  at depth  $d$ , and for every  $r$ ,  $0 \leq r \leq 3$ , the label of the edge going to the  $r$ th child from right is  $\theta_r$  and the label of the edge coming back from the child is  $\theta_r^{-1}$ .
3. For every  $d$ ,  $0 \leq d \leq n-1$ , such that  $Q_d = \exists$ , for every nonleaf  $u$  at depth  $d$ , and for every  $r$ ,  $0 \leq r \leq 3$ , the label of the edge going to the  $r$ th child from right is
  - $\gamma^{-1} \circ \theta_3 \circ \gamma$  if  $r = 0$  and
  - $\gamma^{-1} \circ \theta_{3-r}$  if  $r = 1, 2, 3$ ,
 and the label of the edge coming back from the  $r$ th child is  $\theta_{3-r}^{-1}$ .

Now we show that these labels give us the property we need.

**Fact 7.9** *For every node  $u$  of  $T_\zeta$ ,  $P(u) = \gamma$  if  $F(u) = \text{True}$  and  $I_5$  otherwise.*

**Proof of Fact 7.9** We prove the fact by induction on the height  $h$  of the subtree of  $T_\zeta$  rooted at  $u$ . For the base case, suppose that  $h = 0$ . Then  $u$  is a leaf. Then  $P(u) = \lambda_V(u)$ . According to rule 1,  $\lambda_V(u)$  equals  $\gamma$  if  $F(u) = \text{True}$  and equals  $I_5$  otherwise. Thus the claim holds for  $h = 0$ .

For the induction step, suppose that  $h = h_0$  for some  $h_0 > 0$  and that the claim holds for all values of  $h$  less than  $h_0$  and greater than or equal to 0. Let  $u$  be a node such that the subtree rooted at  $u$  has height  $h$ . Let  $Q = Q_{n+1-h}$ . First suppose that  $Q = \forall$ . Let  $v_0, \dots, v_3$  be the children of  $u$  enumerated from right to left. Note that the downward path from the root to  $v_0$  is identical to that to  $v_2$  except that the second-to-last bit is a 0 for  $v_0$  and is a 1 for  $v_2$ . Since the second-to-last bit is not used to determine  $F(v_0)$  or  $F(v_2)$ , we have  $F(v_0) = F(v_2)$ . For much the same reason,  $F(v_1) = F(v_3)$ . Since  $Q = \forall$ ,  $F(u) = F(v_0) \wedge F(v_1)$ . By rule 2,  $P(u)$  is

$$\begin{aligned}
 &(\theta_3^{-1} \circ P(v_1) \circ \theta_3) \circ (\theta_2^{-1} \circ P(v_0) \circ \theta_2) \\
 &\quad \circ (\theta_1^{-1} \circ P(v_1) \circ \theta_1) \circ (\theta_0^{-1} \circ P(v_0) \circ \theta_0).
 \end{aligned}$$

By our induction hypothesis,  $P(v_0) = \gamma$  if  $F(v_0) = \text{True}$  and  $P(v_0) = I_5$  otherwise, and the same holds for  $P(v_1)$ . According to the analysis for the case in which  $f$  is an AND gate on page 172, we have the following:

- $\theta_3^{-1} \circ P(v_1) \circ \theta_3$  is equal to  $\beta^{-1}$  if  $F(v_1) = \text{True}$  and is equal to  $I_5$  otherwise.
- $\theta_2^{-1} \circ P(v_0) \circ \theta_2$  is equal to  $\alpha^{-1}$  if  $F(v_0) = \text{True}$  and is equal to  $I_5$  otherwise.
- $\theta_1^{-1} \circ P(v_1) \circ \theta_1$  is equal to  $\beta$  if  $F(v_1) = \text{True}$  and is equal to  $I_5$  otherwise.
- $\theta_0^{-1} \circ P(v_0) \circ \theta_0$  is equal to  $\alpha$  if  $F(v_0) = \text{True}$  and is equal to  $I_5$  otherwise.

Thus,  $P(u) = \gamma$  if  $P(v_1) = P(v_0) = \text{True}$  and  $P(u) = I_5$  otherwise. Hence, the claim holds for the case when  $Q = \forall$ .

Next suppose that  $Q = \exists$ . By following an analysis similar to the above,  $P(u)$  is equal to

$$\begin{aligned}
 &(\theta_0 \circ P(v_1) \circ \gamma^{-1} \circ \theta_0^{-1}) \circ (\theta_1 \circ P(v_0) \circ \gamma^{-1} \circ \theta_1^{-1}) \\
 &\quad \circ (\theta_2 \circ P(v_1) \circ \gamma^{-1} \circ \theta_2^{-1}) \circ (\theta_3 \circ P(v_0) \circ \gamma^{-1} \circ \theta_3^{-1} \gamma).
 \end{aligned}$$

By our induction hypothesis,  $P(v_0) = \gamma$  if  $F(v_0) = \text{True}$  and  $P(v_0) = I_5$  otherwise, and the same holds for  $P(v_1)$ . Then, by inverting  $(\theta_0^{-1} \circ \gamma \circ \theta_0)^{-1} = \alpha$ , we have the following:

- $\theta_0 \circ P(v_1) \circ \gamma^{-1} \circ \theta_0^{-1}$  is equal to  $I_5$  if  $F(v_1) = \text{True}$  and is equal to  $\alpha^{-1}$  otherwise.

Similarly, we have the following:

- $\theta_1 \circ P(v_0) \circ \gamma^{-1} \circ \theta_1^{-1}$  is equal to  $I_5$  if  $F(v_0) = \text{True}$  and is equal to  $\beta^{-1}$  otherwise.
- $\theta_2 \circ P(v_1) \circ \gamma^{-1} \circ \theta_2^{-1}$  is equal to  $I_5$  if  $F(v_1) = \text{True}$  and is equal to  $\alpha$  otherwise.
- $\theta_3 \circ P(v_0) \circ \gamma^{-1} \circ \theta_3^{-1}$  is equal to  $\gamma$  if  $F(v_0) = \text{True}$  and is equal to  $\beta \circ \gamma$  otherwise.

Since  $\alpha^{-1} \circ \beta^{-1} \circ \alpha \circ \beta = \gamma^{-1}$ ,  $P(u) = \gamma$  if  $F(u) = \text{True}$  and  $P(u) = I_5$  otherwise. Hence, the claim holds for the case where  $Q = \exists$ . Thus, the claim holds for all  $d$ ,  $0 \leq d \leq n$ .  $\square$  Fact 7.9

Let  $\hat{u}$  be the root of  $T_\zeta$ . Now it suffices to show that there is a polynomial-time computable function  $f$  such that

$$P(\hat{u}) = f(\zeta, 1^{2n}) \circ \dots \circ f(\zeta, 0^{2n}).$$

Recall that the definition of  $P(\hat{u})$  corresponds to the in-order traversal of the tree. For each  $w = b_1 \dots b_{2n} \in \Sigma^{2n} \setminus \{0^{2n}, 1^{2n}\}$ , we define  $f(\zeta, w)$  to be the product of all the labels that are encountered while moving from the leaf  $w'$  to the leaf  $w$  during the in-order traversal of the tree, where  $w'$  is the predecessor of  $w$  in  $\Sigma^{2n}$ . We define  $f(\zeta, 0^{2n})$  to be the product of all the labels that are encountered while moving from the root to the leaf  $0^{2n}$  during the traversal and  $f(\zeta, 1^{2n})$  to be the product of all the labels that are encountered while moving from the leaf  $1^{2n-1}0$  to the root. More precisely,  $f(\zeta, w)$  is defined as follows:

1. If  $w = 0^{2n}$ , then

$$\begin{aligned} f(\zeta, 0^{2n}) &= \lambda_V(0^{2n}) \circ \lambda_E(0^{2n-2}, 0^{2n}) \circ \\ &\dots \circ \lambda_E(00, 0000) \circ \lambda_E(\epsilon, 00). \end{aligned}$$

2. If  $w = 1^{2n}$ , then

$$\begin{aligned} f(\zeta, 1^{2n}) &= \lambda_E(11, \epsilon) \circ \lambda_E(1111, 11) \circ \dots \circ \lambda_E(1^{2n}, 1^{2n-2}) \circ \\ &\lambda_V(1^{2n}) \circ \lambda_E(1^{2n-2}, 1^{2n}) \circ \lambda_E(1^{2n}10, 1^{2n-2}). \end{aligned}$$

3. If  $w = w_1 \dots w_{2n} \in \Sigma^{2n} \setminus \{0^{2n}, 1^{2n}\}$ , let  $w' = w'_1 \dots w'_{2n}$  denote the predecessor of  $w$  in  $\{0, 1\}^{2n}$ . Let  $m$  be the largest integer  $i$  such that the prefix of  $w$  having length  $2i$  is equal to the prefix of  $w'$  having length  $2i$ . In other words,  $w_1 \dots w_{2m}$  is the least common ancestor of  $w$  and  $w'$ . We define

$$\begin{aligned}
f(\zeta, w) &= \lambda_V(w) \circ \\
&\lambda_E(w_1 \cdots w_{2n-2}, w) \circ \cdots \circ \lambda_E(w_1 \cdots w_{2m}, w_1 \cdots w_{2m+2}) \circ \\
&\lambda_E(w'_1 \cdots w'_{2m+2}, w'_1 \cdots w'_{2m}) \circ \cdots \circ \lambda_E(w', w'_1 \cdots w'_{2n-2}).
\end{aligned}$$

It is easy to see that the product

$$f(\zeta, 1^{2n}) \circ \cdots \circ f(\zeta, 0^{2n})$$

is equal to  $R_\zeta$ . The labels  $\lambda_E$  and  $\lambda_V$  can be easily computed. The number of terms in each value of  $f$  is bounded by  $2n + 1$  (the maximum is achieved when  $w = b_1 c_1 0^{2n-2}$  for some  $b_1 c_1 \in \{01, 10, 11\}$ ). So  $f$  is polynomial-time computable. Thus,  $\text{QBF} \in \text{SF}_5$ . Hence,  $\text{PSPACE} \subseteq \text{SF}_5$ .  $\square$  Theorem 7.8

## 7.3 Width-2 Bottleneck Computation

In the previous section, we showed that width-5 bottleneck Turing machines capture PSPACE. Here we study the complexity of width-2 bottleneck computation from three angles. First, we ask what power polynomial-size width-2 bottleneck Turing machines possess. Second we ask, in regards to width-2 computation, how important the order of the instructions is. Then finally we ask how much computational power is added if the machines are allowed to behave probabilistically. In the following discussion let  $\nu_{=1}$  (respectively,  $\nu_{=2}$ ) denote the constant function in  $\mathcal{M}_2$  that maps both 1 and 2 to 1 (respectively, 2).

### 7.3.1 Width-2 Bottleneck Turing Machines

$\oplus\text{OptP}$  is the class of all languages  $L$  for which there exists a language  $A \in \oplus\text{P}$  and a function  $g \in \text{OptP}$  such that for every  $x \in \Sigma^*$

$$x \in L \iff \langle x, g(x) \rangle \in A.$$

The goal of this section is to prove the following theorem, which states that the class of languages accepted by polynomial-time width-2 bottleneck computation is identical to  $\oplus\text{OptP}$ .

**Theorem 7.10**  $\text{SF}_2 = \oplus\text{OptP}$ .

**Proof** Throughout this proof we use the following notation. For each string  $y$ ,  $\text{rank}(y)$  denotes the rank of  $y$  in  $\Sigma^{|y|}$ . Also, for each integer  $n \geq 1$  and  $i$ ,  $1 \leq i \leq 2^n$ ,  $\text{str}_n(i)$  denotes the string  $y \in \Sigma^n$  such that  $\text{rank}(y) = i$ .

We first prove that  $\text{SF}_2 \subseteq \oplus\text{OptP}$ . Suppose that  $L \in \text{SF}_2$ . There exists a polynomial  $p$  and a polynomial-time computable function  $f : \Sigma^* \times \Sigma^* \rightarrow \mathcal{M}_2$  such that, for every  $x \in \Sigma^*$ ,

$$x \in L \iff \left( f(x, 1^{p(|x|)}) \circ \dots \circ f(x, 0^{p(|x|)}) \right) (1) = 1.$$

For each  $x \in \Sigma^*$ , define

$$Q[x] = f(x, 1^{p(|x|)}) \circ \dots \circ f(x, 0^{p(|x|)}).$$

Then, for all  $x \in \Sigma^*$ ,

$$x \in L \iff Q[x] \in \{I_2, \nu_{=1}\}.$$

Define  $N$  to be the nondeterministic Turing machine that, on input  $x \in \Sigma^*$ , guesses a string  $y \in \Sigma^{p(|x|)}$  and then outputs  $\text{rank}(y)$  if  $f(x, y) \in \{\nu_{=1}, \nu_{=2}\}$  and outputs 0 otherwise.  $N$  can be polynomial time-bounded. For all  $x \in \Sigma^*$ ,  $N$  on input  $x$  outputs a nonnegative integer along each computation path. Let  $g$  be the OptP function defined by  $N$ , i.e., for all  $x \in \Sigma^*$ ,

$$g(x) = \max\{i \in \mathbb{N} \mid \text{some path of } N(x) \text{ has } i \text{ as its output}\}.$$

For each  $x \in \Sigma^*$  and each  $i \geq 0$ , define

$$M(x, i) = ||\{z \mid z \in \Sigma^{p(|x|)} \wedge \text{rank}(z) \geq i + 1 \wedge f(x, z) = (1 \ 2)\}||.$$

Define

$$\begin{aligned} A = \{ \langle x, i \rangle \mid & x \in \Sigma^* \wedge i \geq 0 \wedge \\ & ((i = 0 \wedge M(x, 0) \text{ is an even number}) \vee \\ & (1 \leq i \leq p(|x|) \wedge f(x, \text{str}_{p(|x|)}(i)) = \nu_{=1} \wedge \\ & M(x, i) \text{ is an even number}) \vee \\ & (1 \leq i \leq p(|x|) \wedge f(x, \text{str}_{p(|x|)}(i)) = \nu_{=2} \wedge \\ & M(x, i) \text{ is an odd number}) \}. \end{aligned}$$

Then  $A \in \oplus P$ . To see why, let

$$A' = \{ \langle x, i \rangle \mid x \in \Sigma^* \wedge 0 \leq i \leq 2^{p(|x|)} \wedge M(x, i) \text{ is an odd number} \}.$$

Then  $A' \in \oplus P$  and  $A \leq_{1\text{-tt}}^p A'$ . By part 2 of Proposition 4.8,  $\oplus P$  is closed under  $\leq_T^p$ -reductions. So,  $A \in \oplus P$ . We now prove that the membership in  $L$  can be decided by the membership  $A$  with  $g$  as advice.

**Fact 7.11** *For every  $x \in \Sigma^*$ ,  $x \in L \iff \langle x, g(x) \rangle \in A$ .*

**Proof of Fact 7.11** Let  $x \in \Sigma^*$  be fixed. We consider the following three possibilities:

- $g(x) = 0$ ,
- $1 \leq g(x) \leq 2^{p(|x|)}$  and  $f(x, \text{str}_{p(|x|)}(g(x))) = \nu_{=1}$ , and
- $1 \leq g(x) \leq 2^{p(|x|)}$  and  $f(x, \text{str}_{p(|x|)}(g(x))) = \nu_{=2}$ .

First consider the case when  $g(x) = 0$ . Since  $g(x) = 0$ , for all  $y \in \Sigma^{p(|x|)}$ ,  $f(x, y) \in \{I_2, (1\ 2)\}$ . So,  $x \in L \iff Q[x] = I_2$ . It holds that

$$\langle x, 0 \rangle \in A \iff M(x, 0) \text{ is an even number}$$

and

$$Q[x] = I_2 \iff M(x, 0) \text{ is an even number.}$$

Thus,  $x \in L \iff \langle x, g(x) \rangle \in A$ .

Next consider the case when  $1 \leq g(x) \leq 2^{p(|x|)}$  and  $f(x, \text{str}_{p(|x|)}(g(x))) = \nu_{=1}$ . It holds that  $Q[x] \in \{\nu_{=1}, \nu_{=2}\}$ . So,  $x \in L \iff Q[x] = \nu_{=1}$ . It holds that

$$\langle x, g(x) \rangle \in A \iff M(x, g(x)) \text{ is an even number}$$

and

$$Q[x] = \nu_{=1} \iff M(x, g(x)) \text{ is an even number.}$$

Thus,  $x \in L \iff \langle x, g(x) \rangle \in A$ .

Finally, consider the case when  $1 \leq g(x) \leq 2^{p(|x|)}$  and  $f(x, \text{str}_{p(|x|)}(g(x))) = \nu_{=2}$ . It holds that  $Q[x] \in \{\nu_{=1}, \nu_{=2}\}$ . So,  $x \in L \iff Q[x] = \nu_{=1}$ . It holds that

$$\langle x, g(x) \rangle \in A \iff M(x, g(x)) \text{ is an odd number}$$

and

$$Q[x] = \nu_{=1} \iff M(x, g(x)) \text{ is an odd number.}$$

Thus,  $x \in L \iff \langle x, g(x) \rangle \in A$ .

□ Fact 7.11

By Fact 7.11, we have  $L \in \oplus\text{OptP}$ .

Next we prove that  $\oplus\text{OptP} \subseteq \text{SF}_2$ . Let  $L$  be a language in  $\oplus\text{OptP}$ . Let  $A$  be a language in  $\oplus\text{P}$  and let  $g$  be a polynomial, such that  $A$  and  $g$  jointly witness that  $L \in \oplus\text{OptP}$ , i.e., for all  $x \in \Sigma^*$ ,

$$x \in L \iff \langle x, g(x) \rangle \in A.$$

Since  $g \in \text{OptP}$ , there is a polynomial-time nondeterministic Turing machine such that, for every  $x \in \Sigma^*$ ,  $g(x)$  is the maximum of the output values of the machine on input  $x$ . By definition, for every  $x \in \Sigma^*$ ,  $N$  on input  $x$  outputs a nonnegative integer along each computation path. Let  $p$  be a polynomial that bounds the runtime of  $N$ . Then, for all  $x \in \Sigma^*$ , each output string of  $N$  on input  $x$  has at most  $p(|x|)$  bits. This implies that, for all  $x \in \Sigma^*$ , each output of  $N$  on input  $x$  is in the interval  $[0, 2^{p(|x|)} - 1]$ . On the other hand, for each  $x \in \Sigma^*$ , the rank of a string having length  $p(|x|)$  is in the interval  $[1, 2^{p(|x|)}]$ . So, for each  $x \in \Sigma^*$ , we correspond  $\Sigma^{p(|x|)}$  to  $\{0, \dots, 2^{p(|x|)} - 1\}$  by letting each  $y \in \Sigma^{p(|x|)}$  represent the integer  $\text{rank}(y) - 1$ .

We may assume that at each computation step,  $N$  has two possible (not necessarily distinct) moves. Then, for all  $x \in \Sigma^*$ , each computation path of  $N$  on  $x$  can be uniquely encoded as a string of length  $p(|x|)$ . Since  $A \in \oplus\text{P}$ , there exist a polynomial  $r$  and  $B \in \text{P}$ , such that, for all  $x \in \Sigma^*$ ,

$x \in A \iff ||\{y \mid y \in \Sigma^{r(|x|)} \wedge \langle x, y \rangle \in B\}||$  is an odd number.

We can choose the polynomial  $r$  so that it is strictly increasing, i.e., for all  $n \geq 0$ ,  $r(n+1) > r(n)$ . Take  $k$  to be the smallest integer such that, for all  $n \geq 0$ ,  $kn^k + k \geq r(n)$ . We'll replace  $r(n)$  by  $r'(n) = kn^k + k$  and replace  $B$  by

$$B' = \{\langle x, yw \rangle \mid y \in \Sigma^{r'(|x|)} \wedge w = 0^{r'(|x|)-r(|x|)} \wedge \langle x, y \rangle \in B\}.$$

Then  $r'$  is strictly increasing and, for all  $x \in \Sigma^*$ ,

$$||\{y \mid y \in \Sigma^{r'(|x|)} \wedge \langle x, y \rangle \in B\}|| = ||\{y \mid y \in \Sigma^{r'(|x|)} \wedge \langle x, y \rangle \in B'\}||.$$

Let  $l$  be a polynomial such that, for all  $x \in \Sigma^*$  and  $i$ ,  $0 \leq i \leq 2^{p(|x|)} - 1$ ,  $|\langle x, i \rangle| \leq l(|x|)$ . Define  $s(n) = 2p(n) + r'(l(n))$ .

Define  $f : \Sigma^* \times \Sigma^* \rightarrow \mathcal{M}_2$  as follows: Let  $x, w \in \Sigma^*$ .

- If  $|w| \neq s(|x|)$ ,  $f(x, w) = I_2$ .
- If  $|w| = s(|x|)$ , let  $yzuv$  be the decomposition of  $w$  such that  $|y| = |z| = p(|x|)$  and  $|u| = r'(l(|\langle x, \text{rank}(y) - 1 \rangle|))$ . Then the value of  $f(x, w)$  is defined as follows:
  - If  $N$  on input  $x$  along path  $z$  outputs  $\text{rank}(y) - 1$ ,  $u \in 0^*$ ,  $\langle \langle x, \text{rank}(y) - 1 \rangle, u \rangle \in B'$ , and  $v \in 0^*$ , then  $f(x, w) = \nu_{=1}$ .
  - If  $N$  on input  $x$  along path  $z$  outputs  $\text{rank}(y) - 1$ ,  $u \in 0^*$ ,  $\langle \langle x, \text{rank}(y) - 1 \rangle, u \rangle \notin B'$ , and  $v \in 0^*$ , then  $f(x, w) = \nu_{=2}$ .
  - If  $N$  on input  $x$  along path  $z$  outputs  $\text{rank}(y) - 1$ ,  $u \notin 0^*$ ,  $\langle \langle x, \text{rank}(y) - 1 \rangle, u \rangle \in B'$ , and  $v \in 0^*$ , then  $f(x, w) = (1 \ 2)$ .
  - If  $y, z, u$ , and  $v$  satisfy none of the three conditions above, then  $f(x, w) = I_2$ .

Let  $x \in \Sigma^*$  be fixed and let  $n = |x|$ . Let  $\hat{y} = \text{str}_{p(n)}(g(x) + 1)$  and let  $\hat{z} = \max\{z \in \Sigma^{p(n)} \mid N(x) \text{ outputs } g(x) \text{ along the computation path } z\}$ . Then,  $\text{rank}(\hat{y}) = g(x)$ , and for all  $y, z \in \Sigma^{p(n)}$  and  $w \in \Sigma^{s(n)-2p(n)}$ , if  $\hat{y}\hat{z} <_{lex} yz$ , then  $f(x, yzw) = I_2$ . Also,  $f(x, \hat{y}\hat{z}0^{s(n)-2p(n)}) \in \{\nu_{=1}, \nu_{=2}\}$ . So,

$$\begin{aligned} f(x, 1^{s(n)}) \circ \dots \circ f(x, 0^{s(n)}) \\ = f(x, \hat{y}\hat{z}1^{s(n)-2p(n)}) \circ \dots \circ f(x, \hat{y}\hat{z}0^{s(n)-2p(n)}). \end{aligned}$$

Let  $\alpha = r'(l(|\langle x, g(x)\text{rank}(y) - 1 \rangle|))$  and let  $\beta = s(n) - 2p(n) - \alpha$ . Then, for all  $u \in \Sigma^\alpha$  and  $v \in \Sigma^\beta \setminus \{0^\beta\}$ ,

$$f(x, \hat{y}\hat{z}uv) = I_2.$$

So,

$$\begin{aligned} f(x, \hat{y}\hat{z}1^{s(n)-2p(n)}) \circ \dots \circ f(x, \hat{y}\hat{z}0^{s(n)-2p(n)}) \\ = f(x, \hat{y}\hat{z}u_{2^\alpha}0^\beta) \circ f(x, \hat{y}\hat{z}u_{2^\alpha-1}0^\beta) \circ \dots \circ f(x, \hat{y}\hat{z}u_20^\beta) \circ f(x, \hat{y}\hat{z}u_10^\beta), \end{aligned}$$

where for every  $i$ ,  $1 \leq i \leq 2^\alpha$ ,  $u_i = \text{str}_\alpha(i)$ , i.e., the  $i$ th smallest string in  $\Sigma^\alpha$ . For each  $i$ ,  $1 \leq i \leq 2^\alpha$ , let  $\varphi_i = (1 \ 2)$  if  $\langle \langle x, g(x) \rangle, u_i \rangle \in B'$  and  $\varphi_i = I_2$  otherwise. Note that

$$\nu_{=1} = (1 \ 2) \circ \nu_{=2}$$

and that

$$f(x, \hat{y}\hat{z}y_1 0^\beta) = \begin{cases} \nu_{=1} & \text{if } \langle x, \text{rank}(\hat{y}) - 1 \rangle, y_1 \rangle \in B', \\ \nu_{=2} & \text{otherwise.} \end{cases}$$

So,

$$\begin{aligned} & f(x, \hat{y}\hat{z}u_{2^\alpha} 0^\beta) \circ f(x, \hat{y}\hat{z}u_{2^\alpha-1} 0^\beta) \circ \dots \circ f(x, \hat{y}\hat{z}u_2 0^\beta) \circ f(x, \hat{y}\hat{z}u_1 0^\beta) \\ &= \varphi_{2^\alpha} \circ \dots \circ \varphi_1 \circ \nu_{=2}. \end{aligned}$$

Note that  $||\{i \mid 1 \leq i \leq 2^\alpha \wedge \varphi_i = (1 \ 2)\}|| = ||\{i \mid 1 \leq i \leq 2^\alpha \wedge \langle x, g(x) \rangle, u_i \rangle \in B'\}||$ . Also,  $||\{i \mid 1 \leq i \leq 2^\alpha \wedge \langle x, g(x) \rangle, u_i \rangle \in B'\}||$  is an odd number if and only if  $\langle x, g(x) \rangle \in A$ . So, we have

$$\varphi_{2^\alpha} \circ \dots \circ \varphi_1 \circ \nu_{=2} = \begin{cases} \nu_{=1} & \text{if } \langle x, g(x) \rangle \in A, \\ \nu_{=2} & \text{otherwise.} \end{cases}$$

Thus,

$$f(x, 1^{s(n)}) \circ \dots \circ f(x, 0^{s(n)}) = \begin{cases} \nu_{=1} & \text{if } x \in L, \\ \nu_{=2} & \text{otherwise.} \end{cases}$$

Hence,  $L \in \text{SF}_2$ . □

### 7.3.2 Symmetric Width-2 Bottleneck Turing Machines

We now consider symmetric bottleneck Turing machines. They are defined by allowing bottleneck Turing machines to execute their tasks in arbitrary order, and by demanding that, no matter what the order is, the product of the tasks (as mappings) fixes 1 if and only if the input is to be accepted.

We will show that width-2 symmetric bottleneck Turing machines are much weaker than width-2 bottleneck Turing machines, as every language in  $\text{SSF}_2$  is the disjoint union of a language in NP and another in  $\oplus P$ .

**Theorem 7.12** *For every  $L \in \text{coSSF}_2$ , there exist disjoint sets  $L_1$  and  $L_2$ ,  $L_1 \in \text{NP}$  and  $L_2 \in \oplus P$ , such that  $L = L_1 \cup L_2$ .*

**Proof** Let  $L \in \text{coSSF}_2$  be witnessed by a polynomial-time computable function  $f$  and a polynomial  $p$  such that for every  $x \in \Sigma^*$  and every permutation  $\pi$  of  $\Sigma^{p(|x|)}$ , it holds that

$$x \in \bar{L} \iff \left( f(x, \pi(1^{p(|x|)})) \circ \dots \circ f(x, \pi(0^{p(|x|)})) \right) (1) = 1,$$

or equivalently,

$$x \in L \iff \left( f(x, \pi(1^{p(|x|)})) \circ \dots \circ f(x, \pi(0^{p(|x|)})) \right) (1) = 2.$$

For each  $x \in \Sigma^*$ , define  $S(x)$  to be the set of all  $\mu \in \mathcal{M}_2$  such that for some  $y \in \Sigma^{p(|x|)}$  it holds that  $f(\langle x, y \rangle) = \mu$ . Since  $f(x, \pi(1^{p(|x|)})) \circ \dots \circ f(x, \pi(0^{p(|x|)}))$  maps 1 to the same index regardless of the choice of  $\pi$ , at most one element from  $\{\nu_{=1}, \nu_{=2}, (1\ 2)\}$  can be in  $S(x)$ . Then, for every  $x \in \Sigma^*$ ,  $x \in L$  if and only if either  $S(x)$  contains  $\nu_{=2}$  or  $(S(x)$  contains  $(1\ 2)$  and there is an odd number of  $y \in \Sigma^{p(|x|)}$ , such that  $f(\langle x, y \rangle) = (1\ 2)$ ). The former condition can be tested by an NP set

$$L_1 = \{x \mid (\exists y \in \Sigma^{p(|x|)})[f(\langle x, y \rangle) = \nu_{=2}]\}$$

and the latter can be tested by a  $\oplus P$  set

$$L_2 = \{x \mid ||\{y \mid y \in \Sigma^{p(|x|)} \wedge f(\langle x, y \rangle) = (1\ 2)\}|| \text{ is an odd number}\}.$$

Thus,  $L = L_1 \cup L_2$ . For all  $x \in \Sigma^*$ , if  $x \in L_2$ , then  $(1\ 2) \in S(x)$ , so  $\nu_{=2} \notin S(x)$ , and thus,  $x \notin L_1$ . Thus,  $L_1 \cap L_2 = \emptyset$ .  $\square$

### 7.3.3 Probabilistic Symmetric Bottleneck Turing Machines

The power of width-2 symmetric bottleneck Turing machines is, as we showed in the previous theorem, very restricted. They do not seem powerful enough to include the polynomial hierarchy. However, if they are endowed with access to randomness, they gain the polynomial hierarchy.

**Theorem 7.13**  $\text{ProbabilisticSSF}_2 = \text{NP}^{\text{PP}}$ .

**Proof** First we show that  $\text{ProbabilisticSSF}_2 \supseteq \text{NP}^{\text{PP}}$ . Let  $L$  be any language in  $\text{NP}^{\text{PP}}$ . We claim that there exists a polynomial  $p$  and a language  $A \in \text{C=P}$  such that, for every  $x \in \Sigma^*$ ,  $x \in L$  if and only if there exists some  $y \in \Sigma^{p(|x|)}$  such that  $\langle x, y \rangle \in A$ . To see why this claim holds, let  $N$  be a polynomial time nondeterministic Turing machine and let  $B$  be a language in  $\text{PP}$  such that  $L(N^B) = L$ . Let  $B \in \text{PP}$  and let this be witnessed by a polynomial-time nondeterministic Turing machine  $M$  such that, for every  $x \in \Sigma^*$ ,  $x \in B \iff \# \text{gap}_M(x) \geq 0$ . Let  $q$  be a polynomial bounding the runtime of  $N$ . There is a polynomial  $r$  such that, for every  $x \in \Sigma^*$  and every potential query  $y$  of  $N$  on  $x$ , both  $\# \text{acc}_M(y)$  and  $\# \text{rej}_M(y)$  are strictly less than  $2^{r(|x|)}$ . Define  $T$  to be a nondeterministic polynomial-time oracle machine that, on input  $x \in \Sigma^*$ , behaves as follows:

**Step 1**  $T$  nondeterministically simulates  $N$  on  $x$ . Each time  $N$  makes a query, instead of making that query,  $N$  guesses a single bit  $b \in \{0, 1\}$  and then returns to the simulation assuming that the oracle answer is affirmative if  $b = 1$  and the oracle answer is negative if  $b = 0$ .

**Step 2**  $T$  rejects  $x$  immediately if  $N$  on input  $x$  rejects along the computation path simulated in Step 1.



**Step 3** Let  $y_1, \dots, y_m$  be an enumeration of all the queries made by  $N$  along the computation path that has been simulated in Step 1. For each  $i$ ,  $1 \leq i \leq m$ ,  $T$  guesses  $y_i, z_i \in \Sigma^{r(|x|)}$  and sets  $\alpha_i$  to the rank of  $y_i$  in  $\Sigma^{r(|x|)}$  and  $\beta_i$  to the rank of  $z_i$  in  $\Sigma^{r(|z|)}$ .

**Step 4**  $T$  tests whether there is some  $i$ ,  $1 \leq i \leq m$ , such that either

- $\alpha_i \geq \beta_i$  and the bit  $b$  guessed for query  $y_i$  during the simulation in Step 1 is a 0, or
- $\alpha_i < \beta_i$  and the bit  $b$  guessed for query  $y_i$  during the simulation in Step 1 is a 1.

If there is such an  $i$ ,  $T$  immediately rejects  $x$ .

**Step 5**  $T$  asks its oracle whether  $(\forall i, 1 \leq i \leq m) [\alpha_i = \#acc_M(y_i) \wedge \beta_i = \#rej_M(y_i)]$ .  $T$  accepts  $x$  if the answer from the oracle is affirmative and rejects  $x$  otherwise.

Clearly, the machine  $T$  runs in polynomial time. Define

$$W_{acc} = \{\langle y, m \rangle \mid y \in \Sigma^* \wedge m \geq 0 \wedge m = \#acc_M(y)\}$$

and

$$W_{rej} = \{\langle y, m \rangle \mid y \in \Sigma^* \wedge m \geq 0 \wedge m = \#rej_M(y)\}.$$

Then both  $W_{acc}$  and  $W_{rej}$  belong to  $C=P$ . The queries in Step 5 can be done by a single, conjunctive query to the marked union of  $W_{acc}$  and  $W_{rej}$ . Since  $C=P$  is closed under  $\leq_{crt}^p$ -reductions (see Theorem 9.9), there is a language  $D \in C=P$  such that  $D$  can answer the conjunctive query that is made in Step 5. Let  $p$  be a polynomial bounding the runtime of  $T$ . Since  $T$  is polynomial time-bounded, there is a polynomial  $p$  such that, for all  $x \in \Sigma^*$ , each computation path of  $T$  on input  $x$  can be encoded as a string having length at most  $p(|x|)$ . Define  $A = \{\langle x, u \rangle \mid x \in \Sigma^* \wedge u \in \Sigma^{p(|x|)} \wedge u \text{ is an accepting computation path of } T \text{ on input } x \wedge \text{ the query that } T \text{ on input } x \text{ makes in Step 5 along path } u \text{ belongs to } D\}$ . Then  $A \leq_m^p D$ , and thus,  $A \in C=P$ . Since for every  $x \in \Sigma^*$ ,

$$x \in L \iff (\exists u \in \Sigma^{p(|x|)}) [\langle x, u \rangle \in A],$$

the claim holds.

Since  $A \in C=P$ , there exist a language  $B \in P$  and a polynomial  $q$ , such that for all  $x \in \Sigma^*$ ,

$$x \in A \iff ||\{y \in \Sigma^{q(|x|)} \mid \langle x, y \rangle \in B\}|| = ||\{y \in \Sigma^{q(|x|)} \mid \langle x, y \rangle \notin B\}||.$$

Define  $f : \Sigma^* \times \Sigma^* \rightarrow \{(1\ 2), I_2\}$  to be the probabilistic function defined by the following machine  $M_f$ : On input  $\langle x, y \rangle$ ,  $x \in \Sigma^*$  and  $y \in \Sigma^{p(|x|)}$ ,  $M_f$  selects  $z$  from  $\Sigma^{q(|\langle x, y \rangle|)}$  uniformly at random and then outputs  $(1\ 2)$  if  $\langle \langle x, y \rangle, z \rangle \in B$  and outputs  $I_2$  otherwise. For each  $x \in \Sigma^*$  and  $y \in \Sigma^{p(|x|)}$ , define

$$d(\langle x, y \rangle) = \Pr[f(\langle x, y \rangle) = I_2] - \Pr[f(\langle x, y \rangle) = (1\ 2)].$$

Then, for all  $x \in \Sigma^*$  and  $y \in \Sigma^{p(|x|)}$ ,

$$\langle x, y \rangle \in A \iff d(\langle x, y \rangle) = 0.$$

By routine calculation, for every  $x \in \Sigma^*$  and every permutation  $\pi$  of  $\Sigma^{p(|x|)}$ ,

$$\begin{aligned} & \Pr \left[ \left( f(x, \pi(1^{p(|x|)})) \circ \dots \circ f(x, \pi(0^{p(|x|)})) \right) (1) = 1 \right] \\ & - \Pr \left[ \left( f(x, \pi(1^{p(|x|)})) \circ \dots \circ f(x, \pi(0^{p(|x|)})) \right) (1) = 2 \right] \\ & = \prod_{y \in \Sigma^{p(|x|)}} d(\langle x, y \rangle). \end{aligned}$$

Since the sum of the two terms on the left-hand side of the formula is 1, we have

$$\begin{aligned} & \Pr \left[ \left( f(x, \pi(1^{p(|x|)})) \circ \dots \circ f(x, \pi(0^{p(|x|)})) \right) (1) = 1 \right] \\ & = \frac{1}{2} + \frac{1}{2} \prod_{y \in \Sigma^{p(|x|)}} d(\langle x, y \rangle). \end{aligned}$$

So, for every  $x \in \Sigma^*$ , the following conditions hold:

- If  $x \in L$ , then for some  $y \in \Sigma^{p(|x|)}$  it holds that  $d(x, y) = 0$ , so, for every permutation  $\pi$  of  $\Sigma^{p(|x|)}$ ,

$$\Pr \left[ \left( f(x, \pi(1^{p(|x|)})) \circ \dots \circ f(x, \pi(0^{p(|x|)})) \right) (1) = 1 \right] = \frac{1}{2}.$$

- If  $x \notin L$ , then for every  $y \in \Sigma^{p(|x|)}$   $d(x, y) \neq 0$ , so, for every permutation  $\pi$  of  $\Sigma^{p(|x|)}$ ,

$$\Pr \left[ \left( f(x, \pi(1^{p(|x|)})) \circ \dots \circ f(x, \pi(0^{p(|x|)})) \right) (1) = 1 \right] \neq \frac{1}{2}.$$

Hence,  $L \in \text{ProbabilisticSSF}_2$ .

Conversely, suppose that  $L \in \text{ProbabilisticSSF}_2$ . There exist a polynomial time probabilistic Turing machine  $T$  that defines, on each input  $x$ , a distribution over  $\mathcal{M}_2$  and a polynomial  $t$ , such that, for every  $x \in \Sigma^*$ , and every permutation  $\pi$  over  $\Sigma^{t(|x|)}$ ,

$$x \in L \iff \Pr[f(x, \pi(1^{t(|x|)})) \circ \dots \circ f(x, \pi(0^{t(|x|)}))(1) = 1] = \frac{1}{2}.$$

Let  $p$  be a polynomial that bounds the runtime of  $T$ . Let  $x \in \Sigma^*$  be fixed. Let  $M = 2^{t(|x|)}$ . For each  $i$ ,  $1 \leq i \leq M$ , let  $\alpha_i = \Pr[f(x, y_i) = I_2] - \Pr[f(x, y_i) = (1 \ 2)]$  and  $\beta_i = \Pr[f(x, y_i) = \nu_{=1}] - \Pr[f(x, y_i) = \nu_{=2}]$ , where  $y_i$  is the string in  $\Sigma^{t(|x|)}$  having rank  $i$ .

For each permutation  $\pi$  of  $\{1, \dots, M\}$ , let  $Q[\pi]$  denote

$$\begin{aligned} & \beta_{\pi(M)} + \alpha_{\pi(M)}(\beta_{\pi(M-1)} + \alpha_{\pi(M-1)}(\dots \\ & \beta_{\pi(2)} + \alpha_{\pi(2)}(\beta_{\pi(1)} + \alpha_{\pi(1)}))) \end{aligned} \tag{7.2}$$

By routine calculation, for all permutations  $\pi$  of  $\{1, \dots, M\}$ , it holds that

$$\begin{aligned} Q[\pi] &= \Pr[(f(x, \pi(M)) \circ \dots \circ f(x, \pi(1)))(1) = 1] \\ &\quad - \Pr[(f(x, \pi(M)) \circ \dots \circ f(x, \pi(1)))(1) = 2]. \end{aligned}$$

Then, for every permutation  $\pi$  of  $\Sigma^{t(|x|)}$ ,

$$x \in L \iff Q[\pi] = 0.$$

Let  $r$  be a polynomial such that, for all  $u \in \Sigma^*$  and  $v \in \Sigma^{p(|u|)}$ ,  $q(|\langle u, v \rangle|) \leq r(|v|)$ . We claim that  $x \in L$  if and only if one of the following two conditions holds:

- ( $\star$ ) For some  $i$ ,  $1 \leq i \leq M$ ,  $\alpha_i = \beta_i = 0$ .
- ( $\star\star$ ) There exist some  $m$ ,  $1 \leq m \leq r(|x|)$  and  $j_1, \dots, j_m \in \{1, \dots, M\}$  such that
  - for every  $i \in \{1, \dots, M\} \setminus \{j_1, \dots, j_m\}$ ,  $\beta_i = 0$ , and
  - $\beta_{j_m} + \alpha_{j_m}(\dots \beta_{j_2} + \alpha_{j_2}(\beta_{j_1} + \alpha_{j_1})) = 0$ .

First we show that  $x \in L$  if either ( $\star$ ) or ( $\star\star$ ) holds. Suppose that ( $\star$ ) holds. Let  $i \in \{1, \dots, M\}$  be such that  $\alpha_i = \beta_i = 0$ . Let  $\pi$  be a permutation of  $\{1, \dots, M\}$  that maps  $M$  to  $i$ . Then, by (7.2), for some real number  $Z$ , it holds that  $Q[\pi] = \alpha_i + \beta_i Z$ . Since  $\alpha_i = \beta_i = 0$ , this implies that  $Q[\pi] = 0$ . Thus,  $x \in L$ . Next suppose that ( $\star$ ) does not hold and ( $\star\star$ ) holds. Let  $m \in \{1, \dots, r(|x|)\}$  and  $j_1, \dots, j_m \in \{1, \dots, M\}$  for which the two conditions of ( $\star\star$ ) hold. Let  $\pi$  be a permutation such that for every  $i$ ,  $1 \leq i \leq m$ ,  $\pi(i) = j_i$ . Then

$$Q[\pi] = \left( \prod_{i \in \{1, \dots, M\} \setminus \{j_1, \dots, j_m\}} \alpha_i \right) (\beta_{j_m} + \alpha_{j_m}(\dots \beta_{j_2} + \alpha_{j_2}(\beta_{j_1} + \alpha_{j_1}))).$$

By the second condition of ( $\star\star$ ), the second term on the right-hand side is 0. Thus,  $Q[\pi] = 0$ , and thus,  $x \in L$ .

Next we show that if  $x \in L$  then either ( $\star$ ) or ( $\star\star$ ) holds. Suppose that  $x \in L$ . Let  $K = \{i \mid i \in I \wedge \beta_i \neq 0\}$  and  $M' = |K|$ . Let  $S = \{\sigma \mid \sigma \text{ is a permutation of } \{1, \dots, M\} \wedge \sigma(\{1, \dots, M'\}) = K\}$ . For each  $\sigma \in S$ , let  $Q'[\sigma]$  be the formula  $Q[\sigma]$  with everything beyond index  $\sigma(M')$  eliminated, i.e.,

$$\begin{aligned} &\beta_{\sigma(M')} + \alpha_{\sigma(M')}(\beta_{\sigma(M'-1)} + \alpha_{\sigma(M'-1)}(\dots \\ &\quad \dots \beta_{\sigma(2)} + \alpha_{\sigma(2)}(\beta_{\sigma(1)} + \alpha_{\sigma(1)}))). \end{aligned}$$

Then, as we have seen in the previous part of the proof, for all  $\sigma \in S$ ,

$$Q[\sigma] = Q'[\sigma] \prod_{i \in \{1, \dots, M\} \setminus K} \alpha_i. \quad (7.3)$$

Since  $x \in L$  by our assumption, this implies that for all  $\sigma \in S$ , either  $Q'[\sigma] = 0$  or  $\prod_{i \in \{1, \dots, M\} \setminus K} \alpha_i = 0$ . We will show that if  $(\star)$  does not hold, then  $(\star\star)$  holds. Suppose that  $(\star)$  does not hold, i.e., for every  $i \in \{1, \dots, M\} \setminus K$ ,  $\alpha_i \neq 0$ . Then,  $\prod_{i \in \{1, \dots, M\} \setminus K} \alpha_i \neq 0$ . So, for all  $\sigma \in S$ ,  $Q'[\sigma] = 0$ . We will show below that  $M' \leq r(|x|)$ . Then  $(\star\star)$  holds for an arbitrary enumeration  $j_1, \dots, j_{M'}$  of the members of  $K$ .

Note that, for every  $i \in \{1, \dots, M\}$ ,  $|\alpha_i| + |\beta_i| \leq 1$ . For every  $i \in K$ ,  $\beta_i \neq 0$ . This implies that for all  $i \in K$   $\alpha_i \neq 1$   $\alpha_i \neq -1$ . We also claim that, for every  $i \in K$ ,  $\alpha_i \neq 0$ . To see why, assume that there is some  $i \in K$  such that  $\alpha_i = 0$ . Take  $\sigma \in S$  to be the one that maps  $M'$  to this  $i$ . Then  $Q'[\sigma] = \beta_i$ . This implies  $\beta_i = 0$ , a contradiction because  $i \in K$ . Furthermore, note that  $(\star\star)$  trivially holds if  $M' = 1$ . Suppose  $M' = 1$ . Let  $i$  be the only element of  $K$ . Then, for all  $\sigma \in S$ ,  $Q'[\sigma] = \beta_i + \alpha_i$  and this is 0. So,  $(\star\star)$  holds with  $m = 1$  and  $j_m = i$ . In the following discussion, we thus assume that  $M' \geq 2$  and that for all  $i \in K$   $\alpha_i \notin \{-1, 0, 1\}$ .

Let  $k$  and  $l$  be two distinct elements of  $K$ . Let  $\pi \in S$  be such that  $\pi(M') = k$  and  $\pi(M' - 1) = l$ . Let  $\sigma$  be the permutation in  $S$  such that  $\sigma(M') = l$ ,  $\sigma(M' - 1) = k$ , and for all  $i \in \{1, \dots, M\} \setminus \{k, l\}$ ,  $\sigma(i) = \pi(i)$ . Let

$$\theta = \beta_{\pi(M'-2)} + \alpha_{\pi(M'-2)}(\beta_{\pi(M'-3)} + \alpha_{\pi(M'-3)}(\dots \beta_{\pi(1)} + \alpha_{\pi(1)})).$$

Then

$$Q'[\pi] = \beta_k + \alpha_k(\beta_l + \alpha_l\theta)$$

and

$$Q'[\sigma] = \beta_l + \alpha_l(\beta_k + \alpha_k\theta).$$

By our supposition,  $Q'[\pi] = Q'[\sigma] = 0$ , so  $Q'[\pi] = Q'[\sigma]$ . By canceling  $\alpha_k\alpha_l\theta$ , we obtain

$$\beta_k(1 - \alpha_l) = \beta_l(1 - \alpha_k).$$

Since  $l \in K$ ,  $\alpha_l \neq 1$ . So, we have

$$\beta_k = \frac{1 - \alpha_k}{1 - \alpha_l} \beta_l.$$

This relation holds for all pairs of distinct indices  $(k, l)$  in  $K$ .

Let  $j_1, \dots, j_{M'}$  be an arbitrary enumeration of all elements in  $K$ . Then for every  $k$ ,  $2 \leq k \leq M'$ ,

$$\beta_{j_k} = \frac{1 - \alpha_{j_k}}{1 - \alpha_{j_1}} \beta_{j_1}.$$

Let  $\pi \in S$  such that for all  $k$ ,  $1 \leq k \leq M'$ ,  $\pi(k) = j_k$ . In the expression of  $Q'[\pi]$ , for each  $k$ ,  $2 \leq k \leq M'$ , replace  $\beta_{j_k}$  by  $\frac{1 - \alpha_{j_k}}{1 - \alpha_{j_1}} \beta_{j_1}$ . Then we have

$$Q'[\pi] = \frac{(1 - \zeta)\beta_{j_1}}{(1 - \alpha_{j_1})} + \zeta,$$

where  $\zeta = \prod_{1 \leq k \leq M'} \alpha_{j_k}$ . Since  $Q'[\pi] = 0$  by our supposition,

$$\beta_{j_1} + (1 - \alpha_{j_1} - \beta_{j_1})\zeta = 0. \quad (7.4)$$

By definition of  $r$ , for all  $y \in \Sigma^{t(|x|)}$  and  $\mu \in \mathcal{M}_2$ , the precision of the probability that  $T$  on input  $\langle x, y \rangle$  outputs  $\mu$  is at most  $r(|x|)$ . For all  $l \in K$ ,  $\alpha_l \notin \{-1, 0, 1\}$ . Thus, for every  $l$ ,  $1 \leq l \leq M'$ , there exist some odd (not necessarily positive) integer  $h_k$  and some positive integer  $d_k$  such that  $\alpha_{j_l} = \frac{h_k}{2^{d_k}}$ . This implies that  $\zeta = \frac{h_0}{2^{d_0}}$  for some odd (not necessarily positive) integer  $h_0$  and some positive integer  $D_0 \geq M'$ . Note that  $1 - \alpha_{j_1} - \beta_{j_1} \neq 0$ . This is because if  $1 - \alpha_{j_1} - \beta_{j_1} = 0$  then by equation 7.4 we have  $Q'[\pi] = \beta_{j_1} = 0$ , contradiction our assumption that  $j_1 \in K$ . So, the term  $(1 - \alpha_{j_1} - \beta_{j_1})\zeta$  appearing in equation 7.4 can be written as  $\frac{H}{2^D}$  for some odd (not necessarily positive) integer  $H$  and some positive integer  $D \geq M'$ . Furthermore,  $\beta_{j_1} = \frac{H'}{2^{D'}}$  for some odd (not necessarily positive) integer  $H'$  and a positive integer  $D' \leq r(|x|)$ . Now we have

$$Q'[\pi] = \frac{H'}{2^{D'}} + \frac{H}{2^D} = \frac{H'2^{D-D'} + H}{2^D} = 0.$$

Since both  $H'$  and  $H$  are odd integers, the numerator  $H'2^{D-D'} + H$  is not 0 unless  $D = D'$ . So,  $D = D'$ . Note that  $D \geq M'$  since  $\zeta$  is the product of  $M'$  terms, none of which belong to  $\{-1, 0, 1\}$ . So, if  $M' > r(|x|)$ , clearly,  $D \neq D'$ . Thus,  $M' \leq r(|x|)$ . Thus,  $(\star\star)$  holds.

Now we consider the complexity of testing  $(\star)$  and  $(\star\star)$ . Define

$$T_\alpha = \left\{ \langle x, i, H \rangle \mid 1 \leq i \leq 2^{t(|x|)} \wedge -2^{r(|x|)} \leq H \leq 2^{r(|x|)} \wedge \right. \\ \left. \text{and the value of } \alpha_i \text{ for the input } x \text{ is } \frac{H}{2^{r(|x|)}} \right\}$$

and

$$T_\beta = \left\{ \langle x, i, H \rangle \mid 1 \leq i \leq 2^{t(|x|)} \wedge -2^{r(|x|)} \leq H \leq 2^{r(|x|)} \wedge \right. \\ \left. \text{and the value of } \beta_i \text{ for the input } x \text{ is } \frac{H}{2^{r(|x|)}} \right\}.$$

Then  $T_\alpha$  and  $T_\beta$  are in  $C=P$ . We will leave the task of verifying this claim to the reader.

Let  $g \in \text{GapP}$  be a function for checking the value of  $\beta_i$  in  $C=P$ , i.e., for every  $x \in \Sigma^*$ , every  $i$ ,  $1 \leq i \leq 2^{t(|x|)}$ , and every integer  $H$  in the interval  $[-2^{r(|x|)}, 2^{r(|x|)}]$ ,

$$\Pr[f(x, y_i) = \nu_{=1}] - \Pr[f(x, y_i) = \nu_{=2}] = \frac{H}{2^{r(|x|)}} \iff g(x, i, H) = 0,$$

where  $y_i$  denotes the string in  $\Sigma^{t(|x|)}$  having rank  $i$ . For each  $x \in \Sigma^*$  and each nonempty  $J \subseteq \{1, \dots, 2^{t(|x|)}\}$  having cardinality at most  $r(|x|)$ , define

$$g'(x, J) = \sum_{i \in \{1, \dots, 2^{t(|x|)}\} \setminus J} g(x, i, 0)^2.$$

Then, by parts 3 and 5 of Proposition 9.3,  $g' \in \text{GapP}$ . Then, for a given  $J$ , part 1 of  $(\star\star)$  can be tested by asking whether  $g'(x, J) = 0$ . This is a query to a  $\text{C=P}$  language.

Now consider a nondeterministic Turing machine that, on input  $x \in \Sigma^*$ , nondeterministically selects and executes one of the following two tasks:

**Task 1** Nondeterministically select  $i$ ,  $1 \leq i \leq 2^{t(|x|)}$ . Ask the oracle whether  $\alpha_i = \beta_i = 0$ . Accept  $x$  if the answer of the oracle is positive and reject  $x$  otherwise.

**Task 2** Perform the following three operations:

- Nondeterministically select  $m$ ,  $1 \leq m \leq r(|x|)$ ,  $j_1, \dots, j_m$ ,  $1 \leq j_1 < \dots < j_m \leq 2^{t(|x|)}$ , integers  $a_1, \dots, a_m$ ,  $b_1, \dots, b_m$  between  $-2^{r(|x|)}$  and  $2^{r(|x|)}$ .
- Test whether the second condition of  $(\star\star)$  holds with, for all  $i$ ,  $1 \leq i \leq m$ ,  $a_i/2^{p(|x|)}$  in place of  $\alpha_{j_i}$  and with  $\beta_{j_i}$  in place of  $b_i/2^{p(|x|)}$  for  $\beta_{j_i}$ . If the test fails, then immediately reject  $x$ .
- Ask the oracle whether the first condition of  $(\star\star)$  holds. Accept  $x$  if the answer is positive and reject  $x$  otherwise.

By the discussion in the above, a  $\text{C=P}$  oracle can answer each of the questions that are made. Obviously, the machine is polynomial-time bounded. Thus,  $L \in \text{NP}^{\text{C=P}}$ .  $\square$

## 7.4 OPEN ISSUE: How Complex Is Majority-Based Probabilistic Symmetric Bottleneck Computation?

Theorem 7.13 states that probabilistic symmetric bottleneck computation captures precisely  $\text{NP}^{\text{C=P}}$ . We define  $\text{ProbabilisticSSF}_2$  using “the exact half” as the membership criterion. Namely, for every  $x$  and every permutation of the mappings,  $x$  is a member if and only if the probability that 1 is mapped to 1 is exactly a half. What kind of class does it become if we change the definition such that the probability must be more than a half? No one knows. In fact, we don’t even know whether the majority-based class includes the “exactly-half”-based class.

## 7.5 Bibliographic Notes

Theorems 7.2 and 7.6 are due to Barrington [Bar89]. Theorem 7.8 is due to Cai and Furst [CF91].  $\oplus\text{OptP}$  was introduced by Hemachandra and

Hoene [HH91b] for the purpose of studying sets with efficient implicit membership sets. Theorem 7.10 is due to Ogiwara [Ogi94a]. Theorems 7.12 and 7.13 are due to Hemaspaandra and Ogiwara [HO97].

Branching programs were introduced in a paper by Lee [Lee59], who called them “binary-decision programs.” Later the concept was studied in the Master’s thesis of Masek [Mas76] under the name of “decision graphs.” Borodin et al. [BDFP86] and Chandra, Fortune, and Lipton [CFL85] questioned whether simple functions such as the parity function can be computed by polynomial-size, bounded-width branching programs. Barrington [Bar89] positively resolved the question, and this is Theorem 7.2. Barrington’s earlier work [Bar85] characterizes the power of width-3, permutation only polynomial-size branching programs. In [Bar89] Barrington shows that the languages recognized by polynomial-size, permutation-only, branching programs of width less than five are  $AC^0$ -reducible to a mod function.

A further extension of Theorem 7.6 is proven by Barrington [Bar89]. Here the membership is determined by examining whether the product belongs to a set of predetermined elements of a monoid. More precisely, a program over a monoid consists of its instructions and a list of permissible product values, which is a list of elements in the monoid. The program accepts an input  $x$  if and only if the product of the monoid elements generated by the input  $x$  according to the program belongs to the list provided. This is the concept called nonuniform deterministic finite automata (NUDFA) [Bar89] over a finite monoid. Recognition by NUDFA extends the concept of language recognition as word problems over a monoid (translate each input symbol to an element in a monoid and compute the product of the elements). An immediate observation that follows from Theorem 7.6 is that the class  $NC^1$  is equal to the class of languages that are recognized by a family of polynomial-size NUDFA programs on some monoid.

One wonders whether a fine classification of languages in  $NC^1$  can be obtained by restricting the monoid in polynomial-size programs for NUDFA. A monoid is *aperiodic* if every element  $m$  in it satisfies an equation of the form  $m^t = m^{t+1}$  for some  $t \geq 0$ . A monoid is *solvable* if every group contained in it is solvable. For a group  $G$ , its *lower central series* is a sequence of groups  $G_0, G_1, \dots$  defined as follows:  $G_0 = G$  and for every  $i \geq 1$ ,  $G_i$  is the group generated by  $\{h_2^{-1} \circ h_1^{-1} \circ h_2 \circ h_1 \mid h_1 \in G_{i-1} \wedge h_2 \in G\}$ . A group is *nilpotent* if its lower central series converges to the trivial group. Building upon earlier work of Thérien [Thé81], Barrington and Thérien [BT88] show that  $AC^0$  is the class of languages that are recognized by a family of polynomial-size NUDFA programs on some aperiodic monoid and that  $ACC$  is the class of languages that are recognized by a family of polynomial-size NUDFA programs on some solvable monoid. Barrington, Straubing, and Thérien [BST90] show that a language is recognized by a family of polynomial-size programs for NUDFA on a nilpotent group if and only if it is represented by a family of polynomials of constant degree over a direct product of cyclic rings.

The concept of bottleneck Turing machines was introduced in the paper by Cai and Furst [CF91]. In addition to the complete characterization of  $SF_5$  (Theorem 7.8), Cai and Furst observe that  $SF_2$  includes  $\Delta_2^P$  and asked whether any of  $SF_2$ ,  $SF_3$ , and  $SF_4$  contains the polynomial hierarchy. In [Ogi94a] Ogihara obtained upper and lower bounds for these classes, including Theorem 7.10. Ogihara's upper and lower bounds are in the class family  $MOD_6PH$ , where  $MOD_6PH$  is the smallest family  $\mathcal{F}$  of complexity classes satisfying the following conditions: (i)  $P \in \mathcal{F}$  and (ii) for every  $C \in \mathcal{F}$ ,  $NP^C \in \mathcal{F}$ ,  $coNP^C \in \mathcal{F}$ ,  $Mod_2P^C \in \mathcal{F}$ , and  $Mod_3P^C \in \mathcal{F}$ . Ogihara shows that  $SF_4 \supseteq (\Sigma_2^P)^{\oplus P}$ , which implies, by Toda's Theorem 4.12, that  $PH \subseteq SF_4$ , answering the question raised by Cai and Furst. Beigel and Straubing [BS95] also showed some insights into how the upper and lower bounds shown in [Ogi94a] can be tightened. The exact characterizations of  $SF_3$  and  $SF_4$  and special cases of  $SF$  classes are given by Hertrampf ([Her97], see also [Her00]). Hertrampf et al. [HLS<sup>+</sup>93] show that the characterizations of  $AC^0$  and of  $ACC$  proven by Barrington and Thérien [BT88] can be translated into polynomial-time uniform classes to characterize  $PH$  and  $MOD_6PH$ .

There is another application of Theorem 7.2. Let  $k \geq 1$  be an integer. A language  $L$  is called *k-locally self-reducible* if there exists a polynomial time oracle Turing machine  $M$  that decides  $L$  with oracle  $L$  such that, for every input  $x$  and every query  $y$  of  $M$  on input  $x$ , the lexicographic order of  $y$  is between that of  $x$  minus one and that of  $x$  minus  $k$ ; i.e., the membership of only  $k$  predecessors of  $x$  in the lexicographic order can be asked. Beigel and Straubing [BS95] show that for every  $k$ , all  $k$ -locally self-reducible sets are in  $PSPACE$ , and that, while all 2-locally self-reducible sets belong to  $MOD_6PH$ , some 3-locally self-reducible sets are  $PSPACE$ -complete. They also show that there is a  $PSPACE$ -complete 6-locally self-reducible set whose self-reduction is many-one.

The concept of symmetric bottleneck Turing machines was introduced by Hemaspaandra and Ogihara [HO97]. They observe that for every  $k \geq 2$  and every language  $L$  in  $SSF_k$ ,  $L$  is  $\leq_m^p$ -reducible to a language in  $coMod_kP$  by a function that is polynomial-time computable with an oracle in  $PH$ . Hertrampf ([Her99], see also [Her00]) obtained an exact characterization of  $SSF$  classes.

Based on Barrington's  $S_5$  trick, Ben-Or and Cleve [BOC92] showed that algebraic formulas over any ring can be evaluated by straight-line programs using just three registers. Caussinus et al. [CMTV98] use this result to obtain a characterization of the class  $GapNC^1$  in terms of bounded-width branching programs.

Theorem 7.2 can be applied to quantum computation. (For a textbook on quantum computation, see [Gru99].) Ambainis, Schulman, and Vazirani [ASV00] show that width-5 permutation branching programs can be simulated by a quantum computer with three qubits one of which is in a pure initial state and two others are in a completely mixed (random) start-



ing state. Thus,  $\text{NC}^1$  can be computed by quantum computers with all but one qubit in a completely random starting state.



## 8. The Random Restriction Technique

Oracle construction is a major tool for studying questions about complexity classes. Suppose we find an oracle relative to which a complexity-theoretic property  $Q$  holds and another oracle relative to which  $Q$  does not hold. Then we can conclude that settling the question of whether  $Q$  holds without assumption is very tough, in the sense that proof techniques that can be relativized, such as those based on Turing-machine computation, cannot on their own successfully resolve whether  $Q$  holds.

It is often the case that one of the two kinds of oracles—oracles making  $Q$  hold and oracles making  $Q$  fail to hold—is easy to construct, while the other kind is more difficult to construct. An example of this type is the question of whether  $P$  equals  $NP$ . If we relativize this question by any  $PSPACE$ -complete oracle, then the two classes both become  $PSPACE$ , and thus equality holds. On the other hand, the existence of an oracle for which the equality does not hold is typically demonstrated by a diagonalization argument that is more complicated than the few-line proof of the equality.

The focus of this chapter is an oracle construction based on impossibility results about boolean circuits. These impossibility results are proven by randomly fixing the input bits (and so are called *the random restriction technique*). The chapter is organized as follows. Section 8.1 introduces the random restriction technique and presents the first circuit lower bound proven by the technique: Constant depth, polynomial-size circuits cannot compute parity. Section 8.2 presents an exponential-size lower bound for parity and, based on that bound, constructs a world in which  $PH \neq PSPACE$ . Section 8.3 is an interlude. We prove that a probabilistic experiment yields a world separating  $PH$  from  $PSPACE$  with probability one. Section 8.4 is an application of the technique to the question of whether the polynomial hierarchy is infinite.

### 8.1 GEM: The Random Restriction Technique and a Polynomial-Size Lower Bound for Parity

#### 8.1.1 The Idea

First let us briefly sketch the idea behind the technique. Let  $f$  be a function and let  $C$  be a depth- $k$  circuit. We wish to prove that  $C$  does not compute

$f$ . We assume that every downward path in  $C$  from its output gate to an input gate has length  $k$ . Then we divide the gates into  $k + 1$  levels,  $0, \dots, k$ , where the input gates are at level 0, and, for each  $k \geq 1$ , the level  $k$  nodes are those that take inputs from level  $k - 1$ . Suppose that we assign the values 0 and 1 to some of the variables and *restrict* the inputs of both  $C$  and  $f$  to those consistent with the assignment. Some restrictions may force all depth-2 subcircuits of  $C$  to depend on a small number of variables. If there is such a restriction, then all the depth-2 subcircuits can be simplified so that their top gates are the same as the gates in  $C$  at the third level. This simplification will produce either two consecutive levels of  $\wedge$ 's or two consecutive levels of  $\vee$ 's, which can be collapsed to yield an equivalent depth- $(k - 1)$  circuit.

We require that for any restriction,  $f$  depend on all the remaining variables. We search for a restriction sequence that collapses the depth of  $C$  to two and simultaneously forces all the depth-1 subcircuits of  $C$  to have fan-in less than the number of remaining variables. If there is such a sequence, then we can use one more restriction to reduce  $C$  to a constant while keeping  $f$  nontrivial. Now combine all the restrictions that we have identified into one big restriction. Under this combined restriction,  $C$  and  $f$  are different, which shows that  $C$  and  $f$  were different from the very beginning.

Now the question is how to find restrictions with desired properties. It may be very difficult to describe precisely what restrictions will do the job. So we attempt a nonconstructive approach. We introduce probability distributions on restrictions and prove that good restrictions appear with nonzero probability, which guarantees that at least one exists.

## 8.1.2 Preliminaries

We need some preparation.

**8.1.2.1 Unbounded Fan-in Boolean Circuits.** An unbounded fan-in circuit over a set of variables  $\Xi$  is a labeled, directed acyclic graph  $C$  with the following properties:

1. There is a unique node with no outgoing edges, called the *output gate*.
2. Each node with no incoming edges is labeled either by  $x$  or by  $\bar{x}$  for some  $x \in \Xi$ . Such a node is called an *input gate* as well as a *leaf*.
3. Each nonleaf node is labeled either by  $\wedge$  or by  $\vee$ .
4. Every two adjacent nodes are labeled differently.
5. All paths from leaves to the output node have the same length.

Note that the above properties imply that our circuit consists of alternating levels of  $\wedge$  gates and  $\vee$  gates. We assign numbers to the levels of such stratified circuits in a natural way: The input level is level 0, and, for each  $k \geq 1$ , the level  $k$  nodes are those that take inputs from level  $k - 1$ .

For a circuit  $C$ , the size of  $C$ , denoted by  $size(C)$ , is its number of nonleaf nodes. The depth of  $C$ , denoted by  $depth(C)$ , is the length of the paths from

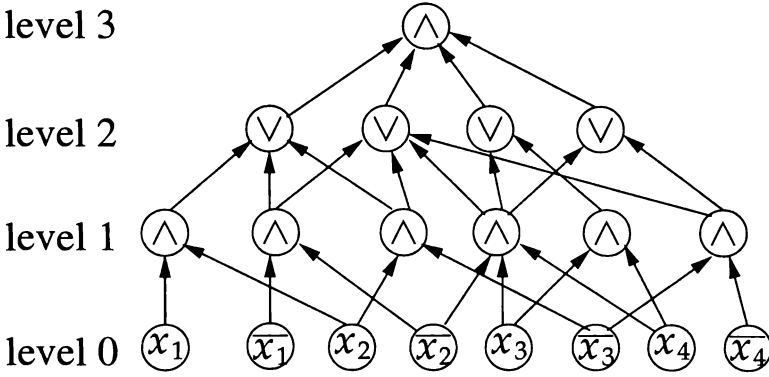


Fig. 8.1 A depth-3 circuit

its leaves to its output node. For a nonleaf  $v$ , the fan-in of  $v$ , denoted by  $\text{fan-in}(v)$ , is the number of edges coming into  $v$ . For simplicity, for a depth-1 circuit  $C$ , we write  $\text{fan-in}(C)$  to denote the fan-in of the output node of  $C$ .

Let  $x_1, \dots, x_n$  be a fixed enumeration of the variables in  $\Xi$ . Let  $a = (a_1, \dots, a_n) \in \{0, 1\}^n$ . The output of  $C$  on input  $a$  is inductively evaluated as follows:

1. If a leaf  $v$  is labeled by  $x_i$  (respectively,  $\bar{x}_i$ ) for some  $i$ ,  $1 \leq i \leq n$ , then the output of  $v$  is 1 if and only if  $a_i = 1$  (respectively,  $a_i = 0$ ).
2. For each nonleaf  $v$  labeled by  $\wedge$ ,  $v$  outputs 1 if and only if all its input signals are 1.
3. For each nonleaf  $v$  labeled by  $\vee$ ,  $v$  outputs 1 if and only if at least one of its input signals is 1.
4. The circuit  $C$  outputs 1 if and only if the output node of  $C$  outputs 1.

We assume that no depth-1 circuit takes as input two conflicting literals, i.e.,  $x$  and  $\bar{x}$  for some variable  $x$ .

**8.1.2.2 Restrictions.** Let  $\Xi$  be a finite set of boolean variables. A *restriction* on  $\Xi$  is a partial assignment of  $\Xi$  to boolean values. Formally, a restriction on  $\Xi$  is a mapping  $\rho$  of  $\Xi$  to  $\{0, 1, *\}$ , where  $\rho(x) = 0$  (respectively,  $\rho(x) = 1$ ) indicates that  $x \in \Xi$  is assigned the value 0 (respectively, 1) and  $\rho(x) = *$  indicates that  $x$  is not assigned a value, i.e., is preserved as a variable.

For a restriction  $\sigma$ ,  $\sigma^{-1}(*)$  (respectively,  $\sigma^{-1}(0)$  and  $\sigma^{-1}(1)$ ) denotes the set of all  $x \in \Xi$  to which  $\sigma$  assigns  $*$  (respectively, 0 and 1), and  $\sigma^{-1}(\{0, 1\})$  denotes  $\sigma^{-1}(0) \cup \sigma^{-1}(1)$ .

Let  $F$  be a function over the variables of  $\Xi$ , and let  $\rho$  be a restriction on  $\Xi$ . We assume that there is a natural order among the elements of  $\Xi$  (that is, it has first, second, etc. elements). Let  $Y = \rho^{-1}(*)$  and let  $m = |Y|$ . Let  $y_1, \dots, y_m$  be the enumeration of all the elements of  $Y$  according to the

natural order among the elements of  $\Xi$ . Then  $F$  under restriction  $\rho$ , denoted by  $F[\rho]$ , is the function  $G$  over  $Y$  such that, for every  $b = b_1 \cdots b_m \in \{0, 1\}^m$ ,  $G$  maps  $b$  to the value of  $F$  when the variables in  $\Xi - Y$  are given values according to  $\rho$  and when for each  $i$ ,  $1 \leq i \leq m$ ,  $y_i$  receives the value  $b_i$ .

Given two restrictions  $\rho_1$  and  $\rho_2$ , their product  $\rho_1 \rho_2$  is the restriction  $\rho'$  defined as follows: For all  $x \in \Xi$ ,

$$\rho'(x) = \begin{cases} * & \text{if } \rho_1(x) = \rho_2(x) = *, \\ 1 & \text{if } \rho_1(x) = 1 \vee (\rho_1(x) = * \wedge \rho_2(x) = 1), \\ 0 & \text{if } \rho_1(x) = 0 \vee (\rho_1(x) = * \wedge \rho_2(x) = 0). \end{cases}$$

Let  $\sigma, \tau$  be restrictions on  $\Xi$ . We say that  $\sigma$  and  $\tau$  are *disjoint* if  $\sigma^{-1}(\{0, 1\}) \cap \tau^{-1}(\{0, 1\}) = \emptyset$ . We say that a restriction  $\sigma$  *subsumes* a restriction  $\tau$  if  $\sigma^{-1}(1) \supseteq \tau^{-1}(1)$  and  $\sigma^{-1}(0) \supseteq \tau^{-1}(0)$ . We write  $\sigma \supseteq \tau$  to denote that  $\sigma$  subsumes  $\tau$ .

For a boolean circuit  $C$  and a restriction  $\rho$ ,  $C[\rho]$  is obtained by simplifying the circuit according to  $\rho$ , working from the input level towards the output level as follows:

- At the input level for each variable  $x$  such that  $\rho(x) \in \{0, 1\}$ , we replace  $x$  by  $\rho(x)$  and  $\bar{x}$  by  $1 - \rho(x)$ .
- At an  $\vee$ -level, for each gate  $g$  at that level, we check whether it has an input fixed to 1. If so, the gate  $g$  is replaced by the constant 1. Otherwise, we eliminate all the 0 inputs to  $g$ . If there is no input left to  $g$ , then we replace  $g$  by the constant 0.
- At an  $\wedge$ -level, for each gate  $g$  at that level, we check whether it has an input fixed to 0. If so, the gate  $g$  is replaced by the constant 0. Otherwise, we eliminate all the 1 inputs to  $g$ . If there is no input left to  $g$ , then we replace  $g$  by the constant 1.

For a function  $F$  we write  $F \equiv 1$  (respectively,  $F \equiv 0$ ) to denote that  $F$  acts as the constant 1 function (respectively, the constant 0 function).

Let  $\Xi$  be a set of variables and let  $p$ ,  $0 < p < 1$ , be a real number. Then  $R_p^\Xi$  is the distribution on the restrictions on  $\Xi$  defined as follows: For each variable  $x \in \Xi$ ,

$$\rho(x) = \begin{cases} * & \text{with probability } p, \\ 0 & \text{with probability } \frac{1-p}{2}, \\ 1 & \text{with probability } \frac{1-p}{2}. \end{cases} \quad (8.1)$$

**8.1.2.3 Minterms and the Parity Function.** We say that a restriction  $\sigma$  is a *minterm* of a function  $F$  if  $F[\sigma] \equiv 1$  and for any restriction  $\sigma' \subsetneq \sigma$ ,  $F[\sigma'] \neq 1$ . Thus the minterm of the constant 1 function is the empty restriction and the minterm of the constant 0 function is undefined. For a restriction  $\sigma$ , the *size* of  $\sigma$ , denoted by  $|\sigma|$ , is the number of  $x \in \Xi$ , satisfying  $\sigma(x) \neq *$ .

The size of the smallest minterm of  $F$  is denoted by  $\text{MIN}(F)$ . If  $F$  is the constant 1 function clearly  $\text{MIN}(F) = 0$ . If  $F$  is the constant 0 function, we define  $\text{MIN}(F) = 0$ . Each minterm  $\sigma$  can be viewed as the smallest  $\wedge$ -gate that outputs 1 if and only if all the value assignments in  $\sigma$  are given. So for any function  $f$ , if  $C$  is the smallest  $\vee$ - $\wedge$  circuit (i.e., an  $\vee$  of  $\wedge$  gates) that computes  $f$  then each  $\wedge$ -gate of  $C$  is a minterm.

The  $n$ -ary parity function,  $\pi_n$ , is the function that maps each  $n$ -bit input to the modulo 2 count of the number of 1s in the input bit. Note that for every  $n$  there are precisely  $2^{n-1}$  minterms of  $\pi_n$ , each of size  $n$ .

We say that a family of circuits  $\{C_n\}_{n \geq 1}$  computes the parity function if, for every  $n \geq 1$ ,  $C_n$  computes  $\pi_n$ .

### 8.1.3 The Main Theorem

**Theorem 8.1** *For no  $k \geq 1$  can the parity function be computed by a family of depth- $k$ , polynomial-size circuits.*

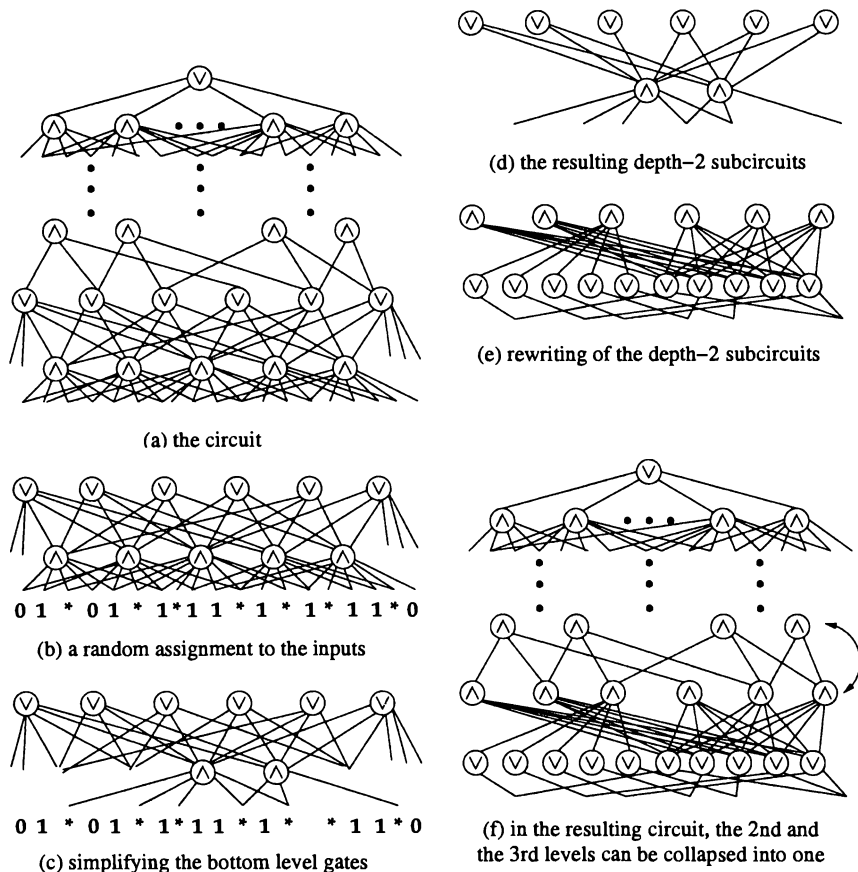
The rest of the section proves the above theorem.

**Proof of Theorem 8.1** We prove the theorem by induction on  $k$ . The base case is when  $k = 2$ . Let  $n \geq 1$  and let  $C$  be the smallest depth-2 circuit of  $n$  inputs that computes  $\pi_n$ . Suppose  $C$  is an  $\vee$ - $\wedge$  circuit. Then we claim that each  $\wedge$ -gate of  $C$  has fan-in  $n$  and for each  $i$ ,  $1 \leq i \leq n$ , takes exactly one of  $x_i$  and  $\bar{x}_i$  as input. To see why, suppose that there is an  $\wedge$ -gate, say  $g$ , such that, for some  $i$ ,  $1 \leq i \leq n$ ,  $g$  takes both  $x_i$  and  $\bar{x}_i$  as input. Then, for every input  $x_1, \dots, x_n$ ,  $g$  outputs 0 because  $x_i \wedge \bar{x}_i = 0$  regardless of whether  $x_i = 0$  or  $x_i = 1$ . Then, since the output gate of  $C$  is an  $\vee$ -gate,  $g$  can be removed from  $C$ . Now assume that there is an  $\wedge$ -gate, say  $g$ , having fan-in less than  $n$ . Then there is some  $i$ ,  $1 \leq i \leq n$ , such that neither  $x_i$  nor  $\bar{x}_i$  is an input to  $g$ . As we have already eliminated gates in  $C$  with input from both a variable and its negation, there is some  $a = (a_1, \dots, a_n)$  such that  $g$  on input  $a$  outputs 1. Let  $a'$  be  $a$  with the  $i$ th bit flipped. Then  $g$  outputs 1 on  $a'$ , too. Since the output of  $C$  is an  $\vee$ -gate,  $C$  outputs 1 both on  $a$  and on  $a'$ . Since the parity of  $a$  is different from the parity of  $a'$ ,  $C$  does not compute  $\pi_n$ . This is a contradiction.

The above observation implies that for each  $\wedge$ -gate of  $C$ , there is only one input for which the gate outputs 1. Since there are  $2^{n-1}$  inputs for which  $C$  needs to output 1,  $C$  needs to have at least  $2^{n-1}$  many  $\wedge$ -gates, which implies that the size of  $C$  is at least  $2^{n-1} + 1$ .

If  $C$  is an  $\wedge$ - $\vee$  circuit, construct  $C'$  from  $C$  by interchanging the labels  $\wedge$  and  $\vee$  and interchanging, for each variable  $x$ , the labels  $x$  and  $\bar{x}$ . Then  $C'$  has the same depth and size as  $C$  does and computes the complement of  $\pi_n$ . The complement has  $2^{n-1}$  minterms so  $\text{size}(C') \geq 2^{n-1} + 1$ .

For the induction step, let  $k \geq 3$  and suppose that the claim holds for all  $k'$ ,  $2 \leq k' \leq k - 1$ . Assume, by way of contradiction, that for some integer  $l \geq 1$ , there is a depth- $k$ , size- $n^l$  circuit family  $\{C_n\}_{n \geq 1}$  that computes



**Fig. 8.2** The random restriction technique

the parity function. Then we show that there is a family of depth- $(k - 1)$ , polynomial-size circuits for the parity function, a contradiction. We derive the contradiction in three phases. In Phase 1, we use a restriction to significantly reduce the bottom fan-in; in Phase 2, we use a restriction on depth-2 circuits at the bottom to reduce the number of inputs that each of these circuits depends on; in Phase 3, we merge level 2 and 3 gates. Figure 8.2 illustrates how the reduction proceeds.

**Phase 1** Let  $\alpha = 4l + 1$ . For each  $n \geq 1$ , pick a random restriction  $\rho$  under distribution  $R_p^{\Xi_n}$  with  $p = n^{-1/2}$ , where  $\Xi_n$  is the set of variables of  $C_n$ .



Let  $n \geq 1$  be fixed and let  $S_1, \dots, S_m$  be an enumeration of all depth-1 subcircuits of  $C_n$ . We say that  $\rho$  *succeeds* if

- $\|\rho^{-1}(\ast)\| \geq \frac{\sqrt{n}}{2}$  and
- for every  $i$ ,  $1 \leq i \leq m$ , if  $S_i[\rho] \neq 0$  and  $S_i[\rho] \neq 1$ , then  $\text{fan-in}(S_i[\rho]) \leq \alpha$ .

Otherwise, we say that  $\rho$  *fails*. Define  $Q$  to be the probability that  $\|\rho^{-1}(\ast)\| < \frac{\sqrt{n}}{2}$  and for each  $i$ ,  $1 \leq i \leq m$ , define  $P_i$  to be the probability that  $S_i[\rho] \neq 0$ ,  $S_i[\rho] \neq 1$ , and  $\text{fan-in}(S_i[\rho]) > \alpha$ . Then the probability that  $\rho$  fails is at most

$$Q + P_1 + \dots + P_m.$$

We will obtain an upper bound on each of these terms.

In order to evaluate  $Q$ , let  $E$  and  $V$  respectively be the expected value and the variance of  $\|\rho^{-1}(\ast)\|$ . Then  $E = np = \sqrt{n}$  and  $V = np(1-p) \leq \sqrt{n}$ . Chebyshev's Inequality (Lemma 6.22) states that if a random variable  $z$  has expectation  $E$  and variance  $V$ , then for every  $d > 0$ , the probability that  $z$  is less than  $E - d$  is at most  $\frac{V}{d^2}$ . By plugging in  $d = \sqrt{n}/2$ ,  $E = \sqrt{n}$ , and  $V \leq \sqrt{n}$ , we have

$$Q \leq \frac{V}{d^2} \leq \frac{4}{\sqrt{n}} = \mathcal{O}(n^{-1/2}).$$

In order to evaluate  $P_1, \dots, P_m$ , fix  $i$ ,  $1 \leq i \leq m$ . Let  $t = \text{fan-in}(S_i)$  and let  $u_1, \dots, u_t$  be an enumeration of all the input literals of  $S_i$ . Let  $b = 0$  if  $S_i$  is an  $\wedge$  circuit and let  $b = 1$  otherwise. If for some  $j$ ,  $1 \leq j \leq t$ ,  $\rho(u_j) = b$  then  $S_i[\rho] \equiv b$ . We divide the analysis into two cases depending on  $t$ .

First suppose that  $t \geq \alpha \ln n$ . If  $\text{fan-in}(S_i[\rho]) > \alpha$  then for every  $j$ ,  $1 \leq j \leq t$ ,  $\rho(u_j) \neq b$ . For each  $j$ ,  $1 \leq j \leq t$ , the probability that  $\rho(u_j) \neq b$  is  $(1+p)/2 = \frac{1}{2} + \frac{1}{2\sqrt{n}}$ . So,  $P_i$  is at most

$$\left(\frac{1}{2} + \frac{1}{2\sqrt{n}}\right)^t \leq \left(\frac{1}{2} + \frac{1}{2\sqrt{n}}\right)^{\alpha \ln n} = o(n^{-(1-\epsilon)\alpha})$$

for every constant  $\epsilon > 0$ . So,

$$P_i = o(n^{-4l}) = o(n^{-2l}).$$

Next suppose that  $t < \alpha \ln n$ . If  $\text{fan-in}(S_i[\rho]) > \alpha$  then  $\rho(u_j) = \ast$  for more than  $\alpha$  integers  $j$ ,  $1 \leq j \leq t$ ; and  $\rho(u_j) = 1 - b$  for all the other  $j$ 's. Thus  $P_i$  is at most

$$\binom{t}{\alpha} \left(\frac{1}{\sqrt{n}}\right)^\alpha \leq \left(\frac{t}{\sqrt{n}}\right)^\alpha < \left(\frac{\alpha \ln n}{\sqrt{n}}\right)^\alpha = o(n^{-(\frac{1}{2}-\epsilon)\alpha})$$

for every constant  $\epsilon > 0$ . So,  $P_i = o(n^{-2l})$ .

Now, since  $m \leq \text{size}(C_n) \leq n^l$ , the probability that  $\rho$  fails is at most

$$\mathcal{O}(n^{-1/2}) + o(n^{-2l})n^l = \mathcal{O}(n^{-1/2}) + o(n^{-l}) = \mathcal{O}(n^{-1/2}).$$

Hence, there exists some  $n_1 > 0$  such that for every  $n \geq n_1$  a successful restriction  $\rho$  exists. For each  $n \geq n_1$  pick a successful restriction  $\rho_n$  and define  $D_n = C_n[\rho_n]$ ,  $Y_n = \rho_n^{-1}(*)$ , and  $\mu_n = |Y_n|$ . We obtain a new family of circuits  $\{D_n\}_{n \geq n_1}$  satisfying the following conditions for all  $n \geq n_1$ :

1.  $D_n$  computes either the parity function or the complement of the parity (depending on the parity of  $|\rho_n^{-1}(1)|$ ) on the set  $Y_n$  of  $\mu_n$  variables, where  $\mu_n = \Omega(\sqrt{n})$ .
2.  $\text{size}(D_n) \leq n^l = \mathcal{O}((\mu_n)^{2l})$ .
3. Each depth-1 subcircuit of  $D_n$  is of fan-in at most  $\alpha$ .

**Phase 2** For each  $n \geq n_1$ , we pick a random restriction  $\sigma$  under  $R_q^{Y_n}$  with  $q = (\mu_n)^{-1/2}$ . We say that  $\sigma$  *succeeds* if

- $|\sigma^{-1}(*)| \geq \frac{\sqrt{\mu_n}}{2}$  and
- every depth-2 subcircuit of  $D_n[\sigma]$  is dependent on at most  $\beta_\alpha$  variables, where the sequence  $\beta_1, \beta_2, \dots$  will be defined below.

Otherwise, we say that  $\sigma$  *fails*. Let  $n \geq n_1$  be fixed. Let  $Q$  be the probability that  $|\sigma^{-1}(*)| < \frac{\sqrt{\mu_n}}{2}$ . As in Phase 1, by Chebyshev's Inequality (Lemma 6.22)

$$Q = \mathcal{O}((\mu_n)^{-1/2}) = o(n^{-1/4}).$$

To bound the probability that there exists a depth-2 subcircuit of  $D_n[\sigma]$  that depends on more than  $\beta_\alpha$  variables, we need the following lemma.

Define  $\gamma = 6l + 1$  (recall that the size of the circuit  $C_n$  is bounded by  $n^l$ ),  $\beta_1 = \gamma$ , and for each  $d \geq 2$ ,  $\beta_d = \gamma + 2^\gamma \beta_{d-1}$ .

**Lemma 8.2** *For every  $d \geq 1$ , there exists a constant  $n_2 \geq n_1$  such that, for all  $n \geq n_2$ , and for all depth-2 subcircuits  $S$  of  $D_n$ , the following is true: If every depth-1 subcircuit of  $S$  has fan-in at most  $d$ , then with probability at least  $1 - o(\mu_n^{-3l})$ ,  $S[\sigma]$  depends on at most  $\beta_d$  variables.*

**Proof of Lemma 8.2** Suppose that depth-2 subcircuits of  $D_n$  are  $\vee$ - $\wedge$  circuits. The proof is by induction on  $d$ . For the base case, let  $d = 1$ . Let  $n \geq n_1$  be fixed and let  $S$  be a depth-2 subcircuit of  $D_n$ . Suppose that all the depth-1 subcircuits of  $D_n$  have fan-in 1. We can eliminate every level-1 gate  $g$  by directly connecting the input literal of  $g$  to each gate that receives signal from  $g$ . This reduces  $S$  to a depth-1 circuit. Let  $t = \text{fan-in}(S)$ . By applying the analysis from the previous phase with  $\mu_n$  in place of  $n$  and  $\beta_1$  in place of  $\alpha$ , we have:

- If  $t > \beta_1 \ln \mu_n$ , then  $\text{fan-in}(S[\sigma]) > \beta_1$  with probability at most

$$\left(\frac{1}{2} + \frac{1}{2\sqrt{\mu_n}}\right)^t \leq \left(\frac{1}{2} + \frac{1}{2\sqrt{\mu_n}}\right)^{\beta_1 \ln \mu_n} = o(\mu_n^{-(1-\epsilon)\beta_1})$$

for every constant  $\epsilon > 0$ . So, the probability in question is  $o(\mu_n^{-3l})$ .

- If  $t \leq \beta_1 \ln \mu_n$ , then the probability that  $\text{fan-in}(S[\sigma]) > \beta_1$  is at most

$$\binom{t}{\beta_1} \left( \frac{1}{\sqrt{\mu_n}} \right)^{\lceil \beta_1 \rceil} \leq \left( \frac{t}{\sqrt{\mu_n}} \right)^{\beta_1} < \left( \frac{\beta_1 \ln \mu_n}{\sqrt{\mu_n}} \right)^{\beta_1} = o(\mu_n^{-(\frac{1}{2}-\epsilon)\beta_1})$$

for every constant  $\epsilon > 0$ . So, the probability in question is  $o(\mu_n^{-3l})$ .

Thus, the probability that  $S[\sigma]$  is dependent on more than  $\beta_1$  variables is  $o(\mu_n^{-3l})$ . Thus the claim holds for  $d = 1$ .

For the induction step, let  $d \geq 2$ . Let  $b = 3l(3^d)$ . Let  $S$  be any depth-2 subcircuit of  $D_n$ . Call depth-1 subcircuits  $F_1, \dots, F_t$  of  $S$  *disjoint* if no distinct two of them depend on a common variable. Let  $r$  be the largest  $t$  such that there are  $t$  disjoint depth-1 subcircuits of  $S$ . Let  $F_1, \dots, F_r$  be such  $r$  disjoint depth-1 subcircuits of  $S$  and  $G_1, \dots, G_s$  be an enumeration of all the remaining depth-1 subcircuits of  $S$ . Then

$$S = \left( \bigvee_{i=1}^r F_i \right) \vee \left( \bigvee_{i=1}^s G_i \right).$$

Here we may assume that  $r < b \ln \mu_n$ . To see why, suppose  $r \geq b \ln \mu_n$ . Since  $\beta_d \geq 1$ , if  $S[\sigma]$  depends on more than  $\beta_d$  variables then  $S[\sigma] \neq 1$ , and if  $S[\sigma] \neq 1$ , none of  $F_1, \dots, F_r$  is reduced to the constant 1. For every  $j$ ,  $1 \leq j \leq r$ ,  $\text{fan-in}(F_j) \leq d$ , so the probability that  $F_j[\sigma] \equiv 1$  is at least  $(\frac{1}{2} - \frac{1}{2\sqrt{\mu_n}})^d$ , and this is  $\omega(3^{-d})$ . Thus the probability that  $F_j[\sigma] \neq 1$  for all  $j$ ,  $1 \leq j \leq r$ , is at most

$$(1 - 3^{-d})^{b \ln \mu_n} = (1 - 3^{-d})^{3l(3^d) \ln \mu_n} = o(2^{-3l \ln \mu_n}) = o(\mu_n^{-3l}).$$

So the probability that  $S[\sigma]$  depends on more than  $\beta_d$  variables is  $o(\mu_n^{-3l})$ .

Thus we can assume that  $r < b \ln \mu_n$ . Let  $H$  be the set of all variables  $x$  on which some  $F_j$  is dependent. Since  $F_j$ 's are disjoint, the distribution  $R_q^{Y_n}$  is identical to that of the products  $\sigma_1 \sigma_2$ , where  $\sigma_1$  is subject to  $R_q^H$  and  $\sigma_2$  is subject to  $R_q^{(Y_n - H)}$ . The probability that  $\|\sigma_1^{-1}(\sigma)\| > \gamma$  is at most

$$\binom{r}{\gamma} \left( \frac{1}{\sqrt{\mu_n}} \right)^\gamma \leq (b \ln \mu_n)^\gamma \left( \frac{1}{\sqrt{\mu_n}} \right)^\gamma = \left( \frac{b \ln \mu_n}{\sqrt{\mu_n}} \right)^\gamma = o(\mu_n^{-(\frac{1}{2}-\epsilon)\gamma})$$

for every constant  $\epsilon > 0$ . So, the probability that  $\|\sigma_1^{-1}(\sigma)\| > \gamma$  is  $o(\mu_n^{-3l})$ .

Fix  $\sigma_1$  under  $R_q^H$  such that  $\|\sigma_1^{-1}(\sigma)\| \leq \gamma$ . Let  $S' = S[\sigma_1]$ ,  $H' = \sigma_1^{-1}(H)$ , and  $h = \|H'\|$ . Let  $a_1, \dots, a_{2^h}$  be an enumeration of all possible assignments to the variables in  $H'$ . For each  $i$ ,  $1 \leq i \leq 2^h$ , let  $A_i$  be the depth-1  $\vee$  circuit that checks whether the input assignment to  $H'$  is different from  $a_i$  and let

$$S_i = A_i \vee (S' \upharpoonright a_i).$$

Let  $v = (v_1, \dots, v_h)$  denote the variables of  $H'$ . For every  $i$ ,  $1 \leq i \leq 2^h$ ,  $S_i = S' \upharpoonright a_i$  if  $v = a_i$  and  $S_i = 1$  otherwise. Thus,  $S' = \bigwedge_{i=1}^{2^h} S_i$ . For every

$i$ ,  $1 \leq i \leq 2^h$ ,  $A_i$  can be viewed as an  $\vee\text{-}\wedge$  circuit of bottom fan-in 1. For every  $j$ ,  $1 \leq j \leq s$ ,  $G_j$  has fan-in at most  $d$  and, since  $F_1, \dots, F_r$  is maximally disjoint,  $G_j$  is dependent on at least one variable in  $H$ . For every  $i$ ,  $1 \leq i \leq 2^h$ ,  $a_i^{-1}(\{0, 1\}) = \{v_1, \dots, v_h\}$ , so  $S_i$  is an  $\vee\text{-}\wedge$  circuit all of whose depth-1 subcircuits have fan-in at most  $d - 1$ . Then for each  $i$ ,  $1 \leq i \leq 2^h$ , the probability that  $S_i[\sigma_2]$  depends on more than  $\beta_{d-1}$  variables is  $o(\mu_n^{-3l})$ . If for every  $i$ ,  $1 \leq i \leq 2^h$ ,  $S_i[\sigma_2]$  depends on at most  $\beta_{d-1}$  variables, then  $S'[\sigma_2]$  depends on at most  $h + 2^h\beta_{d-1}$  variables, and this quantity is at most  $\beta_d$  because  $h \leq \gamma$ . So the probability that  $S'[\sigma_2]$  is dependent on more than  $\beta_d$  variables is at most

$$o(2^h \mu_n^{-3l}) = o(\mu_n^{-3l})$$

because  $h \leq \gamma$  and  $\gamma$  depends only on  $d$  and  $l$ . Thus, the probability that  $\sigma = \sigma_1\sigma_2$  reduces  $S$  to a circuit that is dependent on more than  $\beta_d$  variables is  $o(2\mu_n^{-3l}) = o(\mu_n^{-3l})$ .

In the case where  $S$  is an  $\wedge\text{-}\vee$  circuit we exchange the role of 0 and that of 1 and carry out the same analysis, except that the circuit  $A_i$  checks whether the input assignments on  $H'$  are identical to  $a_i$ , and that  $S_i = A_i \wedge (S' \upharpoonright a_i)$ . Then  $S' = \bigvee_{i=1}^{2^h} S_i$ .  $\square$  Lemma 8.2

By the above lemma, the probability that not all of the depth-2 subcircuits of  $D_n$  are forced to depend on at most  $\beta_\alpha$  variables is  $o(\mu_n^{-3l})\mu_n^{2l} = o(\mu_n^{-l})$ . Thus the probability that  $\sigma$  fails is

$$o(n^{-1/4}) + o(\mu_n^{-l}) = o(n^{-1/4}).$$

Hence, there exists some  $n_2 > n_1 > 0$  such that for every  $n \geq n_2$  there exists a successful restriction  $\sigma$ . So for each  $n \geq n_2$ , we pick a successful  $\sigma_n$  and apply it to  $D_n$  to obtain  $E_n = D_n[\sigma_n]$ . Define  $\hat{\beta} = \beta_\alpha$ . Then the following conditions hold for all  $n \geq n_2$ :

1. For some  $\nu_n = \Omega(n^{1/4})$ ,  $E_n$  computes the parity function of a set of  $\nu_n$  variables.
2.  $\text{size}(E_n) \leq n^l = \mathcal{O}((\nu_n)^{4l})$ .
3. Each depth-2 subcircuit of  $E_n$  is dependent on at most  $\hat{\beta}$  variables.

**Phase 3** Note that if a function is dependent on  $\hat{\beta}$  variables then it can be expressed as an  $\wedge\text{-}\vee$  circuit of top fan-in at most  $2^{\hat{\beta}}$  and of bottom fan-in  $\hat{\beta}$ , and alternatively as an  $\vee\text{-}\wedge$  circuit of top fan-in at most  $2^{\hat{\beta}}$  and of bottom fan-in  $\hat{\beta}$ . For each  $n \geq n_2$ , we apply one of the two conversions to each of the depth-2 subcircuits of  $E_n$  so that the level-2 and level-3 gates have an identical type. Then we collapse the levels into one thereby reducing the depth of the circuit to  $k - 1$ . The resulting circuit has size at most

$$2^{\hat{\beta}} \text{size}(E_n) \leq 2^{\hat{\beta}} 4^{4l} (\nu_n)^{4l}.$$

Thus, we obtain a family of depth- $(k - 1)$  circuits  $\{F_n\}_{n \geq n_2}$  satisfying the following conditions for all  $n \geq n_2$ :

1.  $F_n$  has  $\nu_n \geq \frac{n^{1/4}}{4}$  variables.

2.  $F_n$  computes the parity function of  $\nu_n$  variables.
3.  $\text{size}(F_n) \leq 2^{\hat{\beta}} 4^{4l} (\nu_n)^{4l}$ .

For each  $n \geq 1$ , let  $s(n)$  be the smallest  $n'$  such that  $\nu_{n'} \leq n$ . Then, for all but finitely many  $n$ ,  $s(n) \leq (4n)^4$ . If  $F_{s(n)}$  depends on more than  $n$  variables, then we assign 0 to some variables to make it dependent on exactly  $n$  inputs. Let  $G_n$  be the resulting circuit. Since  $F_{s(n)}$  computes the parity function,  $G_n$  computes the parity function of  $n$  inputs. Then

$$\text{size}(G_n) \leq 2^{\hat{\beta}} 4^{4l} (s(n))^{4l} = \mathcal{O}(n^{16l}).$$

This implies that the parity function can be computed by a family of depth- $(k-1)$ , polynomial-size circuits. This contradicts to our inductive hypothesis. This proves the theorem.  $\square$  Theorem 8.1

## 8.2 An Exponential-Size Lower Bound for Parity

### 8.2.1 Proving the Size Bound

In the previous section we proved that polynomial-size, constant-depth circuits cannot compute the parity function. In this section, we improve upon the proof technique to show an exponential-size lower bound for computing the parity function by constant-depth circuits. Based on this bound, we construct an oracle separating PSPACE from PH. Below is the first of our goals in this section, the exponential lower bound for parity.

**Theorem 8.3** *Let  $k \geq 2$ . Suppose that a family  $\{C_n\}_{n \geq 1}$  of depth- $k$  circuits computes the parity function. Then, for all but finitely many  $n$ ,*

$$\text{size}(C_n) > 2^{(1/10)^{k/(k-1)} n^{1/(k-1)}}.$$

The key ingredient of the proof of Theorem 8.3 is the following lemma, called *the switching lemma*, which generalizes the method we developed in the previous section.

**Lemma 8.4 (Switching Lemma)** *Let  $G$  be an  $\wedge$ - $\vee$  circuit with bottom fan-in at most  $t$ . Let  $p$ ,  $0 < p < 1$ , be such that  $5pt < 1$  and let  $\alpha$  be the unique positive root of the equation*

$$\left(1 + \frac{4p}{(1+p)\alpha}\right)^t = \left(1 + \frac{2p}{(1+p)\alpha}\right)^t + 1.$$

*Suppose that a restriction  $\rho$  is chosen under the distribution  $R_p^\pm$ . Then, for every  $s \geq 0$ , with probability at least  $1 - \alpha^s$ ,  $G[\rho]$  is equivalent to an  $\vee$ - $\wedge$  circuit of bottom fan-in strictly less than  $s$ .*

The lemma follows from the slightly stronger lemma below.

**Lemma 8.5** *Let  $G$  be an  $\wedge$ - $\vee$  circuit with bottom fan-in at most  $t$ . Let  $p$ ,  $0 < p < 1$ , be such that  $5pt < 1$  and let  $\alpha$  denote the unique positive root of the equation*

$$\left(1 + \frac{4p}{(1+p)\alpha}\right)^t = \left(1 + \frac{2p}{(1+p)\alpha}\right)^t + 1.$$

*Let  $F$  be any boolean function and let  $s \geq 0$  be arbitrary. Suppose that a restriction  $\rho$  is chosen under the distribution  $R_p^\Xi$ . Then*

$$\Pr[\text{MIN}(G[\rho]) \geq s \mid F[\rho \equiv 1] \leq \alpha^s.$$

**Proof of Lemma 8.5** Let  $G$ ,  $t$ ,  $p$ ,  $\alpha$ ,  $F$ , and  $s$  be as in the hypothesis. Note that the statement of the lemma trivially holds for  $s = 0$ . So, suppose  $s > 0$ . Let  $G_1, \dots, G_m$  be an enumeration of all the depth-1 subcircuits of  $G$ . Then  $G = \bigwedge_{i=1}^m G_i$ . We prove the statement by induction on  $m$ . The base case is when  $m = 0$ . If  $m$  is 0,  $G$  is a constant function. Which of the two constant functions  $G$  actually is may depend on the context. For every  $\rho$ ,  $G = G[\rho]$ , and thus,  $\text{MIN}(G[\rho]) = \text{MIN}(G) = 0$ . Thus, the probability that  $\text{MIN}(G[\rho]) \geq s$  is 0 regardless of the choice of  $F$ . Hence the claim holds for  $m = 0$ .

For the induction step, let  $m \geq 1$  and suppose that the claim holds for every  $m'$ ,  $0 \leq m' < m$ . We consider the following two cases:

**Case 1**  $G_1[\rho \equiv 1]$ , and

**Case 2**  $G_1[\rho \not\equiv 1]$ .

Then we have only to show that, for each  $i \in \{1, 2\}$ ,

$$\Pr[\text{MIN}(G[\rho]) \geq s \mid \text{Case } i \text{ holds and } F[\rho \equiv 1] \leq \alpha^s. \quad (8.2)$$

As to Case 1, let  $G' = \bigwedge_{i=2}^m G_i[\rho]$  and  $F' = F \wedge G_1$ . Note that if  $G_1[\rho \equiv 1]$  then  $G[\rho \equiv G']$ . Since the conditions  $G_1[\rho \equiv 1]$  and  $F[\rho \equiv 1]$  can be combined into  $F'[\rho \equiv 1]$ , the probability in equation 8.2 can be rewritten as

$$\Pr[\text{MIN}(G'[\rho]) \geq s \mid F'[\rho \equiv 1].$$

Then by our inductive hypothesis, this probability is at most  $\alpha^s$ . Thus, equation 8.2 holds for Case 1.

As to Case 2, define  $P_0 = \Pr[\text{MIN}(G[\rho]) \geq s \mid F[\rho \equiv 1 \wedge G_1[\rho \not\equiv 1]]$ . Then we have to show that  $P_0 \leq \alpha^s$  for all  $s > 0$ .

Let  $H = \bigwedge_{i=2}^m G_i$ . Let  $T$  be the set of all variables on which  $G_1$  depends. Since  $G$  is an  $\wedge$ - $\vee$  circuit, every minterm of  $G$  has to have a literal from  $T$ . Also, since we are considering only  $\rho$  such that  $G_1[\rho \not\equiv 1]$ , each minterm of  $G[\rho]$ , if one exists, has to have a literal from  $T$ . So, split  $\rho$  into the part  $\rho_1$  that assigns values to variables in  $T$  and the part  $\rho_2$  that assigns values to variables in  $\Xi - T$ . Let  $\sigma$  be a restriction. As we did for  $\rho$ , split  $\sigma$  into  $\sigma_1$  and  $\sigma_2$ . If  $\sigma$  is a minterm of  $G[\rho]$ , then the following three conditions must hold:

- (i)  $\sigma_1^{-1}(\{0, 1\}) \neq \emptyset$ .

- (ii)  $\rho_1^{-1}(\{0, 1\}) \cap \sigma_1^{-1}(\{0, 1\}) = \emptyset$ .
- (iii)  $\sigma_2$  is a minterm of  $H[\rho\sigma_1]$ .

So, if  $G[\rho]$  has a minterm of size at least  $s$ ,  $F[\rho] \equiv 1$ , and  $G[\rho] \neq 1$ , then there is some restriction  $\sigma_1$  over  $T$  that is disjoint with  $\rho_1$  such that  $H[\rho\sigma_1]$  has a minterm of size at least  $s - |\sigma_1|$ . For each nonempty restriction  $\sigma_1$  over  $T$ , define

$$Q[\sigma_1] = \Pr[\sigma_1^{-1}(\{0, 1\}) \cap \rho_1^{-1}(\{0, 1\}) = \emptyset \wedge \\ \text{MIN}(H[\rho\sigma_1]) \geq s - |\sigma_1| \mid F[\rho] \equiv 1 \wedge G_1[\rho] \neq 1].$$

Then  $P_0$  is bounded from above by the sum of  $Q[\sigma_1]$  where  $\sigma_1$  ranges over all nonempty restrictions over  $T$ .

Consider

$$Q_1[\sigma_1] = \Pr[\sigma_1^{-1}(\{0, 1\}) \cap \rho_1^{-1}(\{0, 1\}) = \emptyset \mid F[\rho] \equiv 1 \wedge G_1[\rho] \neq 1]$$

and

$$Q_2[\sigma_1] = \Pr[\text{MIN}(H[\rho\sigma_1]) \geq s - |\sigma_1| \mid F[\rho] \equiv 1 \wedge G_1[\rho] \neq 1].$$

Then  $Q[\sigma_1] \leq Q_1[\sigma_1]Q_2[\sigma_1]$ .

**Fact 8.6** For every  $\sigma_1$ ,  $\Pr[\sigma_1^{-1}(\{0, 1\}) \cap \rho_1^{-1}(\{0, 1\}) = \emptyset \mid G_1[\rho_1] \neq 1] \leq (\frac{2p}{1+p})^{|\sigma_1|}$ .

**Proof of Fact 8.6** In order for  $G_1[\rho_1] \neq 1$  to be true,  $\rho_1$  has to assign either 0 or \* to each literal of  $G_1$ . In order for  $\sigma_1^{-1}(\{0, 1\}) \cap \rho_1^{-1}(\{0, 1\}) = \emptyset$  to hold,  $\rho_1$  has to assign \* to all variables in  $\sigma_1^{-1}(\{0, 1\})$ . So, the probability in question is at most  $p^{|\sigma_1|} / (\frac{1-p}{2} + p)^{|\sigma_1|} = (\frac{2p}{1+p})^{|\sigma_1|}$ .  $\square$  Fact 8.6

**Fact 8.7** For any events  $A, B$ , and  $C$ ,  $\Pr[A \mid B \wedge C] \leq \Pr[A \mid C]$  if and only if  $\Pr[B \mid A \wedge C] \leq \Pr[B \mid C]$ .

**Proof of Fact 8.7** The proof is by routine calculation.  $\square$  Fact 8.7

**Fact 8.8**  $Q_1[\sigma_1] \leq (\frac{2p}{1+p})^{|\sigma_1|}$ .

**Proof of Fact 8.8** Let  $A$ ,  $B$ , and  $C$  be the events  $\sigma_1^{-1}(\{0, 1\}) \cap \rho_1^{-1}(\{0, 1\}) = \emptyset$ ,  $F[\rho] \equiv 1$ , and  $G_1[\rho_1] \neq 0, 1$ , respectively. Then  $Q_1[\sigma_1] = \Pr[A \mid B \wedge C]$ , and Fact 8.6 shows that  $\Pr[A \mid C] \leq (\frac{2p}{1+p})^{|\sigma_1|}$ . Note that

$$\Pr[F[\rho] \equiv 1 \mid \sigma_1^{-1}(\{0, 1\}) \cap \rho_1^{-1}(\{0, 1\}) = \emptyset \wedge G_1[\rho_1] \neq 1] \\ \leq \Pr[F[\rho] \equiv 1 \mid G_1[\rho_1] \neq 1],$$

because adding the condition  $\sigma_1^{-1}(\{0, 1\}) \cap \rho_1^{-1}(\{0, 1\}) = \emptyset$  does not increase the probability that  $F[\rho] \equiv 1$ . Thus,  $\Pr[B \mid A \wedge C] \leq \Pr[B \mid C]$ . Now, by Fact 8.7,  $Q_1[\sigma_1] \leq \Pr[A \mid C]$ , and this implies that  $Q_1[\sigma_1] \leq (\frac{2p}{1+p})^{|\sigma_1|}$ .  $\square$  Fact 8.8

**Fact 8.9**  $Q_2[\sigma_1] \leq \alpha^{s-|\sigma_1|}$ .

**Proof of Fact 8.9** Let  $Z$  denote the set of all restrictions over variables  $T$ . Note that

$$Q_2[\sigma_1] \leq \max\{\Pr[\text{MIN}((H[\sigma_1\rho_1])[\rho_2] \geq s - |\sigma_1|) \mid (F[\rho_1\sigma_1])[\rho_2] \equiv 1 \wedge G_1[\rho_1] \neq 0 \wedge G_1[\rho_1] \neq 1] \mid \rho_1 \in Z]\}$$

where  $\rho_2$  is subject to the distribution  $R_p^{\Xi-T}$ . This inequality holds because restricting  $F[\rho]$  to  $\sigma_1$  does not decrease the probability. We can eliminate the condition  $G_1[\rho_1] \neq 0 \wedge G_1[\rho_1] \neq 1$  from this because we are maximizing over all possible  $\rho_1$ . So,

$$Q_2[\sigma_1] \leq \max\{\Pr[\text{MIN}((H[\sigma_1\rho_1])[\rho_2] \geq s - |\sigma_1|) \mid (F[\sigma_1\rho_1])[\rho_2] \equiv 1] \mid \rho_1 \in Z],$$

where  $\rho_2$  is subject to the distribution  $R_p^{\Xi-T}$ . Then, since  $H[\sigma_1\rho_1]$  is an  $\wedge$  of  $m-1$   $\vee$  circuits, each of which has fan-in at most  $t$ , by our inductive hypothesis (recall that we are in the proof of Lemma 8.5),  $Q_2[\sigma_1] \leq \alpha^{s-|\sigma_1|}$ . □ Fact 8.9

By Facts 8.8 and 8.9, we have

$$Q[\sigma_1] \leq \left(\frac{2p}{1+p}\right)^{|\sigma_1|} \alpha^{s-|\sigma_1|}.$$

This implies

$$P_0 \leq \sum_{1 \leq i \leq \|T\|} \sum_{|\sigma_1|=i} \left(\frac{2p}{1+p}\right)^{|\sigma_1|} \alpha^{s-|\sigma_1|}.$$

For each  $i$ ,  $1 \leq i \leq \|T\|$ , and for each nonempty subset of  $T$  of size  $i$ , there are  $2^i - 1$  possibilities for  $\sigma_1$  since  $\sigma_1$  has to assign 1 to at least one literal of  $G_1$ . Then

$$P_0 \leq \sum_{1 \leq i \leq \|T\|} \binom{\|T\|}{i} (2^i - 1) \left(\frac{2p}{1+p}\right)^i \alpha^{s-i}.$$

By hypothesis,  $\|T\| \leq t$ . So

$$P_0 \leq \alpha^s \sum_{0 \leq i \leq t} \binom{t}{i} (2^i - 1) \left(\frac{2p}{1+p}\right)^i \alpha^{-i}.$$

Note that

$$\begin{aligned} & \sum_{0 \leq i \leq t} \binom{t}{i} (2^i - 1) \left(\frac{2p}{1+p}\right)^i \alpha^{-i} \\ &= \sum_{0 \leq i \leq t} \binom{t}{i} \left(\frac{4p}{(1+p)\alpha}\right)^i - \sum_{0 \leq i \leq t} \binom{t}{i} \left(\frac{2p}{(1+p)\alpha}\right)^i \\ &= \left(1 + \frac{4p}{(1+p)\alpha}\right)^t - \left(1 + \frac{2p}{(1+p)\alpha}\right)^t. \end{aligned}$$



This last formula is 1 by the assumption of the lemma. So,  $P_0 \leq \alpha^s$ . Thus, the claim holds. This proves the lemma.  $\square$  Lemma 8.5

Lemma 8.4 immediately follows from Lemma 8.5 by taking  $F = 1$ .

Next we derive another lemma from Lemma 8.4. For each  $k \geq 2$  and  $n \geq 1$ , define  $\ell(n, k) = \frac{1}{10} n^{\frac{1}{k-1}}$ .

**Lemma 8.10** *Let  $k \geq 2$ . Let  $\{C_n\}_{n \geq 1}$  be a family of depth- $k$  circuits. Suppose that for every  $n \geq 1$  the following conditions hold:*

1. *Every depth-1 subcircuit of  $C_n$  is of fan-in at most  $\ell(n, k)$ .*
2. *There are at most  $2^{\ell(n, k)}$  depth-2 subcircuits of  $C_n$ .*

*Then for only finitely many  $n$  does  $C_n$  compute  $\pi_n$ .*

**Proof** The proof is by induction on  $k$ . For the base case, let  $k = 2$ . Then for all  $n \geq 1$   $\ell(n, k) < n$ . Let  $n \geq 1$  and let  $C_n$  be a depth-2 circuit satisfying the two conditions in the statement of the lemma. By property 1, each depth-1 subcircuit of  $C_n$  is of fan-in less than  $n$ . So, in the case where  $C_n$  is an  $\vee$ - $\wedge$  circuit, there is a restriction  $\rho$  of size less than  $n$  that reduces one of the subcircuits to 1, thereby reducing  $C_n$  to 1, and in the case where  $C_n$  is an  $\wedge$ - $\vee$  circuit, there is a restriction of size less than  $n$  that reduces one of the subcircuits to 0, thereby reducing  $C_n$  to 0. Such a restriction does not reduce  $\pi_n$  to a constant function, so  $C_n$  does not compute  $\pi_n$ .

For the induction step, let  $k \geq 3$  and suppose that the claim holds for all  $k'$ ,  $2 \leq k' < k$ . By symmetry between  $\wedge$  and  $\vee$ , we may assume that the depth-2 subcircuits of  $C_n$  are  $\wedge$ - $\vee$  circuits. Let  $n > 10^{k-1}$  and suppose  $C_n$  satisfies properties 1 and 2. Let  $p = \frac{1}{10\ell(n, k)} = n^{-\frac{1}{k-1}}$  and let  $s = t = \ell(n, k)$ . Because  $0 < 5pt = \frac{1}{2} < 1$  and for every  $n > 10^{k-1}$ ,  $0 < p < 1$ , we can apply Lemma 8.4 to each depth-2 subcircuit of  $C_n$ . Then, for each depth-2 subcircuit  $H$  of  $C_n$ , the probability that  $H$  cannot be rewritten as an  $\vee$ - $\wedge$  circuit of bottom fan-in at most  $s$  is at most  $\alpha^{\ell(n, k)}$ , where  $\alpha$  is the unique positive root of the equation

$$\left(1 + \frac{4p}{(1+p)\alpha}\right)^t = \left(1 + \frac{2p}{(1+p)\alpha}\right)^t + 1.$$

By plugging  $p = \frac{1}{10\ell(n, k)}$  and  $t = \ell(n, k)$  into this equation, we obtain

$$\left(1 + \frac{4}{(1+10\ell(n, k))\alpha}\right)^{\ell(n, k)} = \left(1 + \frac{2}{(1+10\ell(n, k))\alpha}\right)^{\ell(n, k)} + 1.$$

Since  $\ell$  is an increasing function of  $n$ , the left-hand side approaches

$$e^{\frac{4\ell(n, k)}{1+10\ell(n, k)\alpha}}$$

as  $n$  increases, which has the limit value of  $e^{\frac{2}{5\alpha}}$ . By a similar analysis, the right-hand side approaches  $1 + e^{\frac{1}{5\alpha}}$ . By replacing  $e^{\frac{1}{5\alpha}}$  by a variable  $Z$ , we get

equation  $Z^2 - Z - 1 = 0$ , which has a unique positive solution  $Z = \frac{1+\sqrt{5}}{2}$ . So, the unique solution of the original equation approaches

$$\frac{1}{5 \left( \ln \frac{1+\sqrt{5}}{2} \right)} = 0.4156 \dots$$

Thus, for sufficiently large  $n$ ,  $\alpha < 0.45 = \frac{9}{20}$ . Since there are at most  $2^{\ell(n,k)}$  depth-2 subcircuits of  $C_n$ , the probability that every depth-2 subcircuit can be rewritten as an  $\vee\wedge$  circuit of bottom fan-in at most  $s$  is at least

$$1 - 2^{\ell(n,k)} \alpha^{\ell(n,k)} = 1 - (2\alpha)^{\ell(n,k)} > 1 - \left( \frac{9}{10} \right)^{\ell(n,k)} > \frac{2}{3}$$

for sufficiently large  $n$ . If we replace each depth-2 circuit of  $C_n[\rho]$  by an equivalent  $\vee\wedge$  circuit, then we can collapse the second and the third levels of  $C_n$  into one because they have the same gate type, thereby obtaining an equivalent depth- $(k-1)$  circuit. Thus, with probability greater than  $\frac{2}{3}$ ,  $C_n[\rho]$  can be rewritten as a circuit of depth  $k-1$  and of size at most  $\text{size}(C_n)$ .

On the other hand, the expected number of variables in  $C_n[\rho]$  is  $pn = n^{\frac{k-2}{k-1}}$ . So, for sufficiently large  $n$ , the probability that the number of variables in  $C_n[\rho]$  is at least  $pn$  is larger than  $\frac{1}{3}$ .

The probability that both of the above events occur at the same time is larger than  $\frac{2}{9}$ . Note that  $m \geq n^{\frac{k-2}{k-1}}$  implies  $\ell(m, k-1) \geq \ell(n, k)$ . Thus, with some positive probability, we can convert  $C_n[\rho]$  to a depth- $(k-1)$  circuit  $D_m$  over  $m \geq pn$  variables such that

1. every depth-1 subcircuit of  $D_m$  is of fan-in at most  $\ell(m, k-1)$  and
2. there are at most  $2^{\ell(m, k-1)}$  depth-2 subcircuits of  $D_m$ .

By our inductive hypothesis,  $D_m$  does not compute  $\pi_m$ , and thus,  $C_n$  does not compute  $\pi_n$ . Thus the claim holds for  $k$ . This proves the lemma.  $\square$

Now we are ready to complete the proof of Theorem 8.3.

**Proof of Theorem 8.3** Let  $n \geq 1$  and let  $C_n$  be a depth- $k$  circuit of size bounded by  $2^{(1/10)^k/(k-1)} n^{1/(k-1)}$ . We can view  $C_n$  as a depth- $(k+1)$  circuit all of whose depth-1 subcircuits are of fan-in 1. Let  $p = \frac{1}{10}$ ,  $s = \frac{1}{10} \left( \frac{n}{10} \right)^{\frac{1}{k-1}}$ , and  $t = 1$ , and apply Lemma 8.4. Since  $\alpha = \frac{2p}{1+p} = \frac{2}{11}$ , the probability that all depth-2 subcircuits of  $C_n[\rho]$  can be rewritten so that the second and the third levels of  $C_n[\rho]$  have the same type and thus can be collapsed into one is at least  $1 - 2^s \alpha^s \geq 1 - 2^s 4^{-s} = 1 - 2^{-s}$ .

On the other hand, the expected number of variables in  $C_n[\rho]$  is  $\frac{n}{10}$ . So, with probability larger than  $\frac{1}{3}$ ,  $C_n[\rho]$  is dependent on at most  $m = \frac{n}{10}$  variables.

So, with probability  $(1 - 2^{-s}) + \frac{1}{3} - 1 > 0$ , the above two events occur at the same time. Thus, for some restriction  $\rho$ ,  $C_n[\rho]$  can be converted to a circuit  $D_m$  such that

1. every depth-1 subcircuit of  $D_m$  is of fan-in at most  $\ell(m, k)$  and

2.  $D_m$  has at most  $2^{\ell(m,k)}$  depth-2 subcircuits.

Then, by Lemma 8.10,  $D_m$  does not compute  $\pi_m$ . Thus,  $C_n$  does not compute  $\pi_n$ . This proves the theorem.  $\square$  Theorem 8.3

### 8.2.2 Constructing an Oracle Separating PSPACE from PH

We will now use Theorem 8.3 to show that there is an oracle relative to which PH is properly included in PSPACE. Our model of PSPACE oracle computation even requires that the query tape is polynomially length bounded. (In the model lacking this requirement, it is trivial to separate PSPACE from PH via an oracle.)

**Theorem 8.11** *There is an oracle  $A$  such that  $\text{PH}^A \neq \text{PSPACE}^A$ .*

**Proof** Since for every oracle  $A$ ,  $\oplus P^A \subseteq \text{PSPACE}^A$ , it suffices to construct an oracle  $A$  such that  $\oplus P^A \not\subseteq \text{PH}^A$ . For each language  $A$ , define

$$W(A) = \{0^n \mid |\{0,1\}^n \cap A| \text{ is odd}\}.$$

Let  $M$  be a polynomial time-bounded Turing machine that, on input  $x \in \Sigma^*$ , guesses  $y$  of length  $|x|$ , and accepts if and only if  $y$  is in the oracle. Then for every oracle  $A$ , and every  $x \in \Sigma^*$ ,  $0^{|x|} \in W(A)$  if and only if  $M^A$  on  $x$  has an odd number of accepting computation paths. Thus,  $W(A) \in \oplus P^A$  for every oracle  $A$ .

In order to construct an oracle  $A$  relative to which  $W(A) \notin \text{PH}^A$ , we first introduce a view of PH in terms of constant-depth circuits and define an enumeration of all relativized PH-machines based on that view. Then we consider a very simple oracle construction scheme in which the machines in the enumeration are “killed” one after another. Finally we argue that the scheme will be successful, because construction failing at a stage would imply that we could construct a constant-depth subexponential-size circuit family for parity, which contradicts Theorem 8.3.

Recall that for a language  $A$ ,  $\overline{A} \oplus A$  denotes  $\{0x \mid x \notin A\} \cup \{1x \mid x \in A\}$ . First we draw an analogy between the polynomial hierarchy and constant-depth circuits.

**Proposition 8.12** *Let  $k \geq 1$  be an integer and let  $A$  be a language. Let  $\mathcal{C}$  be one of  $\Sigma_k^{p,A}$  or  $\Pi_k^{p,A}$ . Then, for every  $L \in \mathcal{C}$ , there exist a polynomial  $p$  and a polynomial-time computable function  $f : \Sigma^* \rightarrow \Sigma^*$  such that, for every  $x \in \Sigma^*$ ,*

$$\begin{aligned} x \in L &\iff (Q_1 y_1 : y_1 \in \Sigma^{p(|x|)}) \dots (Q_k y_k : y_k \in \Sigma^{p(|x|)}) \\ &\quad (Q_{k+1} j : 1 \leq j \leq p(|x|)) [f(\langle x, y_1, \dots, y_k, j \rangle) \in \overline{A} \oplus A], \end{aligned} \quad (8.3)$$

where the quantifiers alternate, and  $Q_1 = \exists$  if  $\mathcal{C} = \Sigma_k^{p,A}$  and  $\forall$  otherwise.

**Proof of Proposition 8.12** Let  $k \geq 1$  and let  $\mathcal{C}$  be either  $\Sigma_k^{p,A}$  or  $\Pi_k^{p,A}$ . Let  $L \in \mathcal{C}$ . Then there exist a polynomial  $p_0$  and a polynomial time-bounded deterministic oracle Turing machine  $M$  such that, for every  $x \in \Sigma^*$ ,

$$x \in L \iff (Q_1 y_1 : y_1 \in \Sigma^{p_0(|x|)}) (Q_2 y_2 : y_2 \in \Sigma^{p_0(|x|)}) \dots (Q_k y_k : y_k \in \Sigma^{p_0(|x|)}) [M^A(\langle x, y_1, \dots, y_k \rangle) \text{ accepts}], \quad (8.4)$$

where the quantifiers alternate, and  $Q_1 = \exists$  if  $\mathcal{C} = \Sigma_k^{p,A}$  and  $Q_1 = \forall$  if  $\mathcal{C} = \Pi_k^{p,A}$ . We divide the proof into four cases:

**Case 1**  $\mathcal{C} = \Sigma_k^{p,A}$  and  $k$  is odd,

**Case 2**  $\mathcal{C} = \Sigma_k^{p,A}$  and  $k$  is even,

**Case 3**  $\mathcal{C} = \Pi_k^{p,A}$  and  $k$  is odd, and

**Case 4**  $\mathcal{C} = \Pi_k^{p,A}$  and  $k$  is even.

First we consider Case 1. Here  $Q_k = \exists$ . Let  $p_1$  be a polynomial bounding the runtime of  $M$ . We can assume that  $p_1$  is increasing; i.e., for all  $n \geq 0$ ,  $p_1(n+1) > p_1(n)$ . We may assume that, for all  $u \in \Sigma^*$ ,  $M$  on input  $u$  makes exactly  $p_1(|u|)$  queries regardless of its oracle. We will replace  $M$  by a new machine,  $M'$ , that on each input  $u$  simulates  $M$  on input  $u$  while counting in a variable  $C$  the number of queries that  $M$  makes along the simulated path. When  $M$  halts, if  $C$  is smaller than  $p_1(|u|)$ , then  $M'$  queries the empty string to the oracle exactly  $p_1(|u|) - C$  times. Then  $M'$  accepts if  $M$  on  $x$  accepts along the simulated path and rejects otherwise.

Let  $N$  be a deterministic Turing machine that, on input  $w = \langle u, v \rangle$ , if  $|v| = p_1(|u|)$ , then simulates  $M'$  on input  $u$  by assuming, for each  $i$ ,  $1 \leq i \leq p_1(|u|)$ , that the answer of the oracle to the  $i$ th query is affirmative if the  $i$ th bit of  $v$  is 1 and the answer is negative if the  $i$ th bit of  $v$  is 0. If  $|v| \neq p_1(|u|)$ , then  $N$  rejects  $w$  immediately. For each  $u \in \Sigma^*$  and each  $v \in \Sigma^{p_1(|u|)}$ , and  $j$ ,  $1 \leq j \leq p_1(|u|)$ , let  $R(u, v, j)$  denote the  $j$ th query of  $M'$  on input  $u$  along the simulation carried out by  $N$  on input  $\langle u, v \rangle$ . Then, for all  $u$ ,  $u \in L$  if and only if for some  $v$ ,  $|v| = p_1(|u|)$ , it holds that  $N$  on input  $\langle u, v \rangle$  accepts and that, for all  $j$ ,  $1 \leq j \leq p_1(|u|)$ ,  $R(u, v, j) \in A$  if the  $j$ th bit of  $v$  is 1 and  $R(u, v, j) \in \bar{A}$  otherwise.

Let  $s_0 \notin \bar{A} \oplus A$  and  $s_1 \in \bar{A} \oplus A$  be fixed. We define a mapping  $f_0$ . For every  $w \in \Sigma^*$ ,  $f_0(w)$  is defined as follows:

- If for some  $u, v \in \Sigma^*$  and  $t$ ,  $1 \leq t \leq |v|$ , it holds that  $|v| = p_1(|u|)$ ,  $w = \langle u, v, t \rangle$ , and  $N$  on input  $\langle u, v \rangle$  accepts, then  $f_0(w) = v_t R(u, v, t)$ , where  $v_t$  is the  $t$ th bit of  $v$ .
- If for some  $u, v \in \Sigma^*$  and  $t$ ,  $1 \leq t \leq |v|$ , it holds that  $|v| = p_1(|u|)$ ,  $w = \langle u, v, t \rangle$ , and  $N$  on input  $\langle u, v \rangle$  rejects, then  $f_0(w) = s_0$ .
- If neither of the above two conditions hold, then  $f_0(w) = s_1$ .

Then  $f_0$  is polynomial-time computable, and for every  $u \in \Sigma^*$ ,  $M^A(u)$  accepts if and only if

$$(\exists v : v \in \Sigma^{p_1(|u|)}) (\forall t : 1 \leq t \leq p_1(|u|)) [f_0(\langle u, v, t \rangle) \in \bar{A} \oplus A].$$

By combining this with equation 8.4,

$$x \in L \iff (\exists y_1 : y_1 \in \Sigma^{p_0(|x|)}) \dots (\exists y_k : y_k \in \Sigma^{p_0(|x|)}) \\ (\exists v : v \in \Sigma^{p_1(|u|)}) (\forall t : 1 \leq t \leq p_1(|u|)) [f_0(\langle u, v, t \rangle) \in \overline{A} \oplus A],$$

where  $u = \langle x, y_1, \dots, y_k \rangle$ . Let  $r$  be a polynomial such that, for all  $x \in \Sigma^*$  and  $y_1, \dots, y_k \in \Sigma^{p_0(|x|)}$ ,  $|\langle x, y_1, \dots, y_k \rangle| \leq r(|x|)$ . Define  $p(n) = p_0(n) + p_1(r(n))$ . We define a new mapping  $f$ . For every  $k+2$  tuple  $w = \langle x, w_1, \dots, w_k, t \rangle$ ,  $f(w)$  is defined as follows:

- If  $|w_k| \geq p(|x|)$  and there exist some  $y_1, \dots, y_k, v \in \Sigma^*$  such that
  - for every  $i$ ,  $1 \leq i \leq k$ ,  $y_i$  is the prefix of  $w_i$  having length  $p_0(|x|)$ ,
  - $v$  is the suffix of  $w_k$  having length  $p_1(|\langle x, y_1, \dots, y_k \rangle|)$ , and
  - $1 \leq t \leq p_1(|\langle x, y_1, \dots, y_k \rangle|)$ ,
 then  $f(w) = f_0(\langle x, y_1, \dots, y_k, v, t \rangle)$ .
- Otherwise,  $f(w) = s_1$ .

Then  $f$  is polynomial-time computable and for every  $x \in \Sigma^*$ ,

$$x \in L \iff (\exists y_1 : y_1 \in \Sigma^{p(|x|)}) \dots (\exists y_k : y_k \in \Sigma^{p(|x|)}) \\ (\forall t : 1 \leq t \leq p(|x|)) [f(\langle x, y_1, \dots, y_k, t \rangle) \in \overline{A} \oplus A],$$

as desired.

We can treat Case 4 similarly. The only difference here is that the first quantifier  $Q_1$  is  $\forall$ .

Next we consider Case 3. Let  $B = \overline{A}$ . Since  $\Sigma_k^{p,A} = \Sigma_k^{p,B}$  and  $\overline{L}$  belongs to  $\Sigma_k^{p,A}$ , we obtain the following characterization of  $\overline{L}$  as in Case 1: For every  $x \in \Sigma^*$ ,

$$x \in \overline{L} \iff (\exists y_1 : y_1 \in \Sigma^{p(|x|)}) \dots (\exists y_k : y_k \in \Sigma^{p(|x|)}) \\ (\forall t : 1 \leq t \leq p(|x|)) [f(\langle x, y_1, \dots, y_k, t \rangle) \in \overline{B} \oplus B],$$

where the quantifiers alternate. By negating both sides of the equality, for every  $x \in \Sigma^*$ ,

$$x \in L \iff (\forall y_1 : y_1 \in \Sigma^{p(|x|)}) \dots (\forall y_k : y_k \in \Sigma^{p(|x|)}) \\ (\exists t : 1 \leq t \leq p(|x|)) [f(\langle x, y_1, \dots, y_k, t \rangle) \notin \overline{B} \oplus B].$$

Since  $\overline{\overline{A} \oplus A} = A \oplus \overline{A}$ , the condition in the bracket can be written as  $f(\langle x, y_1, \dots, y_k, t \rangle) \in B \oplus \overline{B}$ . Since  $B = \overline{A}$ , this condition is  $f(\langle x, y_1, \dots, y_k, t \rangle) \in \overline{A} \oplus A$ . So, for every  $x \in \Sigma^*$ ,

$$x \in L \iff (\forall y_1 : y_1 \in \Sigma^{p(|x|)}) \dots (\forall y_k : y_k \in \Sigma^{p(|x|)}) \\ (\exists t : 1 \leq t \leq p(|x|)) [f(\langle x, y_1, \dots, y_k, t \rangle) \in \overline{A} \oplus A],$$

as desired. Case 2 is the same as Case 3 except that the first quantifier is  $\exists$ . This proves the proposition.  $\square$  Proposition 8.12

Next we present our oracle construction scheme. Let  $\{p_i\}_{i \geq 1}$  be an enumeration of polynomials such that, for each  $i \geq 1$ ,  $p_i(n) = n^i + i$ . For all

polynomials  $p$ , there is some  $i \geq 1$  such that, for all  $n \geq 0$ ,  $p(n) \leq p_i(n)$ . Let  $f_1, f_2, \dots$  be an enumeration of all polynomial-time computable functions. For each triple  $s = \langle i, j, k \rangle$ , let  $K_s(A)$  be the language in  $\Sigma_k^{p_i, A}$  characterized as in equation 8.3 with  $p_i$  and  $f_j$  in place of  $p$  and  $f$ , respectively. More precisely, for every triple  $s = \langle i, j, k \rangle$ , for every  $x \in \Sigma^*$ , and for every oracle  $A$ ,  $x \in K_s(A)$  if and only if

$$(Q_1 y_1 : y_1 \in \Sigma^{p_i(|x|)}) \dots (Q_k y_k : y_k \in \Sigma^{p_i(|x|)}) \\ (Q_{k+1} t : 1 \leq t \leq p_i(|x|)) [f_j(\langle x, y_1, \dots, y_k, t \rangle) \in \bar{A} \oplus A].$$

The language  $A$  is constructed in stages. At stage  $s = \langle i, j, k \rangle$  we will identify an integer  $\ell_s$  and extend  $A$  as well as  $\bar{A}$  up to length  $\ell_s$ , so that there exists some integer  $n \geq 0$  such that  $0^n \in W(A) \iff 0^n \notin K_s(A)$ . We assume that for all  $i, j, k \geq 1$ ,  $\langle i, j, k \rangle \geq 1$  and that  $\langle 1, 1, 1 \rangle = 1$ . Let  $\ell_0 = 0$ . We put the empty string in  $\bar{A}$ .

Let  $s = \langle i, j, k \rangle$ . The construction in stage  $s$  proceeds as follows:

- Let  $A_0$  (respectively,  $A_1$ ) be the set of all strings put in  $\bar{A}$  (respectively,  $A$ ) prior to stage  $s$ . It holds that  $A_0 \cap A_1 = \emptyset$  and  $A_0 \cup A_1 = (\Sigma^*)^{\leq \ell_{s-1}}$ .
- Let  $r$  be the smallest polynomial in the enumeration such that, for all  $x \in \Sigma^*$  and  $y_1, \dots, y_k \in \Sigma^{p_i(|x|)}$ , it holds that  $|f_j(\langle x, y_1, \dots, y_k, j \rangle)| \leq r(|x|)$ .
- For each  $n = \ell_{s-1} + 1, \ell_{s-1} + 2, \dots$ , test whether there is a partition  $(B_0, B_1)$  of  $(\Sigma^*)^{\leq r(n)}$  such that  $B_0 \supseteq A_0$ ,  $B_1 \supseteq A_1$ , and

$$0^n \in W(B_1) \iff 0^n \notin K_s(B_1).$$

If such a partition is found for  $n$ , then do the following:

- Set  $\ell_s$  to  $r(n)$ .
- Add all strings in  $B_0 - A_0$  to  $\bar{A}$ .
- Add all strings in  $B_1 - A_1$  to  $A$ .
- Terminate the loop and proceed to the next stage.

We claim that this construction is successful at every stage. We prove the claim by contradiction. Assume that at stage  $s = \langle i, j, k \rangle$ , for all  $n \geq \ell_{s-1} + 1$ , the search for a desired partition fails, i.e., for all  $n \geq \ell_{s-1} + 1$  and for all partitions  $(B_0, B_1)$  of  $\Sigma^{r(n)}$  such that  $B_0 \supseteq A_0$  and  $B_1 \supseteq A_1$ , it holds that

$$0^n \in W(B_1) \iff 0^n \in K_s(B_1).$$

We construct from  $K_s$  a family of depth- $(k+1)$  circuits  $C_1, C_2, \dots$  for parity in the following way.

For each  $n \geq 1$ , let  $\mu(n) = \min\{l \in \mathbb{N} \mid l \geq \ell_{s-1} + 1 \wedge 2^l \geq n\}$ . For each  $n \geq 1$ , the circuit  $C_n$  is constructed from  $\phi_0$  below, which is the formula for  $K_s$  on input  $0^{\mu(n)}$ :

$$\phi_0 = (Q_1 y_1 : y_1 \in \Sigma^{p_i(\mu(n))}) \dots (Q_k y_k : y_k \in \Sigma^{p_i(\mu(n))}) \\ (Q_{k+1} t : 1 \leq t \leq p_i(\mu(n))) [f_j(\langle 0^{\mu(n)}, y_1, \dots, y_k, t \rangle) \in \bar{A} \oplus A].$$

The quantifiers appearing in  $\phi_0$  alternate; that is, in the case when  $Q_1 = \exists$ , for all  $r$ ,  $1 \leq t \leq k+1$ ,  $Q_r = \exists$  if  $r$  is odd  $Q_r = \forall$  if  $r$  is even, and in the case when  $Q_1 = \forall$ , for all  $r$ ,  $1 \leq t \leq k+1$ ,  $Q_r = \forall$  if  $r$  is odd  $Q_r = \exists$  if  $r$  is even. For each  $r$ ,  $1 \leq r \leq k$ , and each  $y_1, \dots, y_r \in \Sigma^{p_i(\mu(n))}$ , let  $\phi_r[y_1, \dots, y_r]$  denote the formula

$$(Q_{r+1}y_{r+1} : y_{r+1} \in \Sigma^{p_i(\mu(n))}) \dots (Q_k y_k : y_k \in \Sigma^{p_i(\mu(n))}) \\ (Q_{k+1}t : 1 \leq t \leq p_i(\mu(n))) [f_j(\langle 0^{\mu(n)}, y_1, \dots, y_k, t \rangle) \in \bar{A} \oplus A].$$

For each  $t$ ,  $1 \leq t \leq p_i(\mu(n))$ , and each  $y_1, \dots, y_r \in \Sigma^{p_i(\mu(n))}$ , let  $\phi_{k+1}[y_1, \dots, y_r, t]$  denote the formula

$$f_j(\langle 0^{\mu(n)}, y_1, \dots, y_k, t \rangle) \in \bar{A} \oplus A.$$

To construct  $C_n$ , for each of the  $\phi$ 's defined in the above, introduce a gate corresponding to it. The type of the gates is determined as follows:

- The node corresponding to  $\phi_0$  is the output gate. The output gate is an  $\wedge$  gate if  $Q_1 = \forall$  and is an  $\vee$  gate if  $Q_1 = \exists$ .
- Each node corresponding to a  $\phi_{k+1}$  formula is an input gate.
- Let  $1 \leq r \leq k$ . Each node corresponding to a  $\phi_r$  formula is an  $\wedge$  gate if  $Q_{r+1} = \forall$  and is an  $\vee$  gate if  $Q_{r+1} = \exists$ .

The inputs to the nonleaf gates are determined as follows:

- The inputs of the output gate are the gates corresponding to  $\{\phi_1[y_1] \mid y_1 \in \Sigma^{p_i(\mu(n))}\}$ .
- Let  $1 \leq r \leq k-1$ . Let  $g$  be a gate corresponding to  $\phi_r[y_1, \dots, y_r]$  for some  $y_1, \dots, y_r \in \Sigma^{p_i(\mu(n))}$ . The inputs of  $g$  are  $\{\phi_{r+1}[y_1, \dots, y_{r+1}] \mid y_{r+1} \in \Sigma^{p_i(\mu(n))}\}$ .
- Let  $g$  be a gate corresponding to  $\phi_r[y_1, \dots, y_k]$  for some  $y_1, \dots, y_k \in \Sigma^{p_i(\mu(n))}$ . The inputs of  $g$  are  $\{\phi_{k+1}[y_1, \dots, y_k, t] \mid 1 \leq t \leq p_i(\mu(n))\}$ .

Since  $2^{\mu(n)} \geq n$ ,  $\Sigma^{\mu(n)}$  has cardinality at least  $n$ . Let  $W_n = \{w_1, \dots, w_n\}$  be the smallest  $n$  strings of length  $\mu(n)$ . Let  $y_1, \dots, y_k \in \Sigma^{p_i(\mu(n))}$  and  $1 \leq t \leq p_i(\mu(n))$ . Let  $g$  be the input gate corresponding to  $\phi_{k+1}[y_1, \dots, y_k, t]$ . Let  $z = f_j(0^{\mu(n)}, y_1, \dots, y_k, t)$ . The label of  $g$  is determined as follows:

- If for some  $l$ ,  $1 \leq l \leq n$ ,  $z = 1w_l$ , then  $g$  is labeled  $w_l$ .
- If for some  $l$ ,  $1 \leq l \leq n$ ,  $z = 0w_l$ , then  $g$  is labeled  $\bar{w}_l$ .
- If for some  $u \in \Sigma^* - W_n$  it holds that  $z = 1u$ , then  $g$  is assigned 1 if  $u \in A_1$  and is assigned 0 otherwise.
- If for some  $u \in \Sigma^* - W_n$  it holds that  $z = 0u$ , then  $g$  is assigned 0 if  $u \in A_1$  and assigned 1 otherwise.
- If  $z$  is the empty string, assign  $g$  to 0.

Then work from the input level to eliminate all subcircuits whose output is a constant regardless of the values of  $w_1, \dots, w_n$ . This is  $C_n$ . The circuit  $C_n$  clearly has depth  $k+1$ .

By assumption, for every  $B \subseteq \{w_1, \dots, w_n\}$ ,

$$0^n \in W(A_1 \cup B) \iff 0^n \in K_s(A_1 \cup B).$$

Since

$$0^n \in W(A_1 \cup B) \iff \|B\| \text{ is odd}$$

and

$$0^n \in K_s(A_1 \cup B) \iff C_n(\chi_B(w_1) \cdots \chi_B(w_n)) = 1,$$

$C_n$  computes  $\pi_n$ . For every  $n \geq 1$ , the size of  $C_n$  is at most

$$\sum_{0 \leq r \leq k} (2^{p_i(\mu(n))})^r + p_i(\mu(n))(2^{p_i(\mu(n))})^k < (p_i(\mu(n)) + 2)(2^{p_i(\mu(n))})^k.$$

By definition, there exists a fixed constant  $c > 0$  such that for every  $n \geq 1$  the number  $\mu(n)$  in the construction of  $C_n$  is at most  $c \log n$ . So, for every  $n \geq 1$ , the size of  $C_n$  is at most

$$(p_i(c \log n) + 2)2^{kp_i(c \log n)}.$$

For some constant  $c' > 0$ , this is at most

$$c'2^{\log^{i+1} \log n}.$$

Hence,  $\{C_n\}_{n \geq 1}$  is a family of depth- $(k+1)$ , size- $\mathcal{O}(2^{c' \log^{k+1} n})$  circuits computing the parity function. However, this is impossible due to Theorem 8.3 since  $c'2^{\log^{i+1} \log n} = o(2^{(1/10)^{(k+1)/k} n^{1/k}})$ . Thus, the construction is successful at every stage. This proves the theorem.  $\square$

### 8.3 PH and PSPACE Differ with Probability One

In the previous section, we proved an exponential-size lower bound for constant-depth circuits for computing the parity function and, based on that impossibility result, proved the existence of an oracle that separates PSPACE from PH. One might be tempted to ask how common it is for us to find an oracle that separates the two classes if we randomly search for one.

To formalize the question, consider the characteristic sequence for each set; for each  $i \geq 1$ , the  $i$ th bit of the sequence is a 1 if and only if the  $i$ th string of  $\Sigma^*$  belongs to the set. For each set  $A$ ,  $w(A)$  denotes the characteristic sequence of  $A$ . Note that, for each set  $A$ ,  $w(A) \in \{0, 1\}^\omega$ . The question we are asking is how dense is the collection,  $\mathbf{C}$ , of sequences corresponding to oracles that separate PSPACE from PH. The cardinality of all the subsets of  $\Sigma^*$  is  $\aleph_1$ , so ordinary counting methods do not apply to calculating the density. So we use Lebesgue measure. The characteristic sequence of a set is a real number in  $[0, 1]$ . First of all, we check whether  $\mathbf{C}$  is measurable in  $[0, 1]$ ,



and then, if so, measure its density. The mapping from the characteristic sequences to  $[0, 1]$  is one-to-one except for finite and cofinite sets. For a finite set  $S$ , the characteristic sequence of  $S$  and the characteristic sequence of  $\bar{S}$  are the same real number. However, this is not an issue here since the number of finite sets is countable.

It turns out that there are only two choices for the probability that PH and PSPACE differ relative to a thus-chosen oracle—it's either 0 or 1 and, thus, in order to settle the question of whether the probability is 0 or 1, we have only to show either that the probability is not 1 or is not 0.

**Proposition 8.13**  $\mu(C) > 0$  implies  $\mu(C) = 1$  and  $\mu(C) < 1$  implies  $\mu(C) = 0$ .

**Proposition 8.14** Either  $\mu(C) = 1$  or  $\mu(C) = 0$ . Thus, either  $\text{PH}^A \neq \text{PSPACE}^A$  with probability 1 or  $\text{PH}^A \neq \text{PSPACE}^A$  with probability 0.

We will show that the verdict is 1.

**Theorem 8.15** With probability 1, a random oracle separates PSPACE from PH.

In order to prove the theorem, we need to define the notion of probabilistic circuits. A *probabilistic circuit* is a circuit that takes, in addition to its actual input bits, a special set of bits called *random bits*, each of which is assigned either 0 or 1 with probability  $\frac{1}{2}$ . The output of a probabilistic circuit is thus subject to a probability distribution.

The following lemma states that a circuit family computing the parity function correctly for more than half of the inputs can be made errorless at the expense of small increases in the depth and the size.

**Lemma 8.16** Let  $\{C_n\}_{n \geq 1}$  be a family of depth- $d$ , size- $s(n)$  circuits. Suppose that there is a constant  $\epsilon > 0$  such that, for every  $n \geq 1$ , the proportion of the inputs for which  $C_n(x) \neq \pi_n(x)$  is at most  $\frac{1}{2} - \epsilon$ . Then there exists a family  $\{E_n\}_{n \geq 1}$  of depth- $(d+7)$ , size- $\mathcal{O}(n^\alpha s(n) + n^\beta)$  circuits that correctly computes the parity function, where  $\alpha$  and  $\beta$  are constants depending only on  $\epsilon$ .

**Proof** Let  $\{C_n\}_{n \geq 1}$ ,  $d$ ,  $s(n)$ , and  $\epsilon$  be as in the hypothesis of the lemma. Let  $n \geq 1$ . For each  $x = x_1 \cdots x_n \in \{0, 1\}^n$  and  $y = y_0 \cdots y_n \in \{0, 1\}^{n+1}$ , define

$$H_n(x, y) = z_1 \cdots z_n,$$

where for each  $i$ ,  $1 \leq i \leq n$ ,  $z_i = x_i \oplus y_{i-1} \oplus y_i$ , and define

$$F_n(x, y) = C_n(H_n(x, y)) \oplus y_0 \oplus y_n. \quad (8.5)$$

Note that for every  $x \in \{0, 1\}^n$  and  $y \in \{0, 1\}^{n+1}$

$$\begin{aligned}
\pi_n(H_n(x, y)) &= (x_1 \oplus y_0 \oplus y_1) \oplus (x_2 \oplus y_1 \oplus y_2) \oplus \cdots \oplus (x_n \oplus y_{n-1} \oplus y_n) \\
&= \pi_n(x) \oplus y_0 \oplus y_n.
\end{aligned}$$

By rearranging terms, we have

$$\pi_n(x) = \pi_n(H_n(x, y)) \oplus y_0 \oplus y_n. \quad (8.6)$$

By combining equations 8.5 and 8.6, for all  $x \in \{0, 1\}^n$  and  $y \in \{0, 1\}^{n+1}$ ,

$$C_n(H_n(x, y)) = \pi_n(H_n(x, y)) \iff F_n(x, y) = \pi_n(x). \quad (8.7)$$

Define  $C'_n$  to be a probabilistic circuit that computes  $F_n(x, y)$  given  $x$  as the input and  $y$  as the random bits. For all  $x, z \in \{0, 1\}^n$ , there exist exactly two  $y \in \{0, 1\}^{n+1}$  such that  $z = H_n(x, y)$ . Then, by equation 8.7, for every  $x \in \{0, 1\}^n$ , the probability that  $C'_n(x) \neq \pi_n(x)$  is precisely the proportion of  $z \in \{0, 1\}^n$  such that  $C_n(z) \neq \pi_n(z)$ . So, the error probability of  $C'_n$  is at most  $\frac{1}{2} - \epsilon$ . As the exclusive-or of three bits can be computed by a depth-2, size-5 circuit, we can design  $C'_n$  so that its depth is  $d + 4$  and its size is at most  $\mathcal{O}(s(n) + n)$ .

We will convert  $C'_n$  to a deterministic circuit. Let  $a$  be an integer greater than or equal to  $\frac{1}{2\epsilon}$ . Since  $\epsilon$  can be arbitrary small, we may assume that  $0 < \epsilon < \frac{1}{4}$ , so  $a \geq 2$ . Let  $D_n^{(1)}$  be the circuit that computes the  $\wedge$  of  $\lceil 3a \log n \rceil$  copies of  $C'_n$ , where each copy has its own random bits. Since  $C'_n$  computes  $\pi_n$  with error probability at most  $\frac{1}{2} - \epsilon = \frac{1}{2}(1 - a^{-1})$ , the following conditions hold:

1. For every  $x \in \{0, 1\}^n$ , if  $\pi_n(x) = 1$ , then  $D_n^{(1)}(x) = 1$  with probability at least  $[\frac{1}{2}(1 + a^{-1})]^{\lceil 3a \log n \rceil} \geq n^{-3a+2}$ .
2. For every  $x \in \{0, 1\}^n$ , if  $\pi_n(x) = 0$ , then  $D_n^{(1)}(x) = 1$  with probability at most  $[\frac{1}{2}(1 - a^{-1})]^{\lceil 3a \log n \rceil} \leq n^{-3a-2}$ .
3.  $\text{depth}(D_n^{(1)}) = d + 5$  and  $\text{size}(D_n^{(1)}) = \mathcal{O}((s(n) + n) \log n)$ .

Next let  $D_n^{(2)}$  be the circuit that computes the  $\vee$  of  $n^{3a}$  copies of  $D_n^{(1)}$ , where we attach the copies independent random bits. Then the following conditions hold:

1. For every  $x \in \{0, 1\}^n$ , if  $\pi_n(x) = 1$ , then each input bit to the output gate (which is an  $\vee$  gate) of  $D_n^{(2)}(x)$  becomes 0 with probability at most  $1 - n^{-3a+2}$ , so,  $D_n^{(2)}(x) = 0$  with probability at most  $(1 - n^{-3a+2})^{n^{3a}} = (1 - n^{-3a+2})^{n^{3a-2}n^2}$ . This is at most  $2^{-n^2}$  for  $n \geq 2$ .
2. For every  $x \in \{0, 1\}^n$ , if  $\pi_n(x) = 0$ , then each input bit to the output gate of  $D_n^{(2)}(x)$  becomes 1 with probability at most  $n^{-3a-2}$ , so  $D_n^{(2)}(x) = 0$  with probability at least  $1 - n^{3a}n^{-3a-2} = 1 - n^{-2}$ .
3.  $\text{depth}(D_n^{(2)}) = d + 7$  and  $\text{size}(D_n^{(2)}) = \mathcal{O}(n^{3a}(s(n) + n) \log n)$ .

Next let  $D_n^{(3)}$  be the circuit that computes the  $\wedge$  of  $n$  copies of the complement of  $D_n^{(2)}$ , where the copies are given independent random bits. Then the following conditions hold:

1. For every  $x \in \{0, 1\}^n$ , if  $\pi_n(x) = 1$ , then  $D_n^{(3)}(x) = 1$  with probability at least  $(1 - 2^{-n^2})^n \geq 1 - n2^{-n^2}$ . This is more than  $1 - 2^{-n}$  for  $n \geq 3$ .
2. For every  $x \in \{0, 1\}^n$ , if  $\pi_n(x) = 0$ , then  $D_n^{(3)}(x) = 1$  with probability at most  $(n^{-2})^n$ . This is less than  $2^{-n}$  for  $n \geq 2$ .
3.  $\text{depth}(D_n^{(3)}) = d + 7$  and  $\text{size}(D_n^{(3)}) = \mathcal{O}(n^{3a+1}(s(n) + n) \log n)$ .

Thus,  $D_n^{(3)}$  computes  $\pi_n$  with probability greater than  $1 - 2^{-n}$ . For each  $x \in \{0, 1\}^n$ , let  $R(x)$  be the set of all assignments to the random bits that make  $D_n^{(3)}$  err on input  $x$ . Since  $D_n^{(3)}$  makes an error with probability less than  $2^{-n}$ ,

$$\left\| \bigcup_{x \in \{0, 1\}^n} R(x) \right\| < 2^n 2^{-n} = 1.$$

This implies that there is an assignment to the random bits not belonging to  $R(x)$  for any  $x \in \{0, 1\}^n$ . Let  $r$  be such an assignment. Define  $E_n$  to be the deterministic circuit constructed from  $D_n^{(3)}$  by assigning  $r$  to the random bits. Then  $E_n$  correctly computes  $\pi_n(x)$  for all  $x \in \{0, 1\}^n$ ,  $\text{depth}(E_n) = d + 7$ , and  $\text{size}(E_n) = \mathcal{O}(n^{3a+1}(s(n) + n) \log n) = \mathcal{O}(n^{3a+2}(s(n) + n)) = \mathcal{O}(n^{3a+2}s(n) + n^{3a+3})$  as desired. This proves the lemma.  $\square$

Now we are ready to prove Theorem 8.15. Define  $W(A)$  to be the test language we considered in Sect. 8.2; that is,  $W(A)$  is the set of all  $0^n$  such that  $|\{0, 1\}^n \cap A|$  is odd. Recall that we have constructed an enumeration  $K_1, K_2, \dots$  of relativized predicates specifying alternating quantifications such that, for every  $L$  and  $A$ ,  $L \in \text{PH}^A$  if and only if for some  $s \geq 1$  it holds that  $L = K_s(A)$ .

The following proposition, which we present without a proof, is useful.

**Proposition 8.17** *If there exists  $\epsilon > 0$  such that, for every  $s \geq 1$ ,  $\mu(\{A \mid W(A) \neq K_s(A)\}) > \epsilon$ , then  $\mu(\mathbf{C}) = 1$ .*

We obtain the following corollary from Proposition 8.17.

**Corollary 8.18** *If  $\mu(\mathbf{C}) = 0$ , then there exist  $\epsilon > 0$  and  $s \geq 1$  such that, for every  $n \geq 1$ ,  $\mu(\{A \mid 0^n \in W(A) \iff 0^n \in K_s(A)\}) > \frac{1}{2} + \epsilon$ .*

**Proof of Corollary 8.18** Suppose that  $\mu(\mathbf{C}) = 0$ . Then by taking the contrapositive of Proposition 8.17, we have

$$(\forall \epsilon > 0) (\exists s \geq 1) [\mu(\{A \mid W(A) \neq K_s(A)\}) \leq \epsilon].$$

This is equivalent to

$$(\forall \epsilon : 0 < \epsilon < 1/2) (\exists s \geq 1) [\mu(\{A \mid W(A) = K_s(A)\}) \geq 1 - \epsilon].$$

This implies

$$(\exists \epsilon : 0 < \epsilon < 1/2) (\exists s \geq 1) [\mu(\{A \mid W(A) = K_s(A)\}) > \frac{1}{2} + \epsilon].$$

The condition  $W(A) = K_s(A)$  implies that for every  $n \geq 1$ ,  $0^n \in W(A) \iff 0^n \in K_s(A)$ . So, if  $\mu(\{A \mid W(A) = K_s(A)\}) \geq 1 - \epsilon$  then, for every  $n \geq 1$ ,  $\mu(\{A \mid 0^n \in W(A) \iff 0^n \in K_s(A)\}) > \frac{1}{2} + \epsilon$ . Thus, the statement of the corollary holds.  $\square$  Corollary 8.18

Now the rest of the proof is reminiscent of that of Theorem 8.11.

Assume, to the contrary, that  $\text{PH}^A = \text{PSPACE}^A$  with probability 1; that is,  $\mu(\mathbf{C}) = 0$ . By Corollary 8.18, there exist some real  $\epsilon > 0$  and some integer  $s \geq 1$  such that for every  $n \geq 1$

$$\mu(\{A \mid 0^n \in W(A) \iff 0^n \in K_s(A)\}) > \frac{1}{2} + \epsilon.$$

Select such  $\epsilon$  and  $s$ . Let  $n \geq 1$  and let  $\mathcal{D} = \{A \mid 0^n \in W(A) \iff 0^n \in K_s(A)\}$ . Let  $Q$  be the set of all strings queried by  $K_s$  on input  $0^n$ . Note that for every language  $A$  whether  $0^n \in K_s(A)$  depends only on how  $A$  partitions  $Q$  and whether  $0^n \in W(A)$  and depends only on how  $A$  partitions  $\Sigma^n$ . We claim that  $\Sigma^n \subseteq Q$ . To see why, assume  $\Sigma^n \not\subseteq Q$ . Divide  $\Sigma^n$  into two parts,  $S_1 = \Sigma^n \setminus Q$  and  $S_2 = \Sigma^n \cap Q$ . By assumption,  $S_1$  is nonempty. Since for every  $A$ ,  $0^n \in W(A)$  if and only if the number of elements in  $A \cap \Sigma^n$  is odd, for every  $H \subseteq S_2$ , the number of  $H' \subseteq S_1$  such that  $H \cup H' \in \mathcal{D}$  is equal to the number of  $H' \subseteq S_1$  such that  $H \cup H' \notin \mathcal{D}$ . This implies that  $\mu(\mathcal{D}) = \frac{1}{2}$ , a contradiction. So,  $\Sigma^n \subseteq Q$ .

Let  $m = 2^{n-1}$  and let  $V = \{0y \mid y \in \Sigma^{n-1}\}$ . Then  $\|V\| = m$ . For each  $H \subseteq Q - V$ , count for how many  $H' \subseteq V$  does it hold that  $H \cup H' \in \mathcal{D}$ . Since  $\mu(\mathcal{D}) > \frac{1}{2} + \epsilon$ , by the Pigeonhole Principle there is some  $H \subseteq Q - V$  such that for more than  $\frac{1}{2} + \epsilon$  of  $H' \subseteq V$ ,  $H \cup H' \in \mathcal{D}$ . Pick such an  $H$ .

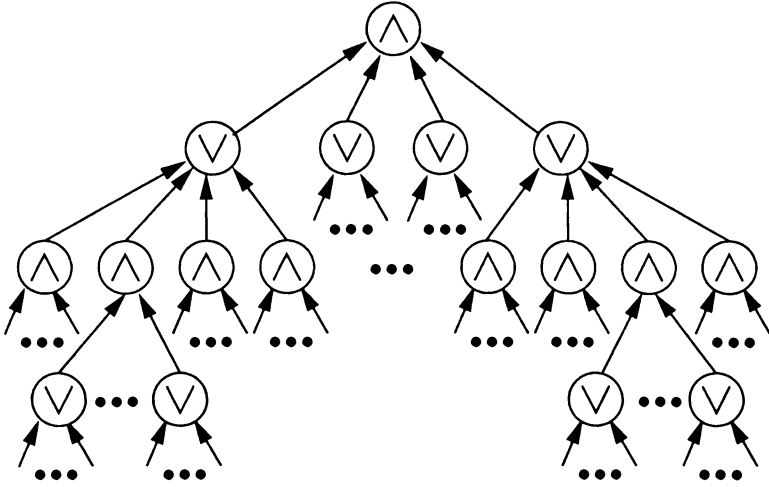
Construct  $C_m$  from the circuit representing the computation of  $K_s$  on  $0^n$  by assigning values to some input variables as follows:

- For each  $w \in H$ , assign 1 to the input  $w$  and 0 to the input  $\bar{w}$ .
- For each  $w \in (Q - V) - H$ , assign 0 to the input  $w$  and 1 to the input  $\bar{w}$ .

Then the proportion of the inputs of length  $m$  for which  $C_m$  computes  $\pi_m$  correctly is more than  $\frac{1}{2} + \epsilon$ . This family, by Lemma 8.16, can be converted to a family of depth- $(k+7)$ , superpolynomial-size circuits that correctly computes the parity function. However, this is impossible by Theorem 8.3. Thus,  $\mu(\mathbf{C}) = 1$ . This proves the theorem.

## 8.4 Oracles That Make the Polynomial Hierarchy Infinite

In this section our major concern is whether there is an oracle relative to which the polynomial hierarchy is infinite. Our viewpoint has been that the


 Fig. 8.3  $h_4^3$ 

relativized polynomial hierarchy is essentially a collection of constant-depth circuits. So, if we can prove that for each  $k \geq 2$  there exists a series of functions computable by depth- $k$ , polynomial-size circuits but not by depth- $(k - 1)$ , superpolynomial-size circuits with small fan-in depth-1 subcircuits, then by using a method similar to that in the proof of Theorem 8.11, we can construct for each  $k \geq 2$  an oracle  $A^{(k)}$  separating  $\Sigma_k^P$  from  $\Sigma_{k-1}^P$ . In the oracle construction for Theorem 8.11 we basically kill each PH machine by identifying a large enough length and then putting some strings of that length in the oracle as well as putting some strings of that length outside the oracle. The procedure can be done in such a way that the lengths that are chosen are widely spaced. Then we can interleave separation procedures of all levels to construct an oracle that separates all the levels of the hierarchy, thereby making the hierarchy infinite.

The following are the magic functions that we use for circuit lower bounds.

**Definition 8.19** Let  $k \geq 2$  and  $m \geq 1$ . Define  $h_m^k$  to be the following function of  $m^k$  variables  $x_1, \dots, x_{m^k}$ :

$$h_m^k(x_1, \dots, x_{m^k}) = (\forall i_1 : 1 \leq i_1 \leq m) (\exists i_2 : 1 \leq i_2 \leq m) \dots (Q_k i_k : 1 \leq i_k \leq m) [x_{(i_1, \dots, i_k)} = 1], \quad (8.8)$$

where  $Q_k = \forall$  if  $k$  is odd and  $\exists$  otherwise and  $(i_1, \dots, i_k)$  denotes the unique number  $i, 1 \leq i \leq m^k$ , whose  $m$ -adic representation is equal to  $i_1 \dots i_k$ . (Here we use the numbers  $1, \dots, m$  instead of  $0, \dots, m - 1$ .)

It is obvious that for each  $k \geq 2$ , and each  $m \geq 1$ , a circuit  $H_m^k$  for computing  $h_m^k$  can be constructed in a straightforward manner by replacing

each occurrence of  $\forall$  in the formula by an  $\wedge$  and each occurrence of  $\exists$  in the formula by an  $\vee$ . The circuit  $H_m^k$  has size  $1 + m + m^2 + \dots + m^k$  and this is less than  $m^{k+1}$ . In order to prove the impossibility result, can we use the distribution  $R_p$  we used for the parity case? We doubt that that is possible. Why? Basically,  $R_p$  is designed to destroy all the depth-2 circuits and so, with high probability, a random restriction under  $R_p$  will not only weaken the superpolynomial size depth- $(k-1)$  circuits but also  $h_m^k$ . Thus, we introduce a new kind of probability distribution for restrictions, in which the probability of assigning 0 can be different from that of assigning 1.

**Definition 8.20** Let  $\Xi$  be a set of  $n$  variables with a fixed enumeration  $x_1, \dots, x_n$ . Let  $r$ ,  $1 \leq r \leq n$ , be an integer. Let  $\mathcal{B} = \{B_1, \dots, B_r\}$  be a partition of  $\Xi$  into nonempty sets. Let  $\Theta = \{s_1, \dots, s_r\}$  be a set of variables each varying over the values 0, 1, and \*. Let  $q$ ,  $0 < q < 1$ , be a real number.

1.  $R_{q,\mathcal{B}}^+$  is the distribution of restrictions  $\rho$  over  $\Xi \cup \Theta$  that are chosen as follows:
  - a) For each  $i$ ,  $1 \leq i \leq r$ ,  $s_i = *$  with probability  $q$  and 0 with probability  $1 - q$ .
  - b) Then for each  $i$ ,  $1 \leq i \leq r$ , and each  $x_j \in B_i$ ,  $\rho(x_j) = \rho(s_i)$  with probability  $q$  and 1 with probability  $1 - q$ .
2. For a restriction  $\rho \in R_{q,\mathcal{B}}^+$ ,  $g(\rho)$  is the restriction  $\sigma$  determined from  $\rho$  as follows: For each  $i$ ,  $1 \leq i \leq r$ , and each  $x_j \in B_i$ ,
  - if  $\rho(x_j) = *$  and for some  $k > j$  it holds that  $x_k \in B_i$  and  $\rho(x_k) = *$ , then  $\sigma(x_j) = 1$ ;
  - otherwise,  $\sigma(x_j) = *$ .
3.  $R_{q,\mathcal{B}}^-$  is defined similarly except that the roles of 0 and 1 are interchanged.
4. For a restriction  $\rho \in R_{q,\mathcal{B}}^-$ ,  $g(\rho)$  is defined similarly except that the roles of 0 and 1 are interchanged.

Here the sets  $B_1, \dots, B_r$  correspond to the blocks of input bits that are fed to the depth-1 subcircuits of  $h_m^k$ . Note that for all  $\rho \in R_{q,\mathcal{B}}^+$  and  $i$ ,  $1 \leq i \leq r$ , the restriction product  $\rho g(\rho)$  assigns \* to at most one variable in  $B_i$ , and if the product assigns \* to exactly one variable, then all the other variables in  $B_i$  are assigned 1. The same property holds for  $R_{q,\mathcal{B}}^-$  with 0 in place of 1.

The following lemma parallels Lemma 8.4. The interested reader may refer to the references we provide at the end of the chapter for its proof.

**Lemma 8.21** Let  $t \geq 1$ , let  $s \geq 1$ , and let  $q$  be such that  $0 < q < 1$ . Let  $G$  be an  $\wedge$ - $\vee$  circuit (respectively, an  $\vee$ - $\wedge$  circuit) with bottom fan-in at most  $t$ . For a random restriction  $\rho$  chosen from  $R_{q,\mathcal{B}}^+$  (respectively, from  $R_{q,\mathcal{B}}^-$ ), the probability that  $G[\rho g(\rho)]$  can be rewritten as an  $\vee$ - $\wedge$  (respectively, an  $\wedge$ - $\vee$  circuit) of bottom fan-in less than  $s$  is at least  $1 - \alpha^s$ , where

$$\alpha = \frac{4q}{2^{\frac{1}{t}-1}} < \frac{4qt}{\log 2} < 6qt.$$

**Theorem 8.22** *Let  $k \geq 2$ . For all but finitely many  $m$ , and for every depth- $k$  circuit  $C$ ,  $C \neq h_m^k$  if  $C$  satisfies the following two conditions:*

1.  $\text{size}(C) \leq 2^{m^{3-k}}$ .
2. Each depth-1 subcircuit of  $C$  is of fan-in at most  $m^{3-k}$ .

**Proof** The proof is by induction on  $k$ . For the base case, let  $k = 2$ . Let  $m \geq 2$ . Suppose  $C$  is a depth-2 circuit satisfying properties 1 and 2 of the theorem. Then  $m^{1/3^2} < m$ , so there is a restriction  $\sigma$  that forces  $C$  to a constant while keeping  $h_m^k$  nonconstant. This implies that  $C \neq h_m^k$ . Thus, the claim holds for  $k = 2$ .

For the induction step, let  $k \geq 3$  and suppose that the claim holds for all  $k'$ ,  $2 \leq k' < k$ . Let  $m \geq 1$  be fixed and let  $C$  be a depth- $k$  circuit satisfying properties 1 and 2 of the theorem. The circuit  $H_m^k$  is built from  $h_m^k$  by translating the formula defining the function. For simplicity, let  $I_1 = \{1, \dots, m^{k-1}\}$  and let  $I_2 = \{1, \dots, m^{k-2}\}$ . Let  $A_1, \dots, A_{m^{k-1}}$  be an enumeration of all depth-1 subcircuits of  $H_m^k$  and let  $D_1, \dots, D_{m^{k-2}}$  be an enumeration of all depth-2 subcircuits of  $H_m^k$ . Note for every  $k \geq 2$  and every  $i \in I_1$  that  $A_i$  is an  $\wedge$ -circuit if  $k$  is odd and an  $\vee$ -circuit if  $k$  is even. For each  $j \in I_2$ , let  $T(j)$  be the set of all  $i \in I_1$  such that  $A_i$  is a subcircuit of  $D_j$ . Then, for every  $j \in I_2$ ,  $T(j) = \{m(j-1) + 1, \dots, mj\}$  and  $|T(j)| = m$ . For each  $i \in I_1$ , let  $B_i$  be the set of all variables appearing in  $A_i$  and  $\mathcal{B} = \{B_1, \dots, B_{m^{k-1}}\}$ . Let  $q = \frac{2}{\sqrt{m}}$  and  $\rho$  be a random restriction chosen under distribution  $R_{q,\mathcal{B}}^+$  if  $k$  is odd and  $R_{q,\mathcal{B}}^-$  if  $k$  is even.

**Fact 8.23** *If  $m$  is sufficiently large, with probability greater than  $\frac{5}{6}$ , the following holds for all  $i \in I_1$ : Either*

1.  $\rho(s_i) = *$  and  $A_i \upharpoonright \rho g(\rho) = x_l$  for some unique variable  $x_l \in B_i$  or
2.  $\rho(s_i) \in \{0, 1\}$  and  $A_i \upharpoonright \rho g(\rho) \equiv \rho(s_i)$ .

**Proof of Fact 8.23** By symmetry we have only to consider the case in which  $k$  is odd. If  $k$  is odd, then  $A_1, \dots, A_{m^{k-1}}$  are  $\wedge$ -circuits and the distribution to be used is  $R_{q,\mathcal{B}}^+$ . Let  $i \in I_1$ . Suppose  $\rho(s_i) = *$ . Then, exactly one of the following is true:  $A_i \upharpoonright \rho g(\rho) = 1$  and for some  $x_l \in B_i$ ,  $A_i \upharpoonright \rho g(\rho) = x_l$ . The former holds with probability  $(1-q)^{|B_i|}$ , so the latter holds with probability  $1 - (1-q)^{|B_i|}$ . Next suppose  $\rho(s_i) = 0$ . Then either  $A_i \upharpoonright \rho g(\rho) \equiv 1$  or  $A_i \upharpoonright \rho g(\rho) \equiv 0$ . Since  $A_i$  is an  $\wedge$  circuit and  $\rho(s_i) = 0$ ,  $A_i \upharpoonright \rho g(\rho) \equiv 1$  if and only if every variable in  $B_i$  is assigned 1. This occurs with probability  $(1-q)^{|B_i|}$ , so  $A_i \upharpoonright \rho g(\rho) = 0$  with probability  $1 - (1-q)^{|B_i|}$ .

Thus, either property 1 or property 2 holds with probability  $1 - (1-q)^{|B_i|}$ . Since  $|I_1| = m^{k-1}$ , the probability that for every  $i \in I_1$  one of the two conditions holds is

$$\begin{aligned} & (1 - (1-q)^{|B_i|})^{m^{k-1}} \\ &= (1 - (1 - m^{-1/2})^m)^{m^{k-1}} \end{aligned}$$

$$\begin{aligned}
&\geq (1 - e^{-\sqrt{m}})m^{k-1} \\
&\geq 1 - m^{k-1}(2^{-\sqrt{m}}) \\
&> \frac{5}{6}
\end{aligned}$$

for sufficiently large  $m$ . This proves the fact.  $\square$  Fact 8.23

**Fact 8.24** *If  $m$  is sufficiently large, with probability greater than  $\frac{5}{6}$ ,*

$$||\{i \in T(j) \mid \rho(s_i) = *\}| \geq \frac{\sqrt{m}}{\ln m} > \lceil m^{1/3} \rceil. \quad (8.9)$$

*holds for all  $j \in I_2$ .*

**Proof of Fact 8.24** Again, by symmetry we have only to consider the case in which  $k$  is odd. Let  $j \in I_2$ . For each  $i \in T(j)$ ,  $\rho(s_i) = *$  with probability  $q = m^{-1/2}$ . For each  $d$ ,  $0 \leq d \leq m$ , let  $p_d$  be the probability that for exactly  $d$  of the  $i \in T(j)$ ,  $\rho(s_i) = *$ , i.e.,  $||\{i \in T(j) \mid \rho(s_i) = *\}| = d$ . Then

$$p_d = \binom{m}{d} (m^{-1/2})^d (1 - m^{-1/2})^{m-d}.$$

The probability that equation 8.9 does not hold for  $j$  is  $p_0 + \dots + p_\nu$ , where  $\nu = \lfloor \frac{\sqrt{m}}{\ln m} \rfloor - 1$ . For all  $m \geq 2$  and  $d$  such that  $2 \leq d \leq m/2$ ,

$$\frac{p_d}{p_{d-1}} = \left( \frac{m-d+1}{d} \right) \left( \frac{m^{-1/2}}{1 - m^{-1/2}} \right) \leq \binom{m}{2} \left( \frac{m^{-1/2}}{\frac{1}{2}} \right) = \sqrt{m}.$$

Also, for every  $m \geq 1$ ,

$$p_0 = (1 - m^{-1/2})^m \leq (1 - m^{-1/2})^{m-1}$$

and

$$p_1 = m(m^{-1/2})(1 - m^{-1/2})^{m-1} = \sqrt{m}(1 - m^{-1/2})^{m-1}.$$

Thus, for every  $m \geq 2$  and  $d$  such that  $0 \leq d \leq m/2$ ,

$$p_d \leq (1 - m^{-1/2})^{m-1} (\sqrt{m})^d.$$

For all  $m \geq 2$ ,  $\nu = \lfloor \frac{\sqrt{m}}{\ln m} \rfloor - 1 < m/2$ . Thus,

$$\begin{aligned}
&p_0 + \dots + p_\nu \\
&\leq (1 - m^{-1/2})^{m-1} \sum_{0 \leq d \leq \nu} (\sqrt{m})^d \\
&\leq (1 - m^{-1/2})^{m-1} (\nu + 1) (\sqrt{m})^\nu \\
&\leq (1 - m^{-1/2})^{m-1} \frac{(\sqrt{m})^{\sqrt{m}/\ln m}}{\ln m} \\
&\leq (1 - m^{-1/2})^{m-1} (\sqrt{m})^{\sqrt{m}/\ln m} \\
&= o(e^{-(\frac{1}{2} + \epsilon)\sqrt{m}} e^{\frac{1}{2}(\ln m)\sqrt{m}/\ln m})
\end{aligned}$$



for some constant  $\epsilon$ ,  $0 < \epsilon < \frac{1}{2}$ . Since  $e^{-(\frac{1}{2}+\epsilon)\sqrt{m}} e^{\frac{1}{2}(\ln m)\sqrt{m}/\ln m} = e^{-\epsilon\sqrt{m}}$ ,

$$p_0 + \cdots + p_\nu = o(e^{-\epsilon\sqrt{m}}).$$

Since  $\|I_2\| = m^{k-2}$ , the probability that equation 8.9 does not hold for some  $j \in I_2$  is

$$o(m^{k-2}e^{-\epsilon\sqrt{m}}).$$

Note that

$$m^{k-2}e^{-\epsilon\sqrt{m}} = e^{-\epsilon\sqrt{m} + (k-2)\ln m} = o(e^{-\epsilon'\sqrt{m}})$$

for some  $\epsilon' > 0$ . So, the probability that equation 8.9 does not hold for some  $j \in I_2$  is

$$o(e^{-\epsilon'\sqrt{m}}).$$

Thus, the probability in question is less than  $\frac{1}{6}$  if  $m$  is large enough.

□ Fact 8.24

**Fact 8.25** *For every  $k \geq 2$ , and for  $m$  sufficiently large, with probability at least  $\frac{2}{3}$ ,  $\rho g(\rho)$  can be extended to a restriction  $\sigma$  such that  $H_m^k[\sigma]$  is equivalent to  $H_{m'}^{k-1}$ , where  $m' = \lceil m^{1/3} \rceil$ .*

**Proof of Fact 8.25** By Facts 8.23 and 8.24, with probability greater than 0, for every  $j \in I_2$ ,  $D_j[\rho g(\rho)]$  is dependent on at least  $\sqrt{m}/\ln m > \lceil m^{1/3} \rceil$  depth-1 subcircuits, each of which is equivalent to a unique variable. So there is some  $\rho$  that makes this happen. Pick such a  $\rho$ . Since  $H_m^k$  is a tree and since for all but finitely many  $m$ ,  $\sqrt{m}/\ln m > \lceil m^{1/3} \rceil$ , we can extend  $\rho g(\rho)$ , by fixing more variables, to a restriction  $\sigma$  in such a way that  $\sigma$  leaves exactly  $m'$  branches at every nonleaf node. Since the fan-in of every level-1 gate is 1, the bottom two levels of this reduced circuit can be collapsed into 1. Thus,  $\sigma$  reduces  $H_m^k$  to  $H_{m'}^{k-1}$ . □ Fact 8.25

Now we are at the final stage of the proof. Suppose that  $m$  is large, and apply Lemma 8.21 to  $C$  with  $s = t = m^{1/3^k}$ . Then with probability greater than or equal to  $1 - \alpha^s$ ,  $\rho g(\rho)$  can be extended to a restriction  $\sigma$  such that each depth-2 subcircuit  $E$  of  $C[\sigma]$  can be rewritten as a circuit  $E'$  satisfying the following conditions:

1.  $E$  is an  $\wedge$ - $\vee$  circuit if and only if  $E'$  is an  $\vee$ - $\wedge$  circuit.
2. Each depth-1 subcircuit of  $E'$  is of fan-in at most  $m^{3^{-k}}$ .

Since  $\alpha < 6qt = 6m^{-1/2}m^{3^{-k}} < m^{-1/4}$ , the probability that the event occurs for all depth-2 subcircuits in  $C$  is at least

$$\begin{aligned} & 1 - (m^{-1/4})^{m^{3^{-k}}} 2^{m^{3^{-k}}} \\ &= 1 - 2^{-\frac{\log m}{4} m^{3^{-k}} + m^{3^{-k}}} \\ &= 1 - \omega(2^{-m^{3^{-k}}}). \end{aligned}$$

So, the probability is more than  $\frac{2}{3}$ . Thus, with probability greater than  $\frac{4}{9}$ , there is a restriction  $\sigma$  depending on  $\rho$  such that

1.  $H_m^k[\sigma]$  is equivalent to  $H_{m'}^{k-1}$  and
2.  $C[\sigma]$  can be converted to a depth  $k-1$  circuit  $C'$  of size at most  $2^{m^{3-k}} \leq 2^{(m')^{3-(k-1)}}$ , each of whose depth-1 subcircuits is of fan-in at most  $m^{3-k} \leq (m')^{3-(k-1)}$ .

By our induction hypothesis,  $H_{m'}^{k-1}$  is not equivalent to  $C'[\sigma]$ . So,  $C$  cannot compute  $H_m^k$ . This proves the theorem.  $\square$

**Theorem 8.26** *There is a relativized world  $A$  in which  $\text{PH}^A$  is infinite.*

**Proof** For each  $k \geq 2$  and each language  $A$ , define  $L_k(A)$  as follows:

$$L_k(A) = \{0^n \mid h_{2^{kn}}^k(\chi_A(x_1), \dots, \chi_A(x_{2^{kn}})) = 1\},$$

where  $x_1, \dots, x_{2^{kn}}$  is an enumeration of all strings of length  $kn$  in increasing lexicographic order. More precisely, for every  $k \geq 2$ , every language  $A$ , and every  $n \geq 1$ ,

$$0^n \in L_k(A) \iff (Q_1 y_1 : y_1 \in \Sigma^n) (Q_2 y_2 : y_2 \in \Sigma^n) \dots (Q_k y_k : y_k \in \Sigma^n) [y_1 \dots y_k \in A],$$

where  $y_1 \dots y_k$  denotes the concatenation of  $y_1, \dots, y_k$  and for each  $i$ ,  $1 \leq i \leq k$ ,  $Q_i = \forall$  if  $i$  is odd and  $\exists$  if  $i$  is even. It is clear from the definition that for every oracle  $A$  and every  $k \geq 2$ ,  $L_k(A) \in \Pi_k^p$ . We construct an oracle  $A$  such that, for every  $k \geq 2$ ,  $L_k(A) \notin \Sigma_{k-1}^p$ . Since for every  $k \geq 2$ ,  $\Sigma_k^p = \Sigma_{k-1}^p$  implies  $\Pi_k^p = \Sigma_{k-1}^p$ , this oracle separates  $\Sigma_k^p$  from  $\Sigma_{k-1}^p$  for all  $k \geq 2$ , and thus makes the polynomial hierarchy infinite. We use the same enumeration  $p_1, p_2, \dots$  of polynomials and the same enumeration  $f_1, f_2, \dots$  of polynomial-time computable functions as we did in the proof of Theorem 8.11. Recall, by Proposition 8.12 that, for every  $k \geq 1$  and for every all  $A \subseteq \Sigma^*$ , if a language  $L$  belongs to  $\Sigma_k^{p,A}$ , then there exist a polynomial  $p_i$  and a polynomial-time computable function  $f_j$  such that, for every  $x$ ,

$$\begin{aligned} x \in L &\iff (Q_1 y_1 : y_1 \in \Sigma^{p_i(|x|)}) \dots (Q_k y_k : y_k \in \Sigma^{p_i(|x|)}) \\ &\quad (Q_{k+1} z : z \in \{1, \dots, p_i(|x|)\}) \\ &\quad [f_j(x, y_1, \dots, y_k, z) \in \bar{A} \oplus A], \end{aligned} \tag{8.10}$$

where for every  $l$ ,  $1 \leq l \leq k+1$ ,  $Q_l = \exists$  if  $l$  is odd and  $Q_l = \forall$  if  $l$  is even. For triples  $s = \langle i, j, k \rangle$  and oracle  $A$ , let  $K_s(A)$  denote the set of all  $x$  satisfying the condition on the right-hand side.

The language  $A$  is constructed in stages. At stage  $s = \langle i, j, k \rangle$  we will identify an integer  $\ell_s$  and extend  $A$  as well as  $\bar{A}$  up to length  $\ell_s$ , so that there exists some integer  $n \geq 0$  such that  $0^n \in L_{k+1}(A) \iff 0^n \notin K_s(A)$ . We assume that for all  $i, j, k \geq 1$ ,  $\langle i, j, k \rangle \geq 1$  and that  $\langle 1, 1, 1 \rangle = 1$ . Let  $\ell_0 = 0$ . We put the empty string in  $\bar{A}$ .

Let  $s = \langle i, j, k \rangle$ . Suppose that we are at the beginning of the stage  $s$ . Let  $A_0$  (respectively,  $A_1$ ) be the set of all strings put in  $\bar{A}$  (respectively,  $A$ ) prior to stage  $s$ . It holds that  $A_0 \cap A_1 = \emptyset$  and  $A_0 \cup A_1 = (\Sigma^*)^{\leq \ell_{s-1}}$ . Consider the following three conditions:

$$n > \ell_{s-1}, \quad (8.11)$$

$$(k+1)2^{(k+1)p_i(n)} < 2^{(2^n)^{3-(k+1)}}, \text{ and} \quad (8.12)$$

$$p_i(n) < (2^n)^{3-(k+1)}. \quad (8.13)$$

Since  $p_i$  is a polynomial and  $k$  is fixed at stage  $s$ , there exists an integer  $n_0$  such that for all  $n \geq n_0$   $n$  satisfies all the three conditions. We claim that for some  $n \geq n_0$  there exists a partition  $(B_0, B_1)$  of  $(\Sigma^*)^{\leq n}$  such that  $B_0 \supseteq A_0$ ,  $B_1 \supseteq A_1$ , and

$$0^n \in L_{k+1}(B_1) \iff 0^n \notin K_s(B_1).$$

We prove the claim by way of contradiction. Assume that the claim does not hold, i.e., for all  $n \geq n_0$  and all partitions  $(B_0, B_1)$  of  $(\Sigma^*)^{\leq n}$  such that  $B_0 \supseteq A_0$  and  $B_1 \supseteq A_1$ , it holds that

$$0^n \in L_{k+1}(B_1) \iff 0^n \in K_s(B_1). \quad (8.14)$$

For each  $n \geq 1$ , let  $\mu(n)$  denote the smallest integer  $l$  such that  $l \geq \ell_{s-1} + 1$  and  $2^l \geq n$ . For each  $n \geq 1$ , the circuit  $C_n$  is constructed from the formula for  $K_s$  on input  $0^{\mu(n)}$ . Let  $\phi_0$  be the formula

$$x \in L \iff (Q_1 y_1 : y_1 \in \Sigma^{p_i(|x|)}) \cdots (Q_k y_k : y_k \in \Sigma^{p_i(|x|)}) \\ (Q_{k+1} z : z \in \{1, \dots, p_i(|x|)\}) [f_j(x, y_1, \dots, y_k, z) \in \bar{A} \oplus A],$$

where for every  $l$ ,  $1 \leq l \leq k+1$ ,  $Q_l = \exists$  if  $l$  is odd and  $Q_l = \forall$  if  $l$  is even. For each  $r$ ,  $1 \leq r \leq k$ , and each  $y_1, \dots, y_r \in \Sigma^{p_i(\mu(n))}$ , let  $\phi_r[y_1, \dots, y_r]$  denote the formula

$$(Q_{r+1} y_{r+1} : y_{r+1} \in \Sigma^{p_i(\mu(n))}) \cdots (Q_k y_k : y_k \in \Sigma^{p_i(\mu(n))}) \\ (Q_{k+1} t : 1 \leq t \leq p_i(\mu(n))) [f_j(0^{\mu(n)}, y_1, \dots, y_k, t) \in \bar{A} \oplus A].$$

For each  $t$ ,  $1 \leq t \leq p_i(\mu(n))$ , and each  $y_1, \dots, y_r \in \Sigma^{p_i(\mu(n))}$ , let  $\phi_{k+1}[y_1, \dots, y_r, t]$  denote the formula

$$f_j(0^{\mu(n)}, y_1, \dots, y_k, t) \in \bar{A} \oplus A.$$

To construct  $C_n$ , for each of the  $\phi$ 's defined in the above, introduce a gate corresponding to it. The type of the gates is determined as follows:

- The node corresponding to  $\phi_0$  is the output gate. The output gate is an  $\vee$  gate.
- Each node corresponding to a  $\phi_{k+1}$  formula is an input gate.
- Let  $1 \leq r \leq k$ . Each node corresponding to a  $\phi_r$  formula is an  $\wedge$  gate if  $r+1$  is even and an  $\vee$  gate if  $r+1$  is odd.

The inputs to the nonleaf gates are determined as follows:

- The inputs of the output gate are the gates corresponding to  $\{\phi_1[y_1] \mid y_1 \in \Sigma^{p_i(\mu(n))}\}$ .
- Let  $1 \leq r \leq k-1$ . Let  $g$  be a gate corresponding to  $\phi_r[y_1, \dots, y_r]$  for some  $y_1, \dots, y_r \in \Sigma^{p_i(\mu(n))}$ . The inputs of  $g$  are  $\{\phi_{r+1}[y_1, \dots, y_{r+1}] \mid y_{r+1} \in \Sigma^{p_i(\mu(n))}\}$ .
- Let  $g$  be a gate corresponding to  $\phi_r[y_1, \dots, y_k]$  for some  $y_1, \dots, y_k \in \Sigma^{p_i(\mu(n))}$ . The inputs of  $g$  are  $\{\phi_{k+1}[y_1, \dots, y_k, t] \mid 1 \leq t \leq p_i(\mu(n))\}$ .

Let  $W_n = \{w_1, \dots, w_{n^{k+1}}\}$  be the smallest  $n^{k+1}$  strings of length  $(k+1)\mu(n)$ . Let  $y_1, \dots, y_k \in \Sigma^{p_i(\mu(n))}$  and  $1 \leq t \leq p_i(\mu(n))$ . Let  $g$  be the input gate corresponding to  $\phi_{k+1}[y_1, \dots, y_k, t]$ . Let  $z = f_j(0^{\mu(n)}, y_1, \dots, y_k, t)$ . The label of  $g$  is determined as follows:

- If for some  $l$ ,  $1 \leq l \leq n^{k+1}$ ,  $z = 1w_l$ , then  $g$  is labeled  $w_l$ .
- If for some  $l$ ,  $1 \leq l \leq n^{k+1}$ ,  $z = 0w_l$ , then  $g$  is labeled  $\overline{w_l}$ .
- If for some  $u \in \Sigma^* - W_n$  it holds that  $z = 1u$ , then  $g$  is assigned 1 if  $u \in A_1$  and is assigned 0 otherwise.
- If for some  $u \in \Sigma^* - W_n$  it holds that  $z = 0u$ , then  $g$  is assigned 0 if  $u \in A_1$  and is assigned 1 otherwise.
- If  $z$  is the empty string,  $g$  is assigned 0.

Then work from the input level to eliminate all subcircuits whose output is a constant regardless of the values of  $w_1, \dots, w_{n^{k+1}}$ . This is  $C_n$ . The circuit  $C_n$  clearly has depth  $k+1$ .

By assumption, for every  $B \subseteq \{w_1, \dots, w_{n^{k+1}}\}$ ,

$$0^n \in L_{k+1}(A_1 \bigcup B) \iff 0^n \in K_s(A_1 \bigcup B).$$

Since

$$0^n \in L_{k+1}(A_1 \bigcup B) \iff h_n^{k+1}(w_1, \dots, w_{n^{k+1}}) = 1$$

and

$$0^n \in K_s(A_1 \bigcup B) \iff C_n(\chi_B(w_1) \cdots \chi_B(w_{n^{k+1}})) = 1,$$

$C_n$  computes  $h_n^{k+1}$ . For all but finitely many  $n \geq n_0$ , both (8.12) and (8.13) hold. So, for all but finitely many  $n$ , the size of  $C_n$  is smaller than  $2^{m^{3-(k+1)}}$  and each depth-1 subcircuit of  $C_n$  has fan-in smaller than  $n^{3-(k+1)}$ . Thus, by Theorem 8.22,  $C_n$  cannot compute  $h_n^{k+1}$ . So, there exists an assignment to  $w_1, \dots, w_{n^{k+1}}$  with respect to which  $C_n$  disagrees with  $h_n^{k+1}$ . This implies that there is a set  $Y \subseteq \{w_1, \dots, w_{n^{k+1}}\}$  such that

$$C_n(\chi_Y(w_1) \cdots \chi_Y(w_{n^{k+1}})) = 1 \iff h_n^{k+1}(\chi_Y(w_1) \cdots \chi_Y(w_{n^{k+1}})) = 0.$$

This is equivalent to

$$0^n \in K_s(B_1) \iff 0^n \notin L_{k+1}^{B_1},$$

where  $B_1 = A_1 \cup Y$ . Let  $r$  be the smallest integer such that for all  $y_1, \dots, y_k \in \Sigma^{p_i(n)}$  and  $t$ ,  $1 \leq t \leq p_i(n)$ ,  $|f_j(\langle 0^n, y_1, \dots, y_k, t \rangle)| \leq r$  and  $B_0 = (\Sigma^*)^{\leq r} - B_1$ . Then  $(B_0, B_1)$  is an extension of  $(A_0, A_1)$ , which contradicts our assumption that equation 8.14 holds for every extension. This proves our claim. So, there is an extension  $(B_0, B_1)$  of  $(A_0, A_1)$  such that equation 8.14 does not hold. Pick such an extension. Set  $\ell_s$  be the smallest integer  $r$  such that  $r \geq \ell_{s-1}$  and for all  $y_1, \dots, y_k \in \Sigma^{p_i(n)}$  and  $t$ ,  $1 \leq t \leq p_i(n)$ ,  $|f_j(\langle 0^n, y_1, \dots, y_k, t \rangle)| \leq r$ . We will set  $A_1$  to  $B_1$  and set  $A_0$  to  $(\Sigma^*)^{\leq r} - B_1$ . Then the property  $L_{k+1}^A \neq K_s(A)$  will be preserved in the future stages. This proves the theorem.  $\square$

**Corollary 8.27** *There is a relativized world in which  $\text{PSPACE} \neq \text{PH}$  and  $\text{PH}$  is infinite.*

## 8.5 OPEN ISSUE: Is the Polynomial Hierarchy Infinite with Probability One?

Does a “probability-one” separation hold for infiniteness of the polynomial hierarchy? Proving such a result seems out of reach as long as we use the function family  $\{h_m^k\}_{k,m \geq 1}$ . In order to apply the method in Sect. 8.3, the function family  $\mathcal{F}$  must possess the following property:

Any deterministic circuit computing  $\mathcal{F}$  with bounded error can be converted, at the cost of constant increase in depth and polynomial increase in size, to a probabilistic circuit computing  $\mathcal{F}$  with bounded error probability.

Our function family  $\{h_m^k\}_{k,m \geq 1}$  seems to lack this property. Can we find another family with this property? The question is subtle. Functions with the property are more or less symmetric, in the sense that the outcome is heavily dependent on the number of 1s in the input bits. In general, symmetric functions, such as the parity function, are “provably” harder than constant-depth, polynomial-size circuits. So, a family endowed with the property seemingly cannot be used to separate the polynomial hierarchy.

## 8.6 Bibliographic Notes

Baker, Gill, and Solovay [BGS75] introduced the concept of oracle separations. They construct an oracle relative to which  $\text{P} = \text{NP}$  and a another oracle relative to which  $\text{P} \neq \text{NP}$ . The meaning and interpretation of oracle—also known as relativization—results has been the topic of interesting discussions [Har85, All90, HCCR90, HCC<sup>+</sup>92, For94]. Though much is known about relativization theory (see [Ver94]), many open problems remain (see, e.g., [HRZ95]).

The random restriction technique was invented independently by Furst, Saxe, and Sipser [FSS84] and by Ajtai [Ajt83]. Both groups proved Theorem 8.1. The exponential-size lower bound for depth-2 circuits, mentioned in the proof of Theorem 8.1, is due to Lupanov [Lup61].

An exponential-size lower bound for parity as well as an oracle separation of PSPACE from PH was first proved by Yao [Yao85]. These two results were improved by Håstad [Hås87, Hås89]. Our presentation in Sect. 8.2 is based on the approach of Håstad.

The notion of random oracles was pioneered in the study by Bennett and Gill [BG81]. They showed that  $P \neq NP$  with probability 1. Propositions 8.17 and 8.13 are from their paper. The probability-one separation of PSPACE from PH is due to Cai [Cai89]. Babai [Bab87] presents a simpler proof built on Håstad's result. Our presentation is based on Babai's proof [Bab87]. Lemma 8.16 is due to Ajtai and Ben-Or [ABO84].

The function  $h_m^k$  of Sect. 8.4 and the biased restriction scheme presented in Sect. 8.4 are both due to Sipser [Sip83]. Using these techniques, Sipser proves a polynomial-size lower bound for constant-depth circuits computing  $h_m^k$ . He conjectured that it is possible to strengthen the result to superpolynomial-size. Yao [Yao85] proves an exponential-size lower bound, but the paper did not contain a proof. Based on this lower bound, Yao constructs an oracle making PH infinite. Håstad provides a complete proof of Yao's lower bound. Actually, Håstad's result [Hås87] significantly improves upon Yao's result. Lemma 8.21 is taken from Håstad's thesis [Hås87]. Our presentation is based on the function proposed by Sipser. This function gives a size lower bound weaker than that of Håstad [Hås87].

The random restriction technique has been widely used as a tool for proving lower bounds and constructing oracles separating complexity classes. For example, Ko [Ko89] constructs, for each  $k \geq 1$ , an oracle that makes the polynomial hierarchy separate up to exactly  $\Sigma_k^P$  while making PSPACE different from (or equal to) the polynomial hierarchy. Sheu and Long [SL94] show that there exists an oracle relative to which, for all  $k \geq 2$ ,  $\Delta_k^P \subsetneq \Sigma_k^P$  and  $\Theta_k^P \subsetneq \Delta_k^P$ . They also prove that the extended low hierarchy is indeed infinite. Bruschi [Bru92] constructs, for every  $k \geq 1$ , an oracle relative to which there exists a set in  $\Sigma_k^P$  that is immune to  $\Delta_k^P$ .

A *perceptron* [MP88] is an AND-OR circuit with a threshold gate at the top. Improving upon Håstad's switching lemma, Green [Gre91] proves an exponential lower bound on the size of constant-depth perceptrons computing parity. Based on the lower bound he proves that there is a relativized world in which  $\oplus P \not\subseteq PP^{PH}$ . Berg and Ulfberg [BU98] construct functions that are computable by linear-size, depth- $k$  boolean circuits and that for no  $k < \log n / (6 \log \log n)$  can be computed by polynomial-size, depth- $(k-1)$  perceptrons. Based on the lower bound they show that there is an oracle  $A$  relative to which, for all  $k \geq 2$ ,  $\Sigma_k^{P,A} \not\subseteq PP^{\Sigma_{k-2}^{P,A}}$ .

Pitassi, Beame, and Impagliazzo [PBI93] obtain an exponential lower bound on the size of bounded-depth Frege proofs for the Pigeonhole Principle and an  $\Omega(\log \log n)$  lower bound on the depth of polynomial-size Frege proofs for the Pigeonhole Principle. Beame, Impagliazzo, and Pitassi [BIP98] show that for no functions  $k(n)$  can the problem of determining whether a given  $n$ -node graph has an  $s$ - $t$  path of length at most  $k(n)$  be solved by polynomial-size circuits of depth  $o(\log \log k(n))$ . There is a survey by Beame [Bea94] that discusses a variety of switching lemmas.





## 9. The Polynomial Technique

One way of understanding the computational flexibility inherent in a complexity class,  $\mathcal{C}$ , is to determine the closure properties the class possesses and lacks. Under which operations is the class closed: complementation? union? intersection? symmetric difference? And under which reducibilities is the class closed? That is, for a class  $\mathcal{C}$ , we may naturally ask: For which reductions  $\leq_r$  does it hold that  $R_r(\mathcal{C}) \subseteq \mathcal{C}$ ? Answering such a question can give insight not just into the computational flexibility of a class but also into the identity of the class. If  $\mathcal{C}$  is closed? under some operation and  $\mathcal{D}$  is not, then  $\mathcal{C} \neq \mathcal{D}$ . And, more typically in the world of complexity, if  $\mathcal{C}$  is closed under some operation and  $\mathcal{D}$  has to date defeated all efforts to prove it closed under that operation, then we may take this as one piece of evidence that may suggest that the classes may differ.

The focus of this chapter is on proving closure properties of PP (and related classes) via construction of low-degree multivariate polynomials of a special kind of counting function, the *gap functions*. Gap functions are those that count the difference between the number of accepting and rejecting computation paths of nondeterministic Turing machines. The breakthrough on these difficult problems came from a novel polynomial construction technique for approximating the sign function. Combining this technique with gap functions gives us relatively easy proofs of the properties.

This chapter is organized as follows. In Sect. 9.1 we introduce GapP and show its closure properties. We demonstrate the usefulness of GapP functions by presenting some simple closure properties of PP and  $\text{C=P}$ . In Sect. 9.2 we introduce an approximation formula for the sign function and use it to prove closure properties of PP. In particular, we prove that PP is closed under intersection and under polynomial-time truth-table reductions. In Sect. 9.3 we introduce GapL, the logarithmic space version of GapP, and we show that the probabilistic logspace hierarchy collapses. In Sect. 9.4 we discuss an important open issue.

## 9.1 GEM: The Polynomial Technique

A GapP function counts the difference between the number of accepting and rejecting computation paths of a nondeterministic Turing machine.

**Definition 9.1** *Let  $M$  be a halting nondeterministic Turing machine, i.e., one that halts on all inputs and along all computation paths. The gap function of  $M$ , denoted by  $\#gap_M$ , is a mapping from  $\Sigma^*$  to  $\mathbb{Z}$  defined for all  $x \in \Sigma^*$  by*

$$\#gap_M(x) = \#acc_M(x) - \#rej_M(x).$$

GapP is the collection of all gap functions of polynomial time-bounded nondeterministic Turing machines.

GapP offers the following new characterization of PP.

**Proposition 9.2** *Let  $L$  be a language.  $L$  belongs to PP if and only if there exists some total function  $f \in \text{GapP}$  such that, for every  $x \in \Sigma^*$ ,  $x \in L$  if and only if  $f(x) \geq 0$ .*

**Proof** Let  $L$  be an arbitrary language. Suppose that  $L$  is in PP. So there exist a polynomial  $p$ , a language  $A \in P$ , and a total function  $f \in FP$  such that, for every  $x \in \Sigma^*$ ,  $x \in L$  if and only if  $|\{y \mid |y| = p(|x|) \wedge \langle x, y \rangle \in A\}| \geq f(x)$ . Define  $M$  to be the nondeterministic Turing machine that, on input  $x \in \Sigma^*$ , guesses  $b \in \{0, 1\}$  and  $y \in \Sigma^{p(|x|)}$ , and accepts  $x$  if either ( $b = 0$  and  $\langle x, y \rangle \in A$ ) or ( $b = 1$  and the rank of  $y$  in  $\Sigma^{p(|x|)}$ —i.e.,  $\{x \mid z \in \Sigma^{p(|x|)} \wedge y \leq_{lex} z\}$ —is at most  $2^{p(|x|)} - f(x)$ ) and rejects otherwise. The machine  $M$  can be made to run in polynomial time. For every  $x \in \Sigma^*$ ,  $M$  on input  $x$  has exactly  $2^{p(|x|)+1}$  computation paths and  $\#acc_M(x)$  is equal to

$$2^{p(|x|)} - f(x) + |\{y \in \Sigma^{p(|x|)} \mid \langle x, y \rangle \in A\}|.$$

This is at least  $2^{p(|x|)}$  if  $x \in L$  and is less than  $2^{p(|x|)}$  otherwise. Since  $2^{p(|x|)}$  is exactly half of  $2^{p(|x|)+1}$ , for every  $x \in \Sigma^*$ ,  $x \in L$  if and only if  $\#gap_M(x) \geq 0$ .

Conversely, suppose that  $L$  is a language and  $f$  is a GapP function such that, for every  $x \in \Sigma^*$ ,  $x \in L$  if and only if  $f(x) \geq 0$ . Let  $M$  be a nondeterministic Turing machine such that  $f = \#gap_M$  and let  $p$  be a polynomial bounding the runtime of  $M$ . Define  $N$  to be the nondeterministic Turing machine that, on input  $x \in \Sigma^*$ , operates as follows:  $N$  simulates  $M$  on input  $x$  while counting in a variable  $C$  the number of nondeterministic moves that  $M$  makes along the simulated path. When  $M$  halts,  $N$  guesses a binary string  $z$  of length  $p(|x|) - C$  using exactly length  $p(|x|) - C$  bits and also guesses a single bit  $b$ . Then  $N$  accepts if and only if either

- $z \in 0^*$  and the simulation path was accepting, or
- $z \notin 0^*$  and  $b = 0$ .

For every input  $x \in \Sigma^*$ , the number of computation paths of  $N$  on input  $x$  is exactly  $2^{p(|x|)+1}$ , and the number of its accepting computation paths is

$$\begin{aligned} & 2\#\text{acc}_M(x) + (2^{p(|x|)} - \#\text{acc}_M(x) - \#\text{rej}_M(x)) \\ &= \#\text{acc}_M(x) - \#\text{rej}_M(x) + 2^{p(|x|)}. \end{aligned}$$

So for every  $x \in \Sigma^*$ ,  $x \in L$  if and only if  $\#\text{acc}_N(x) \geq 2^{p(|x|)}$ . Let  $A = \{\langle x, y \rangle \mid |y| = p(|x|) + 1 \wedge N \text{ on input } x \text{ along path } y \text{ accepts}\}$ . Then, for every  $x$ ,  $x \in L$  if and only if the number of  $y \in \Sigma^{p(|x|)+1}$  such that  $\langle x, y \rangle \in A$  is at least  $2^{p(|x|)}$ . Since the function  $f(x) = 2^{p(|x|)}$  is polynomial-time computable, this implies that  $L \in \text{PP}$ . This proves the proposition.  $\square$

The above characterization simplifies the process of proving containment of languages in PP; we now have only to construct a GapP function that is nonnegative on all members and negative on all nonmembers. It is thus meaningful to know what functions belong to GapP.

### Proposition 9.3

1. Every total mapping in FP from  $\Sigma^*$  to  $\mathbb{Z}$  is a member of GapP.
2. Every function in #P is a member of GapP.
3. Let  $f \in \text{GapP}$  and let total function  $g : \Sigma^* \rightarrow \Sigma^*$  be a member of FP. Let  $h : \Sigma^* \rightarrow \mathbb{Z}$  be defined for all  $x \in \Sigma^*$  by

$$h(x) = f(g(x)).$$

Then  $h \in \text{GapP}$ .

4. Let  $f$  and  $g$  be GapP functions. Let  $h : \Sigma^* \rightarrow \mathbb{Z}$  be defined for all  $x \in \Sigma^*$  by

$$h(x) = f(x) + g(x).$$

Then  $h \in \text{GapP}$ .

In general, for each polynomial  $p$  and each  $f \in \text{GapP}$ , let  $h : \Sigma^* \rightarrow \mathbb{Z}$  be defined for all  $x \in \Sigma^*$  by

$$h(x) = \sum_{|w|=p(|x|)} f(\langle x, w \rangle).$$

5. Let  $f$  and  $g$  be GapP functions. Let  $h : \Sigma^* \rightarrow \mathbb{Z}$  be defined for all  $x \in \Sigma^*$  by

$$h(x) = f(x)g(x).$$

Then  $h \in \text{GapP}$ .

In general, for each polynomial  $p$  and each  $f \in \text{GapP}$ , let  $h : \Sigma^* \rightarrow \mathbb{Z}$  be defined for all  $x \in \Sigma^*$  by

$$h(x) = \prod_{1 \leq i \leq p(|x|)} f(\langle x, i \rangle),$$

where the symbol  $i$  appearing as the second argument on the pairing is a binary encoding of  $i$ . Then  $h \in \text{GapP}$ .

**Proof** (1) Let  $f : \Sigma^* \rightarrow \mathbb{Z}$  be a total function in FP. There exists a polynomial  $p$  such that, for every  $x \in \Sigma^*$ , the absolute value of  $f(x)$  is less than  $2^{p(|x|)}$ . Pick such a  $p$ . Define  $M$  to be the nondeterministic Turing machine that, on input  $x \in \Sigma^*$ , computes  $f(x)$ , guesses  $y \in \Sigma^{p(|x|)}$ , and executes one of the following depending on the sign of  $f(x)$ :

- In the case where  $f(x) \geq 0$ , if the rank of  $y$  in  $\Sigma^{p(|x|)}$  is no greater than  $f(x)$  then  $M$  accepts  $x$ ; otherwise,  $M$  guesses a bit  $b$  and accepts  $x$  if and only if  $b = 0$ ;
- In the case where  $f(x) < 0$ , if the rank of  $y$  in  $\Sigma^{p(|x|)}$  is no greater than  $-f(x)$  then  $M$  rejects  $x$ ; otherwise,  $M$  guesses a bit  $b$  and accepts  $x$  if and only if  $b = 0$ .

Recall that the rank of a string  $y \in \Sigma^{p(|x|)}$  is the number of strings in  $\Sigma^{p(|x|)}$  that are lexicographically less than or equal to  $y$ . The process of guessing a bit  $b$  and accepting if and only if the bit is a 0 generates precisely one accepting path and one rejecting path each time it is applied. Hence the paths that go through this process contribute a sum of 0 to the gap of  $M$ . This implies that for every  $x \in \Sigma^*$ ,  $\#gap_M(x)$  is precisely  $f(x)$  if  $f(x) \geq 0$ , and is precisely  $-1$  times the absolute value of  $f(x)$  if  $f(x) < 0$ . Thus  $\#gap_M = f$ .

(2) We use the same “contribution canceling” technique as in the proof of part 1. Let  $f = \#acc_M$  be a function in  $\#P$ , where  $M$  is some nondeterministic polynomial-time Turing machine. Define  $N$  to be the machine that, on input  $x \in \Sigma^*$ , nondeterministically guesses and simulates a path of  $M$  on input  $x$ , and then executes the following:

- If  $M$  on the path has accepted then  $N$  on that path accepts  $x$ ; otherwise,  $N$  on that path guesses a bit  $b$  and accepts if and only if  $b$  is a 0.

Then, for every  $x \in \Sigma^*$ ,  $\#acc_N(x) = \#acc_M(x) + \#rej_M(x)$  and  $\#rej_N(x) = \#rej_M(x)$ , so  $\#gap_N(x) = \#acc_M(x)$ . Since  $M$  is polynomial time-bounded,  $N$  can be made to run in polynomial time. Thus  $f \in \text{GapP}$ .

(3) Let  $f = \#gap_M$  for some nondeterministic polynomial-time Turing machine  $M$ . Let  $g$  be a total function in FP. Define  $N$  to be the machine that, on input  $x \in \Sigma^*$ , computes  $y = g(x)$ , guesses and simulates a path of  $M$  on input  $y$ , and accepts on the guessed path if and only if  $M$  accepted  $y$  on the guessed path. Then for every  $x \in \Sigma^*$   $\#gap_N(x) = f(g(x))$ . Since  $g$  is polynomial-time computable,  $g$  is polynomially length-bounded, and so  $N$  can be made to run in polynomial time. Thus  $h \in \text{GapP}$ .

(4) We prove the general statement only. Let  $f = \#gap_M$  for some nondeterministic polynomial-time Turing machine  $M$ . Define  $N$  to be the nondeterministic machine that, on input  $x \in \Sigma^*$ , guesses  $w \in \{0, 1\}^{p(|x|)}$ , guesses and simulates a path of  $M$  on input  $\langle x, w \rangle$ , and then accepts on its current path if  $M$  has accepted on that path and rejects on its current path otherwise. Then for every  $x \in \Sigma^*$   $\#gap_N(x) = h(x)$ . Since  $p$  is a polynomial and  $M$  is polynomial time-bounded,  $N$  can be made to run in polynomial time. Thus  $h \in \text{GapP}$ .

(5) We prove the general statement only. Let  $f = \# \text{gap}_M$  for some non-deterministic polynomial-time Turing machine  $M$ . For each  $x \in \Sigma^*$  and  $i$ ,  $1 \leq i \leq p(|x|)$ , let  $S(x, i)$  denote the set of all computation paths of  $f$  on input  $\langle x, i \rangle$  and, furthermore, for each  $\pi \in S(x, i)$ , define  $\alpha(x, i, \pi) = 1$  if  $\pi$  is an accepting computation path and  $-1$  otherwise. Then for each  $x \in \Sigma^*$

$$h(x) = \sum_{\pi_1 \in S(x, 1)} \cdots \sum_{\pi_{p(|x|)} \in S(x, p(|x|))} \alpha(x, 1, \pi_1) \cdots \alpha(x, p(|x|), \pi_{p(|x|)}).$$

Define  $N$  to be the nondeterministic Turing machine that, on input  $x \in \Sigma^*$ , behaves as follows:  $N$  nondeterministically guesses and simulates a path of  $M$  on input  $\langle x, i \rangle$  for all  $i$ ,  $1 \leq i \leq p(|x|)$ . In the course of doing this,  $N$  computes the parity of the number of values of  $i$ ,  $1 \leq i \leq p(|x|)$ , such that  $M$  on input  $\langle x, i \rangle$  rejects. When all the simulations have been completed,  $N$  accepts  $x$  on its current path if and only if this value is even. Note that, for every  $x \in \Sigma^*$ , on the path of  $N$  on input  $x$  corresponding to the guesses  $(\pi_1, \pi_2, \dots, \pi_{p(|x|)})$ , the product  $\alpha(x, 1, \pi_1) \cdots \alpha(x, p(|x|), \pi_{p(|x|)})$  is 1 if and only if  $N$  accepts along that path and that the product is  $-1$  if and only if  $N$  rejects along that path. Thus, for every  $x \in \Sigma^*$ ,  $\# \text{gap}_N(x) = h(x)$ . Since  $p$  is a polynomial and  $M$  is polynomial time-bounded,  $N$  can be made to run in polynomial time. This implies  $h \in \text{GapP}$ .  $\square$

Proposition 9.3 gives an alternative characterization of PP. Let  $f$  be a function in GapP witnessing that a language  $L$  belongs to PP. Define  $g$  for all  $x \in \Sigma^*$  by  $g(x) = 2f(x) + 1$ . The constant functions 2 and 1 are both FP functions, so they are GapP functions by part 1 of Proposition 9.3. So by parts 4 and 5,  $g$  belongs to GapP. For every  $x \in \Sigma^*$ ,  $g(x)$  is always odd and so never equals zero. Also, for every  $x \in \Sigma^*$ ,  $g(x) > 0$  if and only if  $f(x) \geq 0$ . Hence,  $g$  also witnesses that  $L \in \text{PP}$ . Thus we have proved the following result.

**Proposition 9.4** *For every language  $L$ ,  $L$  belongs to PP if and only if there exists a function  $f \in \text{GapP}$  such that, for every  $x \in \Sigma^*$ ,  $f(x) \geq 1$  if  $x \in L$  and  $f(x) \leq -1$  otherwise.*

Based on the above proposition, it is easy to prove that the class PP is closed under complementation. Take an arbitrary language  $L$  in PP. Let  $f$  be a GapP function witnessing the membership of  $L$  in PP as stated in Proposition 9.4. Define  $f' = -f$ . Then  $f' \in \text{GapP}$  (by part 5 of Proposition 9.3, via the constant GapP function  $g(x) = -1$ ) and witnesses that  $\bar{L} \in \text{PP}$  in the sense of Proposition 9.4.

**Proposition 9.5** *PP is closed under complementation.*

The following proposition follows immediately from part 3 of Proposition 9.3.

**Proposition 9.6** *PP is closed under polynomial-time many-one reductions.*

In the next section we will prove various closure properties of PP, proceeding from intersection and union towards polynomial-time constant-round truth-table reductions. In the rest of this section, we demonstrate the usefulness of GapP-based characterizations of complexity classes by presenting some closure properties of  $C=P$ .

Recall that  $C=P$  is the class of languages  $L$  for which there exist a polynomial  $p$  and a language  $A \in P$  such that, for every  $x \in \Sigma^*$ ,  $x \in L$  if and only if the number of  $y \in \Sigma^{p(|x|)}$  such that  $\langle x, y \rangle \in A$  is exactly  $2^{p(|x|)-1}$ . By a proof similar to that of Proposition 9.2, we can obtain the following characterization of  $C=P$  in terms of GapP functions.

**Proposition 9.7** *Let  $L$  be any language.  $L$  belongs to  $C=P$  if and only if there exists some  $f \in \text{GapP}$  such that, for every  $x \in \Sigma^*$ ,  $x \in L$  if and only if  $f(x) = 0$ .*

A simple tweak—squaring the function  $f$  in the above—gives us the strengthened direction of the following proposition.

**Proposition 9.8** *A language  $L$  belongs to  $C=P$  if and only if there exists a nonnegative function  $f \in \text{GapP}$  such that, for every  $x \in \Sigma^*$ ,  $x \in L$  if and only if  $f(x) = 0$ .*

A perceptive reader may notice the similarity between  $C=P$  and  $\text{coNP}$ ; by replacing GapP by  $\#P$  we obtain a definition of  $\text{coNP}$ . Indeed, to the best of our knowledge every closure property possessed by  $\text{coNP}$  is possessed by  $C=P$ , and vice versa, and every collapse of reducibility degrees that holds for  $\text{coNP}$  also holds for  $C=P$  and vice versa. We now give some examples. It is well-known that  $\text{coNP}$  is closed under polynomial-time disjunctive truth-table reductions and under polynomial-time conjunctive truth-table reductions. We show below that these closures hold for  $C=P$ .

**Theorem 9.9**  *$C=P$  is closed under polynomial-time disjunctive truth-table reductions and under polynomial-time conjunctive truth-table reductions.*

**Proof** Let  $A \in C=P$  and take  $f$  to be a GapP function witnessing, in the sense of Proposition 9.8, that  $A \in C=P$ . Suppose that a language  $L$  is reducible to  $A$  via a polynomial-time disjunctive truth-table reduction. That reduction maps each  $x \in \Sigma^*$  to a list of strings,  $g(x)$ , such that  $x \in L$  if and only if at least one member of the list belongs to  $A$ . Define the function  $h$  for all  $x \in \Sigma^*$  by

$$h(x) = \prod_{1 \leq j \leq m} f(y_j),$$

where  $\langle y_1, \dots, y_m \rangle$  is the list  $g(x)$ . Recall that by convention  $\prod_{1 \leq j \leq 0} f(y_j) = 1$ . By parts 3 and 5 of Proposition 9.3,  $h \in \text{GapP}$ . Let  $x \in \Sigma^*$  and let  $g(x) = \langle y_1, \dots, y_m \rangle$ . If  $x \in L$ , then there exists some  $i$ ,  $1 \leq i \leq m$ , such that  $y_i \in A$ . For this  $i$  we have  $f(y_i) = 0$ , and this implies  $h(x) = 0$ . If  $x \in \bar{L}$ , then

there is no  $i$ ,  $1 \leq i \leq m$ , such that  $y_i \notin A$ , and so there is no  $i$ ,  $1 \leq i \leq m$ , such that  $f(y_i) = 0$ . So  $h(x) \neq 0$ . Hence  $g$  witnesses that  $L \in C=P$  in the sense of Proposition 9.8.

For the conjunctive reducibility case, suppose that a language  $L$  is reducible to  $A$  via a polynomial-time conjunctive truth-table reduction. That reduction maps each  $x \in \Sigma^*$  to a list of strings,  $g(x)$ , such that  $x \in L$  if and only if all the members in the list belong to  $A$ . Define the function  $h$  for all  $x \in \Sigma^*$  by

$$h(x) = \sum_{1 \leq j \leq m} f(y_j),$$

where  $\langle y_1, \dots, y_m \rangle$  is the list  $g(x)$ . By parts 3 and 4 of Proposition 9.3,  $h \in \text{GapP}$ . Let  $x \in \Sigma^*$  and let  $g(x) = \langle y_1, \dots, y_m \rangle$ . Suppose  $x \in L$ . Then there is no  $i$ ,  $1 \leq i \leq m$ , such that  $y_i \in \bar{A}$ , so there is no  $i$ ,  $1 \leq i \leq m$ , such that  $f(y_i) > 0$ . Since  $f \geq 0$ , this implies that  $h(x) = 0$ . Suppose  $x \in \bar{L}$ . Then there is some  $i$ ,  $1 \leq i \leq m$ , such that  $y_i \in A$ , so there is some  $i$ ,  $1 \leq i \leq m$ , such that  $f(y_i) > 0$ . Since  $f \geq 0$  this implies that  $h(x) > 0$ . Thus  $L \in C=P$ .  $\square$

A language  $A$  is coNP-many-one reducible to  $B$ , denoted by  $A \leq_m^{\text{comp}} B$ , if there exist a polynomial  $p$  and a polynomial-time computable total function  $g$  such that for all  $x \in \Sigma^*$ ,  $x \in A$  if and only if it holds that  $(\forall y \in \Sigma^{p(|x|)})[g(\langle x, y \rangle) \in B]$ . It is well known that coNP is closed under coNP-many-one reductions. We show below that the closure holds for  $C=P$ .

**Theorem 9.10**  *$C=P$  is closed under coNP-many-one reductions.*

**Proof** Let  $A \in C=P$  and take  $f$  to be a GapP function witnessing, in the sense of Proposition 9.8, that  $A \in C=P$ . Suppose that a language  $L$  is reducible to  $A$  via coNP-many-one reduction. Let  $p$  be a polynomial and let  $g$  be a polynomial-time computable total function witnessing that  $L \leq_m^{\text{comp}} A$ . So for every  $x \in \Sigma^*$ ,  $x \in L$  and only if  $(\forall y \in \Sigma^{p(|x|)})[g(\langle x, y \rangle) \in A]$ . Define the function  $h$  for all  $x$  by

$$h(x) = \sum_{|y|=p(|x|)} f(\langle x, y \rangle).$$

Then  $h$  is in GapP by part 4 of Proposition 9.3. Suppose  $x \in L$ . Then, for all  $y \in \Sigma^{p(|x|)}$ ,  $\langle x, y \rangle \in A$ , so, for all  $y \in \Sigma^{p(|x|)}$ ,  $f(\langle x, y \rangle) = 0$ . Thus,  $h(x) = 0$ . Suppose  $x \in \bar{L}$ . Then, for some  $y \in \Sigma^{p(|x|)}$ ,  $\langle x, y \rangle \in \bar{A}$ , so, for some  $y \in \Sigma^{p(|x|)}$ ,  $f(\langle x, y \rangle) \neq 0$ . Since  $f$  is nonnegative, this implies that  $h(x) > 0$ . Thus,  $h$  witnesses that  $L \in C=P$  in the sense of Proposition 9.8.  $\square$

## 9.2 Closure Properties of PP

The goal of this section is to prove essential closure properties of PP. We first prove the closure of PP under intersection. Then we prove its closure under

polynomial-time truth-table reductions and under polynomial-time constant-round truth-table reductions.

To show that PP is closed under intersection, it would suffice to have a two-variable polynomial  $q$  having only positive coefficients such that, for all integers  $z_1, z_2$ ,  $q(z_1, z_2) > 0$  if and only if  $(z_1 > 0 \text{ and } z_2 > 0)$ . Then using as arguments to  $q$  the GapP representations of  $L_1$  and  $L_2$  would yield a GapP representation of  $L_1 \cap L_2$ . Unfortunately, no such polynomial exists. However, we only need this property to hold for  $|z_1|$  and  $|z_2|$  up to  $2^{p(n)}$  for some appropriate polynomial  $p$ . The following lemma, which is the basis of all the results we prove in this section, addresses this issue. Below, we assume that  $\{0, 1\} \subseteq \Sigma$ .

**Lemma 9.11** *For every  $L \in \text{PP}$  and every polynomial  $r$ , there exist GapP functions  $g : \Sigma^* \rightarrow \mathbb{N}$  and  $h : \Sigma^* \rightarrow \mathbb{N}^+$  such that, for all  $x \in \Sigma^*$ ,  $h(x) > 0$  and, for all  $x \in \Sigma^*$  and  $b \in \{0, 1\}$ ,*

1. *if  $\chi_L(x) = b$ , then  $1 - 2^{-r(|x|)} \leq \frac{g(\langle x, b \rangle)}{h(x)} \leq 1$ , and*
2. *if  $\chi_L(x) = 1 - b$ , then  $0 \leq \frac{g(\langle x, b \rangle)}{h(x)} \leq 2^{-r(|x|)}$ .*

Here  $\chi_L$  is the characteristic function of  $L$ , i.e., for every  $x \in \Sigma^*$ ,  $\chi_L(x) = 1$  if  $x \in L$  and  $\chi_L(x) = 0$  otherwise.

The proof of Lemma 9.11 makes uses of a formula that approximates the sign function of integers, i.e., the function that maps all positive integers to +1 and all negative integers to -1.

**Definition 9.12 (Low-Degree Polynomials to Approximate the Sign Function)** *Let  $m$  and  $r$  be positive integers. Define:*

$$\mathcal{P}_m(z) = (z - 1) \prod_{1 \leq i \leq m} (z - 2^i)^2. \quad (9.1)$$

$$\mathcal{Q}_m(z) = -\mathcal{P}_m(z) - \mathcal{P}_m(-z). \quad (9.2)$$

$$\mathcal{A}_{m,r}(z) = (\mathcal{Q}_m(z))^{2r}. \quad (9.3)$$

$$\mathcal{B}_{m,r}(z) = (\mathcal{Q}_m(z)^{2r}) + (2\mathcal{P}_m(z))^{2r}. \quad (9.4)$$

$$\mathcal{R}_{m,r}(z) = \left( \frac{2\mathcal{P}_m(z)}{\mathcal{Q}_m(z)} \right)^{2r} \quad (9.5)$$

$$\mathcal{S}_{m,r}(z) = (1 + \mathcal{R}_{m,r}(z))^{-1}. \quad (9.6)$$

$\mathcal{R}_{m,r}(z)$  and  $\mathcal{S}_{m,r}(z)$  are two auxiliary functions that will help us understand the properties of  $\mathcal{A}_{m,r}(z)$  and  $\mathcal{B}_{m,r}(z)$ . In the next lemma we explore properties of  $\mathcal{A}_{m,r}(z)$ ,  $\mathcal{B}_{m,r}(z)$ , and  $\mathcal{S}_{m,r}(z)$ .

**Lemma 9.13**

1. *For all positive integers  $m$  and  $r$ ,  $\mathcal{S}_{m,r}(z) = \frac{\mathcal{A}_{m,r}(z)}{\mathcal{B}_{m,r}(z)}$ .*



2. For every positive integers  $m$  and  $r$ , both  $\mathcal{A}_{m,r}(z)$  and  $\mathcal{B}_{m,r}(z)$  are polynomials in  $z$  of degree  $\mathcal{O}(rm)$ .
3. For all integers  $m, r \geq 1$  and every integer  $z$ ,
- if  $1 \leq z \leq 2^m$ , then  $1 - 2^{-r} \leq \mathcal{S}_{m,r} \leq 1$ , and
  - if  $-2^m \leq z \leq -1$ , then  $0 \leq \mathcal{S}_{m,r}(z) \leq 2^{-r}$ .

**Proof** The proofs of (1) and (2) are by routine calculation. We leave them to the reader. To prove (3), let  $m$  and  $r$  be positive integers. First consider the case when  $1 \leq z \leq 2^m$ . In this case  $\mathcal{P}_m(z) \geq 0$  and  $\mathcal{P}_m(-z) < 0$ . We prove the following claim.

**Claim 9.14** *If  $1 \leq z \leq 2^m$ , then  $0 \leq \mathcal{P}_m(z) < -\frac{\mathcal{P}_m(-z)}{9}$ .*

**Proof of Claim 9.14** The claim clearly holds for  $z = 1$ . So suppose  $2 \leq z \leq 2^m$ . There is a unique  $i$ ,  $1 \leq i \leq m$ , such that  $2^i \leq z < 2^{i+1}$ . Let  $t$  be that  $i$ . Then (i)  $2^t \leq z$  and (ii)  $z/2 < 2^t$ . By combining (i) and (ii) we get  $0 \leq (z - 2^t) < \frac{z}{2}$ , and from (ii) we get  $z + 2^t > \frac{3z}{2}$ , and thus,  $\frac{z}{2} < |z - 2^t|/3$ . By combining the two inequalities, we have  $(z - 2^t)^2 < \frac{(z - 2^t)^2}{9}$ . Note that  $z - 1 < z + 1$  and  $|z - 2^i| \leq z + 2^i$  for every  $i$ ,  $1 \leq i \leq m$ . Thus, in light of the definition of  $\mathcal{P}_m$ ,  $\mathcal{P}_m(z) < -\frac{\mathcal{P}_m(-z)}{9}$ .  $\square$  Claim 9.14

Now by the above claim  $0 \leq \mathcal{P}_m(z) < -\frac{\mathcal{P}_m(-z)}{9}$ . Combining this with  $\mathcal{P}_m(-z) \leq 0$  yields  $\mathcal{Q}_m(z) > -\frac{8\mathcal{P}_m(-z)}{9} > 0$ . Thus

$$0 \leq \mathcal{R}_{m,r}(z) < \left( \frac{2(\frac{-1}{9})\mathcal{P}_m(-z)}{-\frac{8}{9}\mathcal{P}_m(-z)} \right)^{2r} = \left( \frac{1}{4} \right)^{2r} < 2^{-r}.$$

Since  $\mathcal{R}_{m,r}(z) \geq 0$  and since for every  $\delta \geq 0$ ,  $(1 + \delta) > 0$  and  $(1 + \delta)(1 - \delta) = 1 - \delta^2 \leq 1$ , we have

$$1 \geq \mathcal{S}_{m,r}(z) = \frac{1}{1 + \mathcal{R}_{m,r}(z)} > 1 - \mathcal{R}_{m,r}(z) > 1 - 2^{-r}.$$

Hence (3a) holds.

Next consider the case when  $-2^m \leq z \leq -1$ . For this range of values of  $z$ , in light of Claim 9.14 we have  $0 \leq \mathcal{P}_m(-z) < -\frac{\mathcal{P}_m(z)}{9}$ . This implies  $0 < \mathcal{Q}_{m,r}(z) < -\mathcal{P}_m(z)$ . Thus

$$\mathcal{R}_{m,r}(z) \geq \left( \frac{2\mathcal{P}_m(z)}{-\mathcal{P}_m(z)} \right)^{2r} = 4^r > 2^r.$$

This implies

$$\mathcal{S}_{m,r}(z) = \frac{1}{1 + \mathcal{R}_{m,r}(z)} < \frac{1}{\mathcal{R}_{m,r}(z)} < 2^{-r}.$$

Hence (3b) holds.  $\square$  Lemma 9.13

**Proof of Lemma 9.11** Now we turn to proving Lemma 9.11. Let  $L \in \text{PP}$  and  $f$  be a GapP function witnessing, in the sense of Proposition 9.4, the membership of  $L$  in PP. Then, for every  $x \in \Sigma^*$ , the absolute value of  $f(x)$  is at least one. Let  $m$  be a polynomial such that, for every  $x \in \Sigma^*$ , the absolute value of  $f(x)$  is at most  $2^{m(|x|)}$ . Let  $r$  be an arbitrary polynomial. Define:

$$\begin{aligned} h(x) &= \mathcal{B}_{m(|x|), r(|x|)}(f(x)), \\ g(\langle x, 1 \rangle) &= \mathcal{A}_{m(|x|), r(|x|)}(f(x)), \quad \text{and} \\ g(\langle x, 0 \rangle) &= \mathcal{B}_{m(|x|), r(|x|)}(f(x)) - \mathcal{A}_{m(|x|), r(|x|)}(f(x)). \end{aligned}$$

Then, for every  $x \in \Sigma^*$ ,  $g(\langle x, 0 \rangle) + g(\langle x, 1 \rangle) = h(x)$ . For every  $x \in \Sigma^*$ , by part 1 of Lemma 9.13,  $\mathcal{S}_{m(|x|), r(|x|)}(f(x)) = \frac{g(\langle x, 1 \rangle)}{h(x)}$  and since  $g(\langle x, 0 \rangle) + g(\langle x, 1 \rangle) = h(x)$ ,  $1 - \mathcal{S}_{m(|x|), r(|x|)}(f(x)) = \frac{g(\langle x, 0 \rangle)}{h(x)}$ . So, by Lemma 9.13 and the first claim of the previous sentence,  $1 - 2^{-r(|x|)} \leq \frac{g(\langle x, 1 \rangle)}{h(x)} \leq 1$  if  $f(x) > 0$  and  $2^{-r(|x|)} \geq \frac{g(\langle x, 1 \rangle)}{h(x)} \geq 0$  if  $f(x) < 0$ . Since  $\frac{g(\langle x, 0 \rangle)}{h(x)} = 1 - \frac{g(\langle x, 1 \rangle)}{h(x)}$ , we have  $1 - 2^{-r(|x|)} \leq \frac{g(\langle x, 0 \rangle)}{h(x)} \leq 1$  if  $f(x) < 0$ , and  $2^{-r(|x|)} \geq \frac{g(\langle x, 0 \rangle)}{h(x)} \geq 0$  if  $f(x) > 0$ . Now it remains to prove that both  $g$  and  $h$  are in GapP, but this is easy to prove because we have Proposition 9.3. We will leave that verification as an exercise for the reader.  $\square$  Lemma 9.11

### 9.2.1 PP Is Closed Under Intersection

All the groundwork has now been done, and so we may give the proof of the closure of PP under intersection.

**Theorem 9.15** *PP is closed under intersection.*

**Proof** Let  $L$  and  $L'$  be arbitrary languages in PP. Let  $r$  be the constant polynomial 2. Let  $g$  and  $h$  be the two functions given by Lemma 9.11 for  $L$  and  $r$  and let  $g'$  and  $h'$  be those for  $L'$  and  $r$ . For every  $x \in \Sigma^*$ , the following conditions hold:

- $\frac{3}{4} \leq \frac{g(\langle x, 1 \rangle)}{h(x)} \leq 1$  if  $x \in L$  and  $0 \leq \frac{g'(\langle x, 1 \rangle)}{h'(x)} \leq \frac{1}{4}$  otherwise.
- $\frac{3}{4} \leq \frac{g'(\langle x, 1 \rangle)}{h'(x)} \leq 1$  if  $x \in L'$  and  $0 \leq \frac{g(\langle x, 1 \rangle)}{h(x)} \leq \frac{1}{4}$  otherwise.

Define

$$\rho(x) = \frac{g(\langle x, 1 \rangle)}{h(x)} + \frac{g'(\langle x, 1 \rangle)}{h'(x)} - \frac{3}{2}.$$

For every  $x \in \Sigma^*$ , if  $x \in L \cap L'$ , then the first two terms of  $\rho(x)$  are both greater than or equal to  $\frac{3}{4}$  so  $\rho(x) \geq 2(\frac{3}{4}) - \frac{3}{2} = 0$ ; and if either  $x \notin L$  or  $x \notin L'$ , then one of the two is at most  $\frac{1}{4}$  and both are at most 1, so  $\rho(x) \leq \frac{5}{4} - \frac{3}{2} < 0$ . So for every  $x \in \Sigma^*$ ,  $x \in L \cap L'$  if and only if  $\rho(x) \geq 0$ . Define the function  $T$  by  $T(x) = 2h(x)h'(x)\rho(x)$ . So

$$T(x) = 2(g(\langle x, 1 \rangle)h'(x) + g'(\langle x, 1 \rangle)h(x)) - 3h(x)h'(x).$$

Since  $h$  and  $h'$  are both positive, for every  $x \in \Sigma^*$ ,  $x \in L \cap L'$  if and only if  $T(x) \geq 0$ . By Proposition 9.3,  $T \in \text{GapP}$ . Thus, by Proposition 9.2,  $L \cap L' \in \text{PP}$ .  $\square$

For every  $L$  and  $L'$ ,  $L \cup L' = \overline{\overline{L} \cap \overline{L'}}$ . Thus, in light of Proposition 9.5, we have the following corollary to Theorem 9.15.

**Corollary 9.16** *PP is closed under union.*

Corollary 9.17 extends these results to obtain a much stronger conclusion than closure under union or intersection. The reason that Corollary 9.17 implies closure under union and intersection is that PP is closed under disjoint union, and clearly  $A \cup B \leq_{2\text{-tt}}^p A \oplus B$  and  $A \cap B \leq_{2\text{-tt}}^p A \oplus B$ .

**Corollary 9.17** *PP is closed under polynomial-time bounded-truth-table reductions.*

**Proof** Let  $A \in \text{PP}$  and  $L$  be  $\leq_{k\text{-tt}}^p$ -reducible to  $A$  for some  $k \geq 1$ . It follows, via augmenting Appendix B's definition with the fact that we can add on ignored queries in such a way as to make a  $k\text{-tt}$  reduction always ask exactly  $k$  queries, that there is a polynomial time-bounded Turing machine  $M$  such that on each input  $x \in \Sigma^*$ ,  $M$  generates a list of strings  $\langle y_1, \dots, y_k \rangle$  and a  $k$ -ary boolean function  $\alpha$ , such that  $x \in L$  if and only if

$$\alpha(\chi_A(y_1), \dots, \chi_A(y_k)) = 1.$$

The  $y$ 's and  $\alpha$  are functions of  $x$ . For each  $k$ -bit string  $b = b_1 \dots b_k$ , define  $S_b$  to be the set of all  $x \in \Sigma^*$  such that the boolean function  $\alpha$  that  $M$  generates on  $x$  has the value 1 at  $b$ . For each  $i$ ,  $1 \leq i \leq k$ , define  $T_i^{(1)}$  to be the set of all  $x \in \Sigma^*$  such that the  $i$ th string that  $M$  on input  $x$  generates belongs to  $A$ . Similarly, define  $T_i^{(0)}$  with  $\overline{A}$  in place of  $A$ . By Proposition 9.6 PP is closed under  $\leq_m^p$ -reductions, so for every  $i$ ,  $1 \leq i \leq k$ ,  $T_i^{(1)}$  belongs to PP. Furthermore, since PP is closed under complementation, for every  $i$ ,  $1 \leq i \leq k$ ,  $T_i^{(0)}$  is in PP. Now

$$L = \bigcup_{b_1 \dots b_k} \left( S_{b_1 \dots b_k} \cap \bigcap_{1 \leq i \leq k} T_i^{(b_i)} \right).$$

Since PP is closed under intersection and  $k$  is a constant, each  $S_{b_1 \dots b_k} \cap \bigcap_{1 \leq i \leq k} T_i^{(b_i)}$  is in PP. So since PP is closed under union and  $2^k$  is a constant,  $L \in \text{PP}$ .  $\square$

### 9.2.2 PP Is Closed Under Truth-Table Reductions

The following result is so general that it implies, as each consequences, all the results of Sect. 9.2.1.

**Theorem 9.18** *PP is closed under polynomial-time truth-table reductions.*

Some preparation is necessary for the proof of Theorem 9.18. Below, we assume that  $\{0, 1\} \subseteq \Sigma$ . We fix a scheme for encoding any nonempty list of strings into a string. Let  $\#$  be a symbol not in  $\Sigma$  and let  $\tilde{\Sigma} = \Sigma \cup \{\#\}$ . For each integer  $k \geq 1$ , and  $k$  strings  $y_1, \dots, y_k$ , let

$$\Lambda_k(y_1, \dots, y_k) = y_1\# \cdots \#y_k.$$

Note that  $\Lambda_k$  is a mapping from  $(\Sigma^*)^k$  to  $(\tilde{\Sigma})^*$  and is polynomial-time computable and polynomial-time invertible. For each  $k \geq 1$ , let  $U_k$  be the range of  $\Lambda_k$ , i.e.,  $\{w \mid w \in \tilde{\Sigma}^* \text{ and for some } y_1, \dots, y_k \in \Sigma^*, w = \Lambda_k(y_1, \dots, y_k)\}$ . Note that for all positive integers  $k, l, k \neq l$ , each element in  $U_k$  has exactly  $k - 1$  occurrences of  $\#$  and each element in  $U_l$  has exactly  $l - 1$  occurrences of  $\#$ , and thus,  $U_k$  and  $U_l$  are disjoint. So, given a string  $w \in \bigcup_{k \geq 1} U_k$  one can compute in polynomial time the integer  $k \geq 1$  such that  $w \in U_k$  and the unique list of strings  $[y_1, \dots, y_k]$  such that  $w = \Lambda_k(y_1, \dots, y_k)$ .

A *query generator* is a Turing machine that maps each input string to a nonempty list of strings encoded using  $\Lambda$ . A query generator  $g$  is *length-increasing* if for all  $x \in \Sigma^*$  each element in the list that  $g(x)$  encodes has at least  $|x|$  bits. For a query generator  $g$  and a language  $A$ , define  $\Gamma_g^A$  to be the function that maps each  $x \in \Sigma^*$  to  $\chi_A(y_1) \cdots \chi_A(y_k)$ , where  $g(x) = \Lambda_k(y_1, \dots, y_k)$ .

Let  $M$  be a polynomial-time machine computing a truth-table reduction of some language  $L$  to some language  $A$ . Then there is a polynomial-time machine computing a truth-table reduction of  $L$  to  $A$  that makes at least one query for each input. To see this, define  $N$  to be the oracle Turing machine that on each input  $x$  does the following:  $N$  simulates  $M$  on input  $x$  and then accepts  $x$  if  $M$  accepts and rejects otherwise, but just before accepting or rejecting  $N$  checks whether a query is made during the current simulation, and if no query is made,  $N$  makes a query about the empty string and ignores the oracle's answer. Since  $N$  makes the additional ignored query exactly in the case when  $M$  does not make a query,  $N$  computes a truth-table reduction. Furthermore,  $N$  is clearly polynomial time-bounded, makes at least one query for each input, and for each oracle, accepts the same language as  $M$  does with the oracle. In particular,  $L(N^A) = L(M^A)$ . The addition of one query may deprive  $N$  of some properties about the queries  $M$ , but this is fine.

The following proposition summarizes the above discussion.

**Proposition 9.19** *If a language  $L$  is polynomial-time truth-table reducible to a language  $A$ , then there exists a polynomial-time oracle Turing machine  $M$  computing a truth-table reduction from  $L$  to  $A$  such that for each input  $x \in \Sigma^*$ ,  $M$  makes at least one query to its oracle.*

Let  $M$  be a polynomial-time oracle Turing machine computing a truth-table reduction from a language  $L$  to a language  $A$  in the sense of Proposition 9.19. Since  $M$  makes at least one query for each input, we can split

the program of  $M$  into two parts, query generation and evaluation of oracle answers. More precisely, we have the following proposition.

**Proposition 9.20** *If a language  $L$  is polynomial-time truth-table reducible to a language  $A$ , then there exist some polynomial-time query generator  $g$  and some polynomial-time computable evaluator  $e : \Sigma^* \times \Sigma^* \rightarrow \{0, 1\}$  such that, for each  $x \in \Sigma^*$ ,*

$$x \in L(N^A) \iff e(x, \Gamma_g^A(x)) = 1.$$

Below is a key technical lemma that is used to prove Theorem 9.18.

**Lemma 9.21** *Let  $g$  be a polynomial-time computable, length-increasing query generator and let  $A$  be a language in PP. For every polynomial  $r$  there exist GapP functions  $s : \Sigma^* \rightarrow \mathbb{N}$  and  $t : \Sigma^* \rightarrow \mathbb{N}^+$  such that, for every  $x, w \in \Sigma^*$ , the following two conditions hold:*

- *If  $w = \Gamma_g^A$ , then  $1 - 2^{-r(|x|)} \leq \frac{s(\langle x, w \rangle)}{t(x)} \leq 1$ .*
- *If  $w \neq \Gamma_g^A$ , then  $0 \leq \frac{s(\langle x, w \rangle)}{t(x)} \leq 2^{-r(|x|)}$ .*

Throughout this and the following sections we will use the term *natural polynomials* to refer to univariate polynomials with positive coefficients. For every univariate polynomial  $p$  that is not natural there is a natural polynomial  $p'$  such that, for every  $n$ ,  $p(n) \leq p'(n)$ . Note that every natural polynomial is strictly increasing on domain  $[0, \infty)$ ; i.e., for every integer  $n \geq 0$ , the value of the polynomial at  $n$  is less than the value of the polynomial at  $n+1$ . Note also that the class of natural polynomials is closed under addition, multiplication, and composition. More precisely, for all natural polynomials  $p$  and  $q$ , the polynomials  $p(n) + q(n)$ ,  $p(n)q(n)$ , and  $p(q(n))$  are natural polynomials.

**Proof of Lemma 9.21** Let  $g$  and  $A$  be as in the statement of the lemma. Define  $m$  to be the function that maps each  $x \in \Sigma^*$  to the number of elements (not necessarily distinct) in the list that  $g(x)$  encodes. Let  $p$  be a polynomial such that, for all  $x \in \Sigma^*$ ,  $m(x) \leq p(|x|)$ . Let  $r$  be an arbitrary polynomial. Define  $q$  to be a natural polynomial such that, for all  $n$ ,  $q(n) \geq p(n) + r(n)$ . By Lemma 9.11, there exist GapP functions  $s_0 : \Sigma^* \rightarrow \mathbb{N}$  and  $t_0 : \Sigma^* \rightarrow \mathbb{N}^+$  such that, for all  $y \in \Sigma^*$  and  $b \in \{0, 1\}$ ,

- $1 - 2^{-q(|y|)} \leq \frac{s_0(\langle y, b \rangle)}{t_0(y)} \leq 1$  if  $\chi_A(y) = b$ , and
- $0 \leq \frac{s_0(\langle y, b \rangle)}{t_0(y)} \leq 2^{-q(|y|)}$  otherwise.

Define  $s : \Sigma^* \times \Sigma^* \rightarrow \mathbb{N}$  for all  $x, w \in \Sigma^*$  by

- if  $|w| \neq m(x)$  then  $s(\langle x, w \rangle) = 0$ , and
- if  $|w| = m(x)$  then  $s(\langle x, w \rangle) = \prod_{1 \leq i \leq m(x)} s_0(\langle y_i, w_i \rangle)$ ;

and then define  $t : \Sigma^* \rightarrow \mathbb{N}^+$  for all  $x \in \Sigma^*$  by

$$t(x) = \prod_{1 \leq i \leq m(x)} t_0(y_i),$$

where  $g(x) = \Lambda_{m(x)}(y_1, \dots, y_{m(x)})$ . Then for all strings  $x, w \in \Sigma^*$  the following conditions hold:

- $s(\langle x, w \rangle) \geq 0$  and  $t(x) > 0$ .
- If  $|w| \neq m(x)$ , then  $\frac{s(\langle x, w \rangle)}{t(x)} = 0$ .
- If  $|w| = m(x)$  and  $w \neq \Gamma_g^A(x)$ , then  $0 \leq \frac{s(\langle x, w \rangle)}{t(x)} \leq 2^{-\min\{q(|y_1|), \dots, q(|y_{m(x)}|)\}}$ . Since  $q$  is natural the upper bound is at most  $2^{-q(\min\{|y_1|, \dots, |y_{m(x)}|\})}$ . Since  $g$  is length-increasing and  $q$  is natural, this bound is at most  $2^{-q(|x|)} \leq 2^{-r(|x|)}$ .
- If  $w = \Gamma_g^A(x)$ , then  $1 \geq \frac{s(\langle x, w \rangle)}{t(x)} \geq 1 - m(x)2^{-\min\{q(|y_1|), \dots, q(|y_{m(x)}|)\}}$ . Since  $q$  is natural the lower bound is at least  $1 - m(x)2^{-q(\min\{|y_1|, \dots, |y_{m(x)}|\})}$ . Since  $g$  is length-increasing and  $q$  is natural, the bound is at least  $1 - p(|x|)2^{-q(|x|)}$ . Since  $q(n) = p(n) + r(n)$  and  $p(n) < 2^{p(n)}$  for all  $n$ , the lower bound is at least  $1 - 2^{-r(|x|)}$ .

This proves the lemma. □ Lemma 9.21

Now we turn to proving Theorem 9.18.

**Proof of Theorem 9.18** Let  $A \in \text{PP}$ . Let  $L$  be  $\leq_{tt}^p$ -reducible to  $A$ . By Proposition 9.19 there is a polynomial-time machine,  $M$ , computing a truth-table reduction from  $L$  to  $A$  that makes at least one query for each input, and by Proposition 9.20, there exist some polynomial-time query generator  $g$  and some polynomial-time evaluator  $e$  that jointly achieve the same effect as  $M$ . Without loss of generality, we may assume that  $g$  is length-increasing. If not we will replace  $A$  by  $A' = \{0^i 1 y \mid i \geq 0 \wedge y \in A\}$  and, for all  $x \in \Sigma^*$ , replace the value of  $g(x)$  by  $\Lambda_k(0^{|x|} 1 y_1, \dots, 0^{|x|} 1 y_k)$ , where  $g(x) = \Lambda_k(y_1, \dots, y_k)$ .  $A'$  is in PP, and this altered analog of  $g$  remains polynomial-time computable and is a length-increasing query generator.

Define  $m$  to be the function that maps each  $x \in \Sigma^*$  to the number of (not necessarily distinct) elements in the list  $g(x)$  encodes. Let  $p$  be a natural polynomial such that, for every  $x \in \Sigma^*$ ,  $m(x) \leq p(|x|)$ . Define  $r(n) = p(n) + 2$ . By Lemma 9.21, there exist GapP functions  $s : \Sigma^* \times \Sigma^* \rightarrow \mathbb{N}$  and  $t : \Sigma^* \rightarrow \mathbb{N}^+$  such that, for all  $x, w \in \Sigma^*$ , the following two conditions hold:

- The fraction  $\frac{s(\langle x, w \rangle)}{t(x)}$  is between  $1 - 2^{-r(|x|)}$  and 1 if  $w = \Gamma_g^A(x)$ .
- The fraction  $\frac{s(\langle x, w \rangle)}{t(x)}$  is between 0 and  $2^{-r(|x|)}$  otherwise.

Define  $s^* : \Sigma^* \rightarrow \mathbb{N}$  for all  $x \in \Sigma^*$  by

$$s^*(x) = \sum_{|w|=m(x)} e(x, w) s(\langle x, w \rangle).$$

Then for every  $x \in \Sigma^*$  the following conditions hold:

- If  $x \in L$ , then  $e(x, \Gamma_g^A) = 1$ , so that  $\frac{s^*(x)}{t(x)}$  is at least  $1 - 2^{-r(|x|)} = 1 - 2^{-p(|x|)+2} > \frac{3}{4}$ .
- If  $x \notin L$  then  $e(x, \Gamma_g^A) = 0$  and for every  $w \neq \Gamma_g^A$ ,  $\frac{s(\langle x, w \rangle)}{t(x)}$  is between 0 and  $2^{-r(|x|)}$ . Since there are at most  $2^{p(|x|)}$  many  $w$  of length  $m(x)$ ,  $\frac{s^*(x)}{t(x)}$  is between 0 and  $2^{p(|x|)}2^{-r(|x|)} = \frac{1}{4}$ .

Thus, for every  $x \in \Sigma^*$ ,  $x \in L$  if and only if  $\frac{s^*(x)}{t(x)} \geq \frac{1}{2}$ . Now define  $H(x) = 2s^*(x) - t(x)$ . Then, for every  $x \in \Sigma^*$ ,  $x \in L$  if and only if  $H(x) \geq 0$ . It is easy to see that  $H(x)$  is in GapP. Hence  $L \in \text{PP}$ . This concludes the proof of Theorem 9.18.  $\square$  Theorem 9.18

### 9.2.3 PP Is Closed Under Constant-Round Truth-Table Reductions

We finally extend the closure property of PP to its strongest known form.

Let  $k \geq 1$  be an integer. A language  $A$  is polynomial-time  $k$ -round truth-table reducible to a language  $B$  (write  $A \leq_{tt[k]}^p B$ ) if there exists a polynomial time-bounded oracle Turing machine  $M$  such that  $M$  makes exactly  $k$  rounds of parallel queries to its oracle and such that, for every  $x \in \Sigma^*$ ,  $x \in A$  if and only if  $M^B$  on input  $x$  accepts. Let  $M$  be a machine that computes a polynomial-time  $k$ -round truth-table reduction. We envision that  $M$  executes a query round as follows:

- Let  $[y_1, \dots, y_m]$  be the list of queries to be made. The machine  $M$  writes on the query tape  $y_1\# \dots \#y_m\#$ , where  $\#$  is a symbol not in  $\Sigma$ . Then the machine  $M$  enters the query state.
- Let  $b = b_1 \dots b_m$ , where for every  $i$ ,  $1 \leq i \leq m$ ,  $b_i = 1$  if  $y_i$  is a member of the oracle and  $b_i = 0$  otherwise. In the next computation step, the following two things occur. (i) The word on the query tape is (by magic) replaced by  $b$  so that the first letter of  $b$  is stored in cell 1 of the tape. (ii) The head on the query tape is moved to cell 1.

Now we prove the theorem that states the closure property of PP in its strongest form.

**Theorem 9.22** *For every  $k \geq 1$ , PP is closed under polynomial-time  $k$ -round truth-table reductions.*

**Proof of Theorem 9.22** The proof is by induction on  $k$ . The base case is  $k = 1$ , which we have already proven as Theorem 9.18. For the induction step, let  $k = k_0 > 1$  and suppose that the claim holds for all values of  $k$  less than  $k_0$ . Let  $L$  be  $\leq_{tt[k]}^p$ -reducible to  $A \in \text{PP}$  via a polynomial-time Turing machine  $M$ . We may assume that for every  $x \in \Sigma^*$ ,  $M$  on input  $x$  makes

exactly  $k$  rounds of queries regardless of its oracle. We will replace  $M$  by a new machine,  $M'$ , that on each input  $x$  simulates  $M$  on input  $x$  with the following two additional tasks: (1) If  $M$  enters the query state for the  $(k+1)$ st time, then  $M'$  rejects  $x$  without executing the current step of  $M$ . (2) If  $M$  halts and if  $M$  made fewer than  $k$  query rounds,  $M'$  executes additional ignored query rounds so as to make the total number of query rounds on input  $x$  equal to  $k$ , and then accepts or rejects according to the outcome of the simulation of  $M$  on input  $x$ . The two additional tasks do not increase the computation time significantly. Since  $M$  is polynomial time-bounded the new machine  $M'$  can be made polynomial time-bounded.

By “configuration” of  $M'$  we mean an object consisting of the contents of its tapes, the position of its heads, and its state. A configuration describes the exact condition the machine  $M'$  is in. Suppose that  $M'$  has  $h$  work tapes. Let  $\Gamma$  be the union of the tape alphabets of  $M'$ , which includes  $\{0, 1\}$ . Let  $\$$  be a symbol not in  $\Gamma$  and  $\Gamma' = \Gamma \cup \{\$\}$ . Let  $d$  be the smallest integer such that the cardinality of  $\Gamma'$  is at most  $2^d$ . Fix a  $d$ -bit encoding of the symbols in  $\Gamma'$ . Take any configuration  $S$  of the machine  $M'$ , where  $M'$  has  $x$  on the input tape with the head located on cell  $r_1$ , for each  $i$ ,  $1 \leq i \leq h$ ,  $w_i$  on the  $i$ th work tape with the head located on cell  $p_i$ , and  $y$  on the query tape with the head located on cell  $r_2$ . This configuration  $S$  is encoded by replacing each symbol of  $\Gamma'$  appearing in the word

$$r_1 \$ x \$ r_2 \$ y \$ q \$ p_1 \$ w_1 \$ \cdots \$ p_h \$ w_h$$

by its  $d$ -bit encoding. Since  $M'$  is polynomial time-bounded, there is a polynomial  $p$  such that, for every  $x \in \Sigma^*$ , and for every configuration  $S$  of  $M'$  on input  $x$ , the encoding length of  $S$  is at most  $p(|x|)$ .

Define  $C$  to be the set of all strings  $\langle x, i, b \rangle$  that satisfy the following two conditions:

- $x \in \Sigma^*$ ,  $i \geq 1$ , and  $b \in \{0, 1\}$ .
- The binary encoding of the configuration of  $M'$  on input  $x$  relative to  $A$  when its  $(k-1)$ st round of queries has length at least  $i$  and  $b$  is equal to the  $i$ th bit of the encoding.

Then  $C$  is polynomial-time  $(k-1)$ -round truth-table reducible to  $A$ , so by our induction hypothesis,  $C$  belongs to PP. We claim that  $L$  is polynomial-time two-round truth-table reducible to  $A \oplus C$ . To prove this claim, note that the configuration of  $M'$  on an input  $x \in \Sigma^*$  with oracle  $A$  can be computed via parallel queries

$$\langle x, 1, 0 \rangle, \dots, \langle x, p(|x|), 0 \rangle, \langle x, 1, 1 \rangle, \dots, \langle x, p(|x|), 1 \rangle$$

to the  $C$  part of the oracle, where  $p$  is a polynomial such that, for every  $x \in \Sigma^*$  and every oracle  $Q$ ,  $M^Q$  on input  $x$  halts within  $p(|x|)$  steps. If the encoding length of the configuration is  $d$ , then the oracle gives a positive answer to precisely one of  $\langle x, i, 0 \rangle$  and  $\langle x, i, 1 \rangle$  for each  $i$ ,  $1 \leq i \leq d$ , and to



neither of them for any  $i$ ,  $d + 1 \leq i \leq p(|x|)$ . Once the configuration has been computed, the query strings in the  $k$ th round can be computed in time polynomial in  $|x|$ . Then the rest of the computation of  $M'$  can be simulated with a single parallel query round to the  $A$  part of the oracle.

We have proven the conclusion inductively for  $k > 2$ , but not for  $k = 2$ . We need to show that PP is closed under polynomial-time two-round truth-table reductions. Let  $L$  be  $\leq_{tt[2]}^p$ -reducible to some set  $A \in \text{PP}$  via machine  $M$ . As discussed in the first part of the proof, we may assume that for every  $x \in \Sigma^*$ ,  $M$  on input  $x$  makes precisely two rounds of queries regardless of its oracle. Furthermore, we may assume that there is a natural polynomial  $p$  such that, for every  $x \in \Sigma^*$ ,  $M$  on input  $x$  makes at each round precisely  $p(|x|)$  queries, each of length  $p(|x|)$ , regardless of its oracle. To see why, let  $p$  be a natural polynomial strictly bounding the runtime of  $M$ . We will replace  $A$  by its padded version  $A' = \{0^n 1x \mid n \geq 0 \wedge x \in A\}$ . Then, for each input  $x \in \Sigma^*$ , a query  $y$  to  $A$  can be replaced by the query  $0^{p(|x|)-|y|-1}1y$  to  $A'$  (we here are tacitly using the fact that  $p(|x|) \geq |y| + 1$  for each such query  $y$ ). Then we modify  $M$  so that before entering the query state  $M$  checks whether the number of query strings currently placed on the query tape is  $p(|x|)$ . If not,  $M$  appends sufficiently many copies of a dummy query  $0^{p(|x|)}$  to the list to make the number equal to  $p(|x|)$ . Call this new machine  $M'$ .

Define  $B = \{\langle x, i \rangle \mid x \in \Sigma^* \wedge 1 \leq i \leq p(|x|) \wedge \text{the } i\text{th query of } M' \text{ on input } x \text{ with oracle } A' \text{ at the second round is in } A'\}$ . Then  $L$  is  $\leq_{tt}^p$ -reducible to  $A' \oplus B$ . Since  $A \in \text{PP}$ ,  $A'$  belongs to PP. Thus it suffices to show that  $B \in \text{PP}$ .

Since  $A'$  is in PP there exist a polynomial time-bounded nondeterministic Turing machine  $N$  and a natural polynomial  $q$  such that, for every  $x \in \Sigma^*$ ,  $N$  on input  $x$  has precisely  $2^{q(|x|)}$  computation paths and  $N$  is such that  $x \in A'$  if and only if  $\#\text{acc}_N(x) \geq 2^{q(|x|)-1}$ . Define  $r(n) = p(n) + q(p(n)) + 1$ . Let  $g$  be the query generator corresponding to the first query round of  $M'$ . By Lemma 9.21, there exist GapP functions  $s \geq 0$  and  $t > 0$  such that for all strings  $x$  and  $w$ ,  $x, w \in \Sigma^*$ , the fraction  $\frac{s(\langle x, w \rangle)}{t(x)}$  is between 1 and  $1 - 2^{-r(|x|)}$  if  $w = \Gamma_g^{A'}$  and is between 0 and  $2^{-r(|x|)}$  otherwise. Define, for all  $x \in \Sigma^*$  and  $i$ ,  $1 \leq i \leq p(|x|)$ ,

$$s^*(x, i) = \sum_{|w|=p(|x|)} \#\text{acc}_N(z(x, i, w)) \frac{s(\langle x, w \rangle)}{t(x)}.$$

Here  $z(x, i, w)$  denotes the  $i$ th query that would be made by  $M'$  on input  $x$  in the second round in the case when the answers to the first round queries are provided by the string  $w$ . Put another way, for each  $j$ ,  $1 \leq j \leq p(|x|)$ , the answer to the  $j$ th query is treated as being affirmative if the  $j$ th bit of  $w$  is 1 and treated as being negative otherwise.

We claim that for all  $x \in \Sigma^*$  and  $i$ ,  $1 \leq i \leq p(|x|)$ ,  $\langle x, i \rangle \in B$  if and only if  $\frac{s^*(x, i)}{t(x)} \geq 2^{q(p(|x|))-1} - \frac{1}{2}$ . To prove this claim suppose  $\langle x, i \rangle \in B$ . Since

$z(x, i, \Gamma_g^{A'})$  is precisely the  $i$ th query in the second round of  $M'$  on input  $x$  with  $A'$  as the oracle,  $\#acc_N(z(x, i, \Gamma_g^{A'}))$  is at least  $2^{q(|z(x, i, \Gamma_g^{A'})|)-1} = 2^{q(p(|x|))-1}$ . Since  $\frac{s(\langle x, \Gamma_g^{A'}(x) \rangle)}{t(x)} \geq 1 - 2^{-r(|x|)}$  and all the terms appearing in  $s^*$  are nonnegative,  $s^*(x, i) \geq 2^{q(p(|x|))-1}(1 - 2^{-r(|x|)})$ . This is equal to

$$2^{q(p(|x|))-1} - 2^{q(p(|x|))-p(|x|)-q(p(|x|))-1} = 2^{q(p(|x|))-1} - 2^{-p(|x|)-1},$$

and this is more than  $2^{q(p(|x|))-1} - \frac{1}{2}$ . On the other hand, suppose  $\langle x, i \rangle \notin B$ . Then  $\#acc_N(z(x, i, \Gamma_g^A(x))) \leq 2^{q(p(|x|))-1} - 1$ . For every  $w \neq \Gamma_g^A$  of length  $p(|x|)$ ,  $\#acc_N(z(x, i, w))$  is at most  $2^{q(p(|x|))}$  and  $\frac{s(x, w)}{t(x)}$  is at most  $2^{-r(|x|)}$ . Since the number of  $w \neq \Gamma_g^A$  is  $2^{p(|x|)} - 1$ ,  $\frac{s^*(x, i)}{t(x)}$  is at most

$$\begin{aligned} & (2^{q(p(|x|))-1} - 1) + (2^{p(|x|)} - 1)2^{q(p(|x|))}2^{-r(|x|)} \\ & < 2^{q(p(|x|))-1} - 1 + 2^{p(|x|)}2^{q(p(|x|))}2^{-r(|x|)} \\ & = 2^{q(p(|x|))-1} - 1 + 2^{p(|x|)+q(p(|x|))-p(|x|)-q(p(|x|))-1} \\ & = 2^{q(p(|x|))-1} - \frac{1}{2}. \end{aligned}$$

Thus the claim holds.

Now define

$$H(\langle x, i \rangle) = \begin{cases} 2s^*(x, i) - t(x) & \text{if } 1 \leq i \leq p(|x|), \\ 0 & \text{otherwise.} \end{cases}$$

Then by the above claim, for every  $x \in \Sigma^*$  and  $i$ ,  $1 \leq i \leq p(|x|)$ ,  $\langle x, i \rangle \in B$  if and only if  $H(\langle x, i \rangle) \geq 0$ . It is easy to see that  $H$  is a GapP function. Hence  $B \in \text{PP}$ . This proves the theorem.  $\square$  Theorem 9.22

Does PP have even stronger closure properties than that expressed as Theorem 9.22? The logical next step would be for Theorem 9.22 to extend to nonconstant  $k$ , perhaps  $k = \mathcal{O}(\log n)$ . If  $k$  can be outright arbitrary then we are in effect asking whether PP is closed under polynomial-time Turing reductions, i.e., whether  $\text{PP} = \text{P}^{\text{PP}}$ . This is an open question, and asserts that two classes in the “counting hierarchy”  $\text{P}, \text{PP}, \text{P}^{\text{PP}}, \text{PP}^{\text{PP}}, \text{P}^{\text{PP}^{\text{PP}}}, \dots$  are equal, i.e.,  $\text{P}^{\text{PP}} = \text{PP}$  (which is *not* known to imply other equalities in that hierarchy). Nonetheless, we show in the following section that the analogous hierarchy based on logspace does collapse.

## 9.3 The Probabilistic Logspace Hierarchy Collapses

### 9.3.1 GapL Functions and PL

Recall that a language  $L$  belongs to PL if and only if there is a logarithmic space-bounded, polynomial time-bounded probabilistic Turing machine  $M$

such that, for every  $x \in \Sigma^*$ ,  $x \in L$  if and only if the probability that  $M$  on input  $x$  accepts is at least  $\frac{1}{2}$ . Recall also that  $\#L$  is the class of all functions  $f$  such that for some logarithmic space-bounded, polynomial time-bounded nondeterministic Turing machine  $M$  it holds that  $f = \#acc_M$ . We define the logarithmic-space version of GapP, namely, GapL.

**Definition 9.23**  $\text{GapL} = \{\#gap_M \mid M \text{ is a logarithmic space-bounded, polynomial time-bounded nondeterministic Turing machine}\}$ .

Proposition 9.24 and Lemma 9.25 are analogous to Proposition 9.3 and Lemma 9.11, respectively. Note, however, that part 4 of Proposition 9.24 deals with only polynomially many terms. The proofs of Proposition 9.24 and Lemma 9.25 are quite similar to those of their polynomial-time versions, and thus are omitted.

**Proposition 9.24**

1. Let  $f : \Sigma^* \rightarrow \mathbb{Z}$  be a logspace-computable, total function. Then  $f$  belongs to GapL.
2.  $\#L \subseteq \text{GapL}$ .
3. Let  $f \in \text{GapL}$  and let  $g : \Sigma^* \rightarrow \Sigma^*$  be a logspace-computable function. Define  $h$  for all  $x \in \Sigma^*$  by

$$h(x) = f(g(x)).$$

Then  $h \in \text{GapL}$ .

4. Let  $f$  and  $g$  be GapL functions. Define  $h$  for all  $x \in \Sigma^*$  by

$$h(x) = f(x) + g(x).$$

Then  $h \in \text{GapL}$ . In general, for each polynomial  $p$  and  $f \in \text{GapL}$ , define  $h$  for all  $x \in \Sigma^*$  by

$$h(x) = \sum_{1 \leq i \leq p(|x|)} f(\langle x, i \rangle).$$

Then  $h \in \text{GapL}$ .

5. Let  $g$  and  $g$  be GapL functions. Define  $h$  for all  $x \in \Sigma^*$  by

$$h(x) = f(x)g(x).$$

Then  $h \in \text{GapL}$ . In general, for each polynomial  $p$  and  $f \in \text{GapL}$ , define  $h$  for all  $x \in \Sigma^*$  by

$$h(x) = \prod_{1 \leq i \leq p(|x|)} f(\langle x, i \rangle).$$

Then  $h \in \text{GapL}$ .

**Lemma 9.25** *For each  $L \in \text{PL}$  and each polynomial  $r$ , there exist GapL functions  $g : \Sigma^* \rightarrow \mathbb{N}$  and  $h : \Sigma^* \rightarrow \mathbb{N}^+$  such that, for all  $x \in \Sigma^*$ ,*

1. *if  $\chi_L(x) = b$ , then  $1 - 2^{-r(|x|)} \leq \frac{g(\langle x, b \rangle)}{h(x)} \leq 1$ , and*
2. *if  $\chi_L(x) \neq b$ , then  $0 \leq \frac{g(\langle x, b \rangle)}{h(x)} \leq 2^{-r(|x|)}$ .*

The PL hierarchy is defined under a restriction, called the Ruzzo–Simon–Tompia relativization (the RST relativization), that stipulates that nondeterministic space-bounded oracle Turing machines must behave deterministically during query generation. We call a logarithmic space-bounded, polynomial time-bounded machine working under that restriction an RSTNL machine.

**Definition 9.26** *A language  $L$  belongs to PL relative to an oracle  $A$  if there exists an RSTNL machine  $M$  such that, for every  $x \in \Sigma^*$ ,*

$$x \in L \iff \# \text{gap}_{M^A}(x) \geq 0.$$

For any class  $\mathcal{C}$ ,  $\text{PL}^{\mathcal{C}} = \{\text{PL}^A \mid A \in \mathcal{C}\}$ .

**Definition 9.27** *The PL hierarchy PLH is defined as follows, where relativization is interpreted in the sense of Definition 9.26.*

$$\text{PLH} = \text{PL} \bigcup \text{PL}^{\text{PL}} \bigcup \text{PL}^{\text{PL}^{\text{PL}}} \bigcup \dots \quad (9.7)$$

This hierarchy collapses to PL.

**Theorem 9.28**  $\text{PLH} = \text{PL}$ .

To prove the theorem it suffices to show that the second level of the hierarchy,  $\text{PL}^{\text{PL}}$ , collapses to PL. The proof is reminiscent of that of Theorem 9.22, but more careful treatment is required because of the space bounds imposed on RSTNL machines.

### 9.3.2 Oblivious Oracle NL Machines

We say that an oracle Turing machine is *oblivious* if its queries are dependent solely on the input, not on the oracle. Let  $M$  be any RSTNL machine. We can turn  $M$  into an equivalent oblivious RSTNL machine as follows.

Without loss of generality, we may assume that  $M$  has exactly one work tape. Let  $c$  be an integer constant such that, for every nonempty  $x \in \Sigma^*$ ,  $M$  on  $x$  uses cells  $1, \dots, c \log |x|$  on the work tape for storage. Here  $\log |x|$  is shorthand for  $\lceil \log_2 |x| \rceil$ , and we will use this notation throughout this section. Cells 0 and  $c \log |x| + 1$  hold special delimiters to caution  $M$ 's finite control not to move the head out of that region; thus the position of  $M$ 's head ranges from 0 to  $c \log |x| + 1$ . We assume that the same treatment is applied to the input tape so that the range of the input-tape head is between 0 and  $|x| + 1$ . We also assume that the query tape of  $M$  is write only, that the cells of the

query tape are numbered  $0, 1, \dots$ , and that each time  $M$  enters the query state, the word written on the query tape is submitted to the oracle, the tape is blanked (by magic), the head is moved to cell 0, and then  $M$  enters the state  $q_{YES}$  if the word belongs to the oracle and the state  $q_{NO}$  otherwise.

Now we modify the behavior of  $M$  in three steps. First we require that there is a polynomial  $p$  such that, for all  $n \geq 0$ ,  $p(n) > 1$ , and such that, for each  $x \in \Sigma^*$ ,  $M$  on input  $x$  makes precisely  $p(|x|)$  queries on every computation path. To meet this requirement let  $p(n)$  be a natural polynomial with a positive constant term such that  $M$  is  $p(n)$  time-bounded. Since  $p(n)$  is natural, for all  $n \geq 0$ ,  $p(n) \geq p(0)$ . Since the constant term of  $p(n)$  is positive,  $p(n) > 0$ . Thus, for all  $n \geq 0$ ,  $p(n) > 0$ . We modify  $M$  so that it counts the number of queries that it has made so far and adds dummy queries (e.g., about the empty string) just before halting to make the number of queries equal to  $p(|x|)$ . Call this new machine  $M_1$ . Since  $p$  is a polynomial, counting the number of queries requires only  $\mathcal{O}(\log |x|)$  space. Thus  $M_1$  is an RSTNL machine.

Next we require that  $M_1$  have a special state  $q_{gen}$  such that  $M_1$  enters  $q_{gen}$  exactly when it is about to begin generation of a query. To meet this requirement we modify  $M_1$  so that it keeps track of the position of the query tape head and, whenever the action it is about to take involves shifting of that head from cell 0 to 1 it puts off the move for one step and gets in and out of  $q_{gen}$ . Call this new machine  $M_2$ . Then  $M_2$  is an RSTNL machine.

Finally we replace each query of  $M_2$  by a sequence of queries consisting of all potential queries of  $M_2$ . For each natural number  $n \geq 1$ , let  $\mathcal{I}_n$  be the set of all query IDs (instantaneous descriptions) of  $M$  in which the state is  $q_{gen}$ . More precisely,  $\mathcal{I}_n$  is the set of all triples  $(i, j, w)$  such that  $0 \leq i \leq |x| + 1$ ,  $0 \leq j \leq c \log |x| + 1$ , and  $w \in \Upsilon^{c \log |x|}$ , where  $\Upsilon$  is the set of all symbols (including “blank”) that may appear on cells  $1, \dots, c \log |x|$  of  $M$ . A query ID  $(i, j, w)$  represents the situation in which  $M_2$  is in state  $q_{gen}$  and its input-tape head is positioned on cell  $i$ , its work-tape head is positioned on cell  $j$ , and the contents of the work tape on cells  $1, \dots, c \log |x|$  are  $w$ . Since  $M_2$  generates its query deterministically, for every  $x \in \Sigma^*$ , all possible queries of  $M_2$  on  $x$  can be generated one after another by cycling through all IDs  $I \in \mathcal{I}_{|x|}$  and simulating  $M_2$  on  $x$  from  $I$  until  $M_2$  enters the query state. Such an enumeration requires only  $\mathcal{O}(\log n)$  tape cells.

So, we now modify  $M_2$  to construct a new, two-tape machine  $N$  that behaves as follows on an input  $x \in \Sigma^*$ :  $N$  simulates  $M_2$  on  $x$  using tape 1 while keeping track of  $M_2$ 's state  $q$ , input-tape head position  $i$ , work-tape head position  $j$ , and work-tape contents  $w$  on tape 2. When  $M_2$  accepts or rejects  $N$  does the same. Whenever  $M_2$  enters state  $q_{gen}$ ,  $N$  does the following:

**Step 1**  $N$  records the current values of  $i, j$ , and  $w$  on tape 2. These values will be referred to as  $i_{mem}$ ,  $j_{mem}$ , and  $w_{mem}$ , respectively.

**Step 2**  $N$  continues the simulation until  $M_2$  enters the query state, but  $N$  avoids writing on the query tape by keeping the head on cell 0.

**Step 3** Instead of immediately entering the query state,  $N$  suspends the simulation of  $M_2$  and makes all the potential queries of  $M_2$  on input  $x$ . This is carried out by simulating  $M_2$  on input  $x$  from each query ID in  $\mathcal{I}_{|x|}$ . For each  $i$ ,  $0 \leq i \leq |x| + 1$ , each  $j$ ,  $0 \leq j \leq c \log |x| + 1$ , and each  $w \in \Upsilon^{c \log |x|}$ ,

- (a)  $N$  simulates  $M_2$  on input  $x$  from ID  $(i, j, w)$  until  $M_2$  enters the query state to generate the query string corresponding to that ID; and
- (b) if  $i = i_{\text{mem}}$ ,  $j = j_{\text{mem}}$ , and  $w = w_{\text{mem}}$ , then  $N$  records the answer from the oracle into a variable  $\text{ans}$ .

**Step 4**  $N$  returns to the simulation of  $M_2$  on  $x$  that had been suspended at the beginning of Step 3 with  $\text{ans}$  as the oracle answer.

Note that in Step 3b there is always exactly one combination of  $i$ ,  $j$ , and  $w$  that passes the three equality tests, and that generates the same query that  $M_2$  would have made during the simulation in Step 2. Hence the value of  $\text{ans}$  is precisely the answer that  $M_2$  would have obtained. As there is no extra nondeterministic move that  $N$  makes, the only difference between  $M_2$  and  $N$  is that  $N$  inflates its query sequence. Thus, for every  $x \in \Sigma^*$  and every oracle  $A$ ,  $\# \text{gap}_{NA}(x) = \# \text{gap}_{MA}(x)$ . Define  $m(n) = p(n) \|\mathcal{I}_n\|$ . Then  $m$  is bounded by some polynomial. For every  $x \in \Sigma^*$ , the number of queries that  $N$  on  $x$  makes is precisely  $m(|x|)$  and the query sequence is the same on all computation paths. Thus,  $N$  is an oblivious RSTNL machine.

### 9.3.3 Collapsing the PL Hierarchy

Theorem 9.28 follows immediately from the following claim.

**Proposition 9.29**  $\text{PL}^{\text{PL}} = \text{PL}$ .

**Proof** Let  $L \in \text{PL}^{\text{PL}}$  via an oblivious oracle machine  $N$  and an oracle  $A \in \text{PL}$ . As we did in the proof of Theorem 9.18, we can assume that  $N$ 's query strings are longer than its input. Let  $p$  be a polynomial bounding the runtime of  $N$ . Let  $q$  be a natural polynomial such that, for every  $x \in \Sigma^*$ ,

- $\# \text{acc}_{NA}(x) \leq 2^{q(|x|)}$ , and
- $x \in L \iff \# \text{acc}_{NA}(x) \geq 2^{q(|x|)-1}$ .

Let  $m$  be a polynomially bounded function, as defined above, that maps each integer  $n$  to the number of queries that  $N$  makes on each input of length  $n$ . Then  $m(n) \leq p(n)$  for all  $n$ . For each  $x \in \Sigma^*$  and  $i$ ,  $1 \leq i \leq m(|x|)$ , let  $y_{x,i}$  denote the  $i$ th query string of  $N$  on  $x$ . Pick a natural polynomial  $r$  such that  $r(n) \geq p(n) + q(n)$ . For each  $x \in \Sigma^*$  and  $w$ ,  $|w| = m(|x|)$ , let  $\alpha(x, w)$  denote the number of accepting computation paths that  $M$  on input  $x$  would have if for every  $i$ ,  $1 \leq i \leq m(|x|)$ , the oracle answer to the  $i$ th query of  $N$  on input  $x$  is taken to be affirmative if the  $i$ th bit of  $w$  is a 1 and otherwise is

taken to be negative. Then, for every  $x \in \Sigma^*$ ,  $\alpha(x, \Gamma_N^A(x)) = \#\text{acc}_{NA}(x)$ , and for every  $w \in \Sigma^{m(|x|)}$ ,  $\alpha(x, \Gamma_N^A(x)) \leq 2^{q(|x|)}$ . Since  $N$  is an oblivious RSTNL machine we can view it as a query generator, so we may use  $\Gamma_N^A(x)$  to denote the answer sequence that the oracle  $A$  provides to  $N$  on input  $x$ .

By Lemma 9.25, there exist nonnegative function  $g \in \text{GapL}$  and a strictly positive function  $h \in \text{GapL}$  such that, for all  $x \in \Sigma^*$  and  $b \in \{0, 1\}$ ,

- $1 \geq \frac{g(\langle x, b \rangle)}{h(x)} \geq 1 - 2^{-r(|x|)}$  if  $\chi_A(x) = b$ , and
- $0 \leq \frac{g(\langle x, b \rangle)}{h(x)} \leq 2^{-r(|x|)}$  otherwise.

Define, for each  $x \in \Sigma^*$ ,

$$s(x) = \sum_{|w|=m(|x|)} \alpha(x, w) \prod_{1 \leq i \leq m(|x|)} g(\langle y_{x,i}, w_i \rangle)$$

and

$$t(x) = \prod_{1 \leq i \leq m(|x|)} h(y_{x,i}).$$

We claim that, for every  $x \in \Sigma^*$ ,

$$x \in L \iff \frac{s(x)}{t(x)} \geq 2^{q(|x|)-1} - \frac{1}{4}.$$

To prove the claim let  $x \in \Sigma^*$ . First suppose  $x \in L$ . For  $w = \Gamma_N^A(x)$ , the fraction

$$\kappa(x, w) = \frac{\prod_{1 \leq i \leq m(|x|)} g(\langle y_{x,i}, w_i \rangle)}{t(x)}$$

is at least

$$1 - m(|x|)2^{-\min\{r(|y_{x,1}|), \dots, r(|y_{x,m(|x|)}|)\}}.$$

Because  $m$  is bounded by  $p$ , because  $r$  is a natural polynomial, and because the machine  $N$  is a length-increasing query generator, the above amount is at least

$$1 - p(|x|)2^{-p(|x|)-q(|x|)-1} > 1 - 2^{-q(|x|)-1}.$$

Thus the fraction  $\frac{s(x)}{t(x)}$  is at least

$$2^{q(|x|)-1}(1 - 2^{-q(|x|)-1}) = 2^{q(|x|)-1} - \frac{1}{4}.$$

Next suppose that  $x \notin L$ . For  $w = \Gamma_N^A(x)$ ,  $\kappa(x, w) \leq 1$  and  $\alpha(x, w) = \#\text{acc}_{NA}(x) \leq 2^{q(|x|)} - 1$ . For other  $w$  of length  $m(|x|)$ ,  $\alpha(x, w) \leq 2^{q(|x|)}$  and

$$\kappa(x, w) \leq 2^{-\min\{r(|y_{x,1}|), \dots, r(|y_{x,m(|x|)}|)\}}.$$

Because  $m$  is bounded by  $p$ , because  $r$  is a natural polynomial, and because the machine  $N$  is a length-increasing query generator, this is at most

$2^{-p(|x|)-q(|x|)-1}$ . Since the number of  $w$ ,  $|w| = m(|x|)$ , such that  $w \neq \Gamma_N^A(x)$  is  $2^{m(|x|)} - 1 < 2^{p(|x|)}$ ,  $\frac{s(x)}{t(x)}$  is less than

$$\begin{aligned} & 2^{q(|x|)-1} - 1 + 2^{p(|x|)} 2^{-p(|x|)-q(|x|)-1} \\ &= 2^{q(|x|)-1} - 1 + 2^{-q(|x|)-1} \\ &\leq 2^{q(|x|)-1} - \frac{1}{2}. \end{aligned}$$

Thus the claim holds.

Now define

$$d(x) = 4s(x) - (2^{q(|x|)+1} - 1)t(x).$$

Then, for every  $x \in \Sigma^*$ ,  $x \in L$  if and only if  $d(x) \geq 0$ . We claim that  $d \in \text{GapL}$ . Proving this may seem easy at first glance, for we have already done something similar for GapP to prove Theorem 9.22. However, the machines here are logarithmic space-bounded and, for all  $x \in \Sigma^*$ ,  $s(x)$  is defined as a sum of  $2^{m(|x|)}$  terms that are indexed by  $w$  of length  $m(|x|)$ . So, for a nondeterministic machine to produce  $s$  as its gap function, it may seem necessary that the machine has space to store  $w$ . This is obviously impossible here, since the machines need to be logarithmic space-bounded. Hence we need a trick.

By Proposition 9.24 the second term of  $d$  is in GapL. We thus concentrate on proving that  $s \in \text{GapL}$ . Let  $G$  be a logarithmic space-bounded machine such that  $g = \# \text{gap}_G$ . For every  $x \in \Sigma^*$ ,  $s(x)$  can be written as

$$\sum_{|w|=m(|x|)} \alpha(x, w) \prod_{1 \leq i \leq m(|x|)} (\# \text{acc}_G(\langle x, w_i \rangle) - \# \text{rej}_G(\langle x, w_i \rangle)).$$

Define  $T$  to be the machine that, on input  $x \in \Sigma^*$ , simulates  $N$  on  $x$  as follows: For each  $i$ ,  $1 \leq i \leq p(|x|)$ , when  $N$  enters the query state for the  $i$ th time, instead of making the query, it guesses a bit  $w_i$ , simulates  $G$  on  $\langle y_{x,i}, w_i \rangle$ , and returns to the simulation of  $N$  with  $w_i$  as the oracle answer. During the entire simulation the machine  $N$  counts using a variable  $R$  the number of  $i$ ,  $1 \leq i \leq p(|x|)$ , such that  $G$  rejected. At the end of the simulation, if  $N$  has accepted then  $T$  accepts if and only if  $R$  is even; on the other hand, if  $N$  has rejected then  $T$  guesses one bit  $b$  and accepts if and only if  $b = 0$ .

Let  $x$  be fixed. For simplicity, let  $m$  denote  $m(|x|)$  and for each  $i$ ,  $1 \leq i \leq m$ , let  $y_i$  denote  $y_{x,i}$ . Each computation path  $\pi$  of  $T$  on  $x$  is divided into the components

$$w, \pi_0, \pi_1, \dots, \pi_m, b.$$

Here  $w$  corresponds to the bits  $w_1, \dots, w_m$ ,  $\pi_0$  corresponds to the nondeterministic moves of  $N$  on input  $x$  with  $w$  as the oracle answers,  $\pi_i$  corresponds to the nondeterministic moves of  $G$  on input  $\langle y_i, w_i \rangle$  for all  $i$ ,  $1 \leq i \leq m$ , and  $b$  corresponds to the guess  $b$  at the end in the case when  $N$ 's simulation is rejecting. Write  $E(\pi) = 1$  if  $N$  has accepted along the path  $\pi$  of  $T$  and  $E(\pi) = 0$  otherwise. For each  $i$ ,  $1 \leq i \leq m$ , write  $F_i(\pi) = 1$  if  $G$  on  $\langle y_i, w_i \rangle$



accepts along the path  $\pi$  of  $T$  and  $F_i(\pi) = -1$  otherwise, where the value of  $w_i$  is the  $i$ th bit of the  $w$ -component of  $\pi$ . Since the process of guessing the bit  $b$  and accepting if and only if  $b = 1$  has the effect of canceling the contribution to the gap of  $T$  of the paths passing through the process,  $\#gap_T(x)$  is the sum of

$$E(\pi) \left( \left( \sum_{||\{i | F_i(\pi) = -1\}|| \text{ is even}} 1 \right) - \left( \sum_{||\{i | F_i(\pi) = -1\}|| \text{ is odd}} 1 \right) \right),$$

where  $\pi$  ranges over all computation paths of  $T$  on input  $x$ . Since the product of terms chosen from  $\{+1, -1\}$  is  $+1$  if the number of  $-1$ 's appearing in the product is even and is  $-1$  otherwise, the second term in the above is equal to

$$\prod_{1 \leq i \leq m} F_i(\pi),$$

and so  $\#gap_T(x)$  equals

$$\sum_{\pi} E(\pi) \prod_{1 \leq i \leq m} F_i(\pi).$$

For each  $u \in \Sigma^m$ , let  $Q(u)$  denote the set of all computation paths of  $T$  on input  $x$  whose  $w$ -component is equal to  $u$ . Then  $\#gap_T(x)$  can be written as

$$\sum_{|w|=m} \sum_{\pi \in Q(w)} E(\pi) \prod_{1 \leq i \leq m} F_i(\pi).$$

Since the paths for  $N$  and those for the simulation of  $G$  are pairwise independent for each fixed  $w$ , this is the same as

$$\sum_{|w|=m} \alpha(x, w) \prod_{1 \leq i \leq m} \#gap_G(y_i, w_i).$$

Thus  $s(x) = \#gap_T(x)$ . It is easy to see that  $T$  is logarithmic space-bounded and polynomial time-bounded. Hence  $T$  witnesses that  $s \in \text{GapP}$ .  $\square$

## 9.4 OPEN ISSUE: Is PP Closed Under Polynomial-Time Turing Reductions?

Is PP closed under polynomial-time Turing reductions? The question is subtle. By Toda's Theorem (Theorem 4.12),  $\text{PH} \subseteq \text{P}^{\text{PP}}$ . Also,  $\oplus\text{P}$  is included in  $\text{P}^{\text{PP}}$ . If PP is closed under Turing reductions, then PP includes both PH and  $\oplus\text{P}$ . However, it is known that there exist oracles relative to which  $\text{PH} \not\subseteq \text{PP}$  and oracles relative to which  $\oplus\text{P} \not\subseteq \text{PP}$ . These results indicate that relativizable proof techniques, such as machine simulations, cannot settle the  $\text{PP} = \text{P}^{\text{PP}}$  question.

## 9.5 Bibliographic Notes

Gill [Gil77] and Simon [Sim75] independently introduced the class PP (Simon used different notation for the class, though). The definition of Gill is via probabilistic Turing machines, while Simon's definition employs nondeterministic Turing machines. They both observed that the class is closed under complementation (Proposition 9.5). They left open the question of whether the class is closed under union. Russo [Rus85] and Beigel, Hemachandra, and Wechsung [BHW91] made progress on the problem by showing the closure of the class under symmetric difference and under polynomial-time parity reductions, respectively. Beigel, Reingold, and Spielman [BRS95] affirmatively resolved the problem (Theorem 9.15).

The approximation scheme (Definition 9.12) that Beigel, Reingold, and Spielman used in the proof is based on a formula of Newman [New64]. Paturi and Saks [PS94] applied Newman's Formula to show approximations of threshold circuits by parity circuits.

In addition to the closure property under intersection, Beigel, Reingold, and Spielman showed a number of closure properties of PP, including closure under polynomial-time  $\mathcal{O}(\log n)$  Turing reductions, but they left open the question of whether the class is closed under  $\leq_{tt}^p$ -reductions. Fortnow and Reingold [FR96] gave an affirmative answer to the question by proving Theorem 9.18. They also proved Theorem 9.22. Later, using a different technique, Beigel and Fu [BF00] showed that PP and PL are closed under P-uniform  $\text{NC}^1$ -reductions and under logspace-uniform  $\text{NC}^1$ -reductions, respectively. Caussinus et al. [CMTV98] use Newman's formula to prove that the class probabilistic- $\text{NC}^1$  is closed under intersection.

Gupta [Gup95] and Fenner, Fortnow, and Kurtz [FFK94] were the first to formulate the concept of gap functions. Fenner, Fortnow, and Kurtz defined the class GapP, and Proposition 9.3 is from their paper.

The class PL is due to Gill [Gil77], but his definition does not require that the probabilistic machines be polynomial time-bounded. Jung [Jun85] proved that the two versions are identical. Allender and Ogiwara [AO96] show that this equality holds with respect to any oracle under the RST restriction. The RST restriction first appears in a paper by Ruzzo, Simon, and Tompa [RST84]. The collapse of the PL hierarchy (Theorem 9.28) is due to Ogiwara [Ogi98].

The class  $\text{C=P}$  is due to Simon [Sim75], who first proved (using different notation)  $\text{C=P} \subseteq \text{PP}$ . Wagner [Wag86] rediscovered this class, introduced the name  $\text{C=P}$ , and proved its closure under  $\leq_{ctt}^p$ -reductions. A paper by Beigel, Chang, and Ogiwara [BCO93] presents the folklore "squaring technique" (see Proposition 9.8). This paper also proves Theorem 9.10. Gundermann, Nasser, and Wechsung [GNW90] show that  $\text{C=P}$  is closed under polynomial-time positive truth-table reductions. Ogiwara [Ogi94b] shows that  $\text{C=P}$  and  $\text{coC=P}$  are both closed (downward) under polynomial-time positive Turing reductions. Ogiwara [Ogi95a] shows that its closure under  $\leq_{tt}^p$ -reductions, its

closure under P-uniform  $NC^1$ -reductions, and its closure under P-uniform  $AC^0$ -reductions are all equal.

Allender and Ogihara [AO96] define  $C=L$ , the logarithmic-space version of  $C=P$ , and prove a number of its closure properties. Allender, Beals, and Ogihara [ABO99] show that the  $C=L$  hierarchy collapses.



## A. A Rogues' Gallery of Complexity Classes

*The form is the meaning, and indeed the classic Greek mind, with an integrity of perception lost by later cultures which separated the two, firmly identified them.*

—Vincent Scully, *The Earth, the Temple, and the Gods* [Scu62]

To the computer scientist, structure is meaning. Seeking to understand nature's diverse problems with man's humble resources, we simplify our task by grouping similarly structured problems. The resulting complexity classes, such as P, NP, and PSPACE, are simply families of problems that can be solved with a certain underlying computational power. The range of interesting computational powers is broad—deterministic, nondeterministic, probabilistic, unique, table lookup, etc.—and a suitably rich palette has been developed to reflect these powers—P, NP, PP, UP, P/poly, etc. These classes can themselves be studied in terms of their internal structure and behavior. This chapter briefly reviews the definitions, meanings, and histories of the central complexity classes covered in this book.

The “selected facts and theorems” lists in the tables that follow when possible give references for their facts and theorems. However, in those cases where the facts are presented in this book, the citation in the right margin is merely to the chapter that presents the result. This should not be interpreted as in any way claiming that such results are due to this book. Rather, the Bibliographic Notes section of the appropriate chapter should be consulted to learn the history and source of the result.

Complexity theory is so broad and rich that in an appendix of this size it would be impossible to define or collect the field's most important theorems. Thus, the choice of theorems here is eclectic, with the goal simply of giving a resource pointing towards some of the results known for these classes. However, to making the theorem lists below more useful as starting points into the original literature, we have in some cases included theorems whose statements involve concepts or classes are not discussed or defined in this book.

## A.1 P: Determinism

$$\begin{aligned}
 P &= \bigcup_k \text{DTIME}[n^k] \\
 &= \{L \mid L \text{ is accepted by a polynomial-time deterministic} \\
 &\quad \text{Turing machine}\}.
 \end{aligned}$$

P, deterministic polynomial time, is the class that is widely thought to embody the power of reasonable computation. In the 1930s, Gödel, Church, Turing, and Post [Göd31, Chu36, Tur36, Chu41, Pos46, Dav58] asked what could be *effectively* solved by computing machines—that is, what problems are recursive? In fact, these founding figures went beyond that. In a rediscovered 1956 letter to von Neumann, Gödel focused not only on the importance of the number of steps a Turing machine may need to perform a certain task (deciding whether a formula has a proof of a given length), but also used two particular polynomial bounds (linear and quadratic) as examples of efficient computation, in contrast with exhaustive search (see [Har89, Sip92] for more on this letter). Von Neumann was dying at the time, and it does not seem that he or Gödel ever followed up on the issues that Gödel had raised.

Starting in the 1960s, computer scientists, unaware of Gödel's letter and its musings in this direction, began to ask which problems can be *efficiently* solved by computers. The theory of P and NP, and indeed complexity theory itself, sprang from this desire to understand the limits of feasible computation. The notion that polynomial time,  $\bigcup_k \text{DTIME}[n^k]$ , is the right class to represent feasible computation was suggested by Cobham and Edmonds [Cob64, Edm65] (who, again, were unaware of Gödel's letter). Note that polynomials grow slowly and are closed under composition (thus allowing subroutine calls in the sense that a polynomial-time machine making subroutine calls to polynomial-time subroutines yields an overall polynomial-time procedure). These features support the claim that P is a reasonable resource bound. The view that P loosely characterizes “feasibility” is widely accepted.

One might argue that an algorithm that runs for  $10^{10^{10}}$   $n^{10^{100}}$  steps on inputs of size  $n$  is not practical. Problems are known that provably require high-degree polynomial algorithms (artificial problems must exist via the deterministic time hierarchy theorem [HS65][HU79, Theorem 12.9], and somewhat artificial cat-and-mouse games and pebbling problems [KAI79, AIK84]), and natural problems are known that may require high-degree polynomial algorithms (permutation group membership from generators [Hof82, FHL80], robotics configuration space problems [SS83]).<sup>1</sup>

<sup>1</sup> Of course, many natural problems are known to have superpolynomial lower bounds. For example, Meyer and Stockmeyer [MS72] and Fischer and Rabin [FR74] show, respectively, problems that require exponential space and double exponential nondeterministic time. The problems listed here are natural, fundamental *polynomial-time* problems that may require high-degree polynomial algorithms.

---

## P – Polynomial Time

### Power

Feasible computation.

### Definition

$$P = \bigcup_k \text{DTIME}[n^k].$$

### Background

P was described as embodying the power of feasible computation by Cobham [Cob64] and Edmonds [Edm65]. The field of design and analysis of algorithms attempts to place as many problems as possible in P.

### Complete Languages

P has countless well-known complete languages under  $\leq_m^L$  reductions (see the list compiled by Greenlaw, Hoover, and Ruzzo [GHR95]). Typical P-complete problems include determining whether a given context-free grammar is empty [JL76] and determining whether a given output bit of a given circuit on a given input is on [Lad75a]. Kasai, Adachi, and Iwata [KAI79] have shown combinatorial games providing additional natural complete problems for P.

### Sample Problem

In a fixed, reasonable proof system, asking if  $x$  is a proof of  $T$  is a polynomial-time question. In particular, in polynomial time we can check whether assignment  $x$  satisfies boolean formula  $F$ .

### Selected Facts and Theorems

1. For each  $k$ , there are relatively natural problems, having to do with games of pursuit and evasion, whose deterministic time requirements are  $\Omega(n^k)$ . [AIK84]
2.  $P = L \iff P$  has sparse hard sets with respect to logspace many-one reductions (or even logspace bounded-truth-table reductions). ([CS99,vM96], see also [Ogi96b,CNS96])
3. All P sets are rankable (i.e., have a polynomial-time computable function that, given any string  $x$ , computes the number of strings in the set that are lexicographically less than or equal to  $x$ )  $\iff P = P^{\#P}$ . ([GS91], see also [HR90])
4. All infinite P sets are compressible (i.e., each P set  $A$  has a polynomial-time computable, one-to-one function  $f$  such that  $f(A) = \Sigma^*$ ) if  $E = NE^{NP}$ . ([GHK92], see also [GH96])
5. If every dense P set has at most a sparse subset of Kolmogorov-easy strings, then all polynomial-time pseudorandom generators are insecure. [All89c,HH96]

**Fig. A.1** P

---

Nonetheless, there is a widely held feeling that fundamental natural problems belonging to P will have polynomial-time algorithms of low degree. The field of design and analysis of algorithms attempts to prove that key problems are in P, and then to show that they have algorithms of low time complexity (there exist many books on the analysis of algorithms, e.g., [AHU74, CLRS01, Koz92]).

## A.2 NP: Nondeterminism

*Two roads diverged in a yellow wood,  
And sorry I could not travel both  
And be one traveler...  
—Robert Frost, The Road Not Taken*

$$\begin{aligned} \text{NP} &= \bigcup_k \text{NTIME}[n^k] \\ &= \{L \mid L \text{ is accepted by a polynomial-time nondeterministic} \\ &\quad \text{Turing machine}\}. \end{aligned}$$

P contains the problems we can solve. NP symbolizes the problems man needs to solve to efficiently structure and optimize his world. The  $P=NP$  question asks whether the computers built by man's ingenuity have the power to solve the problems formed by nature's complexity.

NP is the class of languages accepted by nondeterministic polynomial-time Turing machines [HU79]. Intuitively, a nondeterministic machine is one that is allowed to make guesses during its computation, and always guesses correctly. Equivalently, a language  $L$  is in NP if there exists a polynomial-time computable relation  $R(\cdot, \cdot)$  and a polynomial  $q$  such that

$$L = \{x \mid (\exists y : |y| \leq q(|x|)) [R(x, y)]\}.$$

In the early 1970s, the work of Cook and Karp [Coo71, Kar72] showed that NP has natural complete, or “hardest,” languages—languages to which every other NP problem can be polynomial-time many-one reduced. These problems stand or fall together: If one NP-complete problem is in P then all NP-complete problems are in P. During the past quarter century, hundreds of problems from all areas of mathematics, computer science, and operations research have been shown NP-complete. If  $P=NP$  then these and many crucial optimization problems can be solved in polynomial time. And, just as importantly, if  $P \neq NP$  then no NP-complete problem can be solved in polynomial time.

However, the implications of  $P = NP$  are even more profound. An NP machine can answer the question, in a fixed formal system, “Does this theorem



---

**NP – Nondeterministic Polynomial Time**
**Power**

Guessing. Nondeterminism.

**Definition**

$\text{NP} = \bigcup_k \text{NTIME}[n^k]$ .

**Alternate Definition**

A language  $L$  is in NP if there exists a polynomial  $q$  and a polynomial-time predicate  $R$  such that, for each  $x$ ,

$$x \in L \iff (\exists y : |y| \leq q(|x|)) [R(x, y)].$$

**Background**

In the early 1970s, Cook [Coo71] and Levin [Lev75], followed by a key paper by Karp [Kar72], initiated the study of NP and its complete problems. Many NP-complete problems are now known, and the study of NP's structure is a unifying theme of complexity theory.

**Complete Problems**

NP has hundreds of  $\leq_m^p$ -complete (polynomial-time many-one complete) problems [GJ79].

The most studied NP-complete problem is satisfiability.  $\text{SAT} = \{F \mid \text{boolean formula } F \text{ is satisfiable}\}$  was shown to be Turing-complete for NP by Cook. Karp showed that SAT and many other problems are  $\leq_m^p$ -complete for NP.

There are a few problems that are known to be in NP, yet have been neither proven to be NP-complete nor proven to be in P. Examples of such problems are graph isomorphism (i.e.,  $\{(G, H) \mid G \text{ and } H \text{ are isomorphic}\}$ ) and primality.

**Fig. A.2** NP—part I

---

have a proof (of reasonable size)?" Thus NP embodies the power of guessing, or creating, mathematical proofs. P embodies the mechanical process of verifying whether a proof is correct. Asking whether  $P \neq \text{NP}$  is another way of asking whether the creative process in mathematics rises above the complexity of mere mechanical verification. Since men are likely to create mathematical proof structures only of small size, asking whether  $P = \text{NP}$  is one way of asking whether machines can usurp man's role in mathematical discovery. Breakthroughs during the 1990s in the theory of probabilistically checkable proofs have given alternative new insights into the power of NP and the nonapproximability of NP optimization problems (see, for example, the treatments in [Aro94, Sud92, ACG<sup>+</sup>99]). NPNP is the most extensively studied computational complexity class, and many insights into NP's structure have been found during the past decade. Nonetheless, our understanding of NP is fragmented, incomplete, and unsatisfying.

---

**NP – Nondeterministic Polynomial Time**
**Selected Facts and Theorems**

1.  $P \subseteq NP$ .
2. The following are equivalent:
  - a)  $P = NP$ .
  - b) Some NP-complete problem is in  $P$ .
  - c) All NP-complete problems are in  $P$ .
3. **[Cook's Theorem]** Let  $N_i$  be a standard enumeration of NPTMs (nondeterministic polynomial-time Turing machines). There is a polynomial-time computable function  $f_{\text{COOK}}$ , mapping from machine-string pairs to boolean formulas, such that
  - a)  $(\forall i)(\forall x)[N_i(x) \text{ accepts} \iff f_{\text{COOK}}(N_i, x) \text{ is satisfiable}]$ ,
  - b)  $(\exists \text{ polynomial-time computable function } g_{\text{COOK}})(\forall i)(\forall x)[g_{\text{COOK}}(f_{\text{COOK}}(N_i, x)) = \langle N_i, x \rangle]$ , and
  - c)  $(\exists h_{\text{COOK}} \in \text{FP})(\forall i)(\forall x)(\forall a) \text{ [if } a \text{ is a satisfying assignment of } f_{\text{COOK}}(N_i, x), \text{ then } h_{\text{COOK}}(N_i, x, a) \text{ outputs an accepting computation path of } N_i(x)]$ .

[Coo71, Lev75]
- In particular, SAT is  $\leq_m^P$ -complete for NP.
4. NP is closed under union and intersection.
5. NP is closed downward under positive Turing reductions.
 

([Sel82b], see also [HJ91])
6.  $P \neq NP \implies NP - P$  contains sets that are not NP-complete. [Lad75b]
7. If NP has sparse  $\leq_m^P$ -hard sets, or even sparse  $\leq_{btt}^P$ -hard sets then  $P = NP$ .
 

(see Chap. 1)
8. If NP has sparse  $\leq_T^P$ -complete sets, then the polynomial hierarchy collapses to  $\Theta_2^P$ .
 

(see Chap. 1)
9. If NP has sparse  $\leq_{dtt}^P$ -hard sets, then  $RP = NP$  and  $P = UP$ .
 

(see the Bibliographic Notes of Chap. 1)
10. If NP has sparse  $\leq_T^P$ -hard sets, the polynomial hierarchy collapses to  $NP^{NP}$  (and even to  $ZPP^{NP}$ , and even to  $S_2^P$ ).
 

(see the text and Bibliographic Notes of Chap. 1)
11.  $NP - P$  contains sparse sets if and only if  $E \neq NE$ .
 

(see Chap. 1)
12. Many-one one-way functions exist if and only if  $P \neq NP$ .
 

(see Chap. 2)
13. Many-one one-way functions exist if and only if strongly noninvertible, total, commutative, associative, 2-ary, many-one one-way functions exist.
 

(see Chap. 2)
14.  $(\forall L \in NP)[L \leq_{\text{randomized}} \text{USAT}]$ .
 

(see Chap. 4)

**Fig. A.3** NP—part II

---

**A.3 Oracles and Relativized Worlds**

*All is for the best in the best of all possible worlds.*  
—Voltaire, Candide

The seminal paper on oracles was by Baker, Gill, and Solovay [BGS75]. Since then oracles have been discussed extensively in the literature (see, just as a few examples, the seminal paper by Bennett and Gill on random oracles [BG81], the insightful leaf-language/oracle connection work of Bovet,

---

**NP – Nondeterministic Polynomial Time**
**Selected Facts and Theorems (Continued)**

15.  $(\exists A)[P^A = NP^A]$ .  $(\exists B)[P^B \neq NP^B]$ . Indeed, with probability one relative to a random oracle, P and NP differ. [BGS75,BG81]
16. All paddable NP-complete sets are p-isomorphic to SAT. [BH77,MY85]
17. With probability one relative to a random oracle, there are NP-complete sets that are not P-isomorphic. [KMR95]
18. There is a relativized world in which all NP-complete sets are P-isomorphic. [FFK96]
19. If  $P = NP$  and  $S$  is sparse then

$$P^S = NP^S \iff (\exists k)[S \subseteq K^S[k \log n, n^k]],$$

- where  $K[\cdot, \cdot]$  represents time-bounded Kolmogorov complexity. [HH88b]
20. If the graph isomorphism problem is NP-complete, then the polynomial hierarchy collapses. [GS89,BHZ87,GMW91,Sch88]
  21. If  $P \neq NP \cap \text{coNP}$  then there is a set  $S$  so (1)  $S \in P$  and  $S \subseteq \text{SAT}$ , and (2) no P machine can find solutions for all formulas in  $S$ —that is, for any polynomial-time computable function  $g$ , there will be a formula  $f \in S$  such that  $g(f)$  is not a satisfying assignment of  $f$ . [BD76]
  22. For each  $k > 0$ ,  $R_{k-T}^p(\text{NP}) = R_{2^k-1-tt}^p(\text{NP})$ . [Bei91a]
  23.  $NP \cap \text{coNP}$  has  $\leq_m^p$ -complete sets if and only if  $NP \cap \text{coNP}$  has  $\leq_T^p$ -complete sets. [Gur83,HI85]
  24. SAT is iteratively enumerable, i.e., there is an honest, polynomial-time function  $f$  and a string  $x_0$  such that  $\text{SAT} = \{x_0, f(x_0), f(f(x_0)), \dots\}$ . [HHSY91]
  25.  $\Theta_2^p = \text{NC}^1(\text{NP})$  [Got95,Ogi95a]
  26.  $\text{NP} = \text{PCP}(\mathcal{O}(\log n), \mathcal{O}(1))$ , i.e., NP is the class of languages  $L$  for which there exists a probabilistic polynomial-time oracle protocol  $V$  that uses  $\mathcal{O}(\log n)$  coin tosses, makes  $\mathcal{O}(1)$  queries, and, for all  $x \in \Sigma^*$ , satisfies the following two conditions:
    - if  $x \in L$ , then there is an oracle  $A$  relative to which  $V$  on input  $x$  accepts with probability 1,
    - if  $x \notin L$ , then, for every oracle  $A$ ,  $V$  on input  $x$  relative to  $A$  accepts with probability less than  $\frac{1}{2}$ . [ALM<sup>+</sup>98]

**Fig. A.4** NP—part III

---

Crescenzi, and Silvestri [BCS95], and Vereshchagin [Ver94], and the open-questions paper by Hemaspaandra, Ramachandran, and Zimand [HRZ95]). We may think of an oracle  $B$  as a *unit-cost subroutine* for the set  $B$ . For example,  $P^B$  ( $NP^B$ ) is the class of languages computable by deterministic (nondeterministic) polynomial-time Turing machines given unit-cost subroutines (i.e., subroutines that return in one time unit) that test membership in  $B$ . We may think of such a subroutine as changing the ground rules of computation under which the machines operate.

We can also define what it means to relativize a complexity class not with a single set but with another complexity class:

$$\mathcal{C}^{\mathcal{D}} = \bigcup_{A \in \mathcal{D}} \mathcal{C}^A.$$

For example,  $\text{NP}^{\text{NP}} = \bigcup_{A \in \text{NP}} \text{NP}^A = \text{NP}^{\text{SAT}}$ . We may think of  $\mathcal{C}^{\mathcal{D}}$  as the class of languages recognized by  $\mathcal{C}$  machines given free access to the power of some member of  $\mathcal{D}$ .

Though the issue is controversial, many have argued that oracles are a useful tool in understanding *possibilities* for complexity classes (see, e.g., Allender's and Fortnow's eloquent discussions [All90,For94]). Certainly, if we show that some complexity result  $T$  holds in a relativized world (that is, with some oracle  $B$ ), we know that relativizable proof techniques cannot disprove  $T$ . This is because a relativizable disproof of  $T$  would disprove  $T$  in *all* relativized worlds, but we know that  $T$  is true in the world relativized by  $B$ .

Many crucial results in complexity theory can be relativized in conflicting ways. For example, there are oracles  $A$  and  $B$  so that  $\text{P}^A = \text{NP}^A$  yet  $\text{P}^B \neq \text{NP}^B$  [BGS75]. Since most known mathematical proof techniques seem to relativize, such techniques cannot resolve such central questions as  $\text{P} = \text{NP}$ . However, starting around 1990 (but see Hartmanis et al. [HCC<sup>+</sup>92] for a discussion suggesting that nonrelativizable techniques have a much longer history than is commonly realized), the field has witnessed the emergence of some quite nontrivial nonrelativizable proof techniques (for example, those of [LFKN92,Sha92]). Chap. 6 is devoted to the discussion of a key technique of this sort. The breadth of the applicability of these techniques to complexity-theoretic issues is an active research topic [Har85,All90,HCRR90,HCC<sup>+</sup>92,For94].

Though oracles exist to certify many unlikely situations—e.g., there is an oracle  $A$  for which  $\text{P}^A = \text{NP}^A = \text{PSPACE}^A$ , we should not think of oracles as telling us what is the case in the world of computation. Rather, we should think of oracles as suggesting the limitations of relativizable proof techniques.

## A.4 The Polynomial Hierarchy and Polynomial Space: The Power of Quantifiers

### A.4.1 The Polynomial Hierarchy

*A deck of cards was built like the purest of hierarchies, with every card a master to those below it and a lackey to those above it.*  
—Ely Culbertson, Total Peace

The polynomial hierarchy was defined by Meyer and Stockmeyer [MS72, Sto76] as a time-bounded analogue of the Kleene hierarchy (also known as

---

**PH, PSPACE, P, NP, coNP, P<sup>NP</sup>, NP<sup>NP</sup>, ... – The Polynomial Hierarchy and Polynomial Space**


---

**Power**

Alternating polynomially bounded existential and universal quantifiers.

**Definition**

$$\begin{aligned}
 \Sigma_0^P &= \Pi_0^P = P. \\
 \Theta_{i+1}^P &= R_{O(\log n)-T}^P(\Sigma_i^P), \quad i \geq 0. \\
 \Delta_{i+1}^P &= P^{\Sigma_i^P}, \quad i \geq 0. \\
 \Sigma_{i+1}^P &= NP^{\Sigma_i^P}, \quad i \geq 0. \\
 \Pi_{i+1}^P &= \text{co}\Sigma_{i+1}^P = \{L \mid \bar{L} \in \Sigma_{i+1}^P\}, \quad i \geq 0. \\
 \text{PH} &= \bigcup_i \Sigma_i^P. \\
 \text{PSPACE} &= \bigcup_k \text{DSPACE}[n^k].
 \end{aligned}$$

**Alternate Definition**

$L$  is in  $\Sigma_i^P$  if there is a polynomial  $q$  and a polynomial-time predicate  $R$  such that, for all  $x$  it holds that

$$\begin{aligned}
 x \in L \iff & (\exists w_1 : |w_1| \leq q(|x|)) (\forall w_2 : |w_2| \leq q(|x|)) \cdots \\
 & (Q_i w_i : |w_i| \leq q(|x|)) [R(x, w_1, \dots, w_i)],
 \end{aligned}$$

where  $Q_i$  is  $\exists$  if  $i$  is odd and  $\forall$  if  $i$  is even.

**Background**

The polynomial hierarchy was defined by Meyer and Stockmeyer [MS72, Sto76]. Researchers later introduced refined, intermediate levels, namely, the  $\Theta_k^P$  levels (see [PZ83, Wag90]).

**Fig. A.5** The polynomial hierarchy and PSPACE—part I

---

the arithmetical hierarchy) from recursive function theory [Rog67]. The definitions of the polynomial hierarchy appear in Fig. A.5. In particular,  $\Sigma_0^P = P$ ,  $\Sigma_1^P = \text{NP}$ ,  $\Pi_1^P = \text{coNP}$ ,  $\Theta_2^P = R_{O(\log n)-T}^P(\text{NP})$ ,  $\Delta_2^P = \text{P}^{\text{NP}}$ , and  $\Sigma_2^P = \text{NP}^{\text{NP}}$ .

The levels of the polynomial hierarchy have natural descriptions in terms both of Turing machines and logical formulas. Just as the Kleene hierarchy's levels are characterized by quantifier alternation, so also are the levels of the polynomial hierarchy characterized by alternating polynomially bounded quantifiers [Sto76, Wra76]. For example,

$$\begin{aligned}
 \text{NP} &= \{L \mid (\exists k) (\exists \text{ polynomial-time predicate } P) \\
 &\quad [x \in L \iff (\exists y : |y| \leq |x|^k) [P(x, y)]]\}, \text{ and} \\
 \Pi_2^P &= \{L \mid (\exists k) (\exists \text{ polynomial-time predicate } P) \\
 &\quad [x \in L \iff (\forall y : |y| \leq |x|^k) (\exists z : |z| \leq |x|^k) [P(x, y, z)]]\}.
 \end{aligned}$$

---

**PH, PSPACE, P, NP, coNP, P<sup>NP</sup>, NP<sup>NP</sup>, ... – The Polynomial Hierarchy and Polynomial Space**
**Complete Languages**

Canonical complete languages exist for each level of the hierarchy ([Wra76], and the techniques of [Har78]) and for PSPACE [Sto76].

NP has many well-known natural complete problems (see [GJ79]). PSPACE also has many natural complete problems. For example, Fraenkel et al. [FGJ<sup>+</sup>78] showed that generalized checkers is PSPACE-complete, and Iwata and Kasai [IK94] showed that generalized Othello is PSPACE-complete.

$\Sigma_2^P$  also has some interesting, natural complete problems. Some date as far back as the 1970's, when the early work of Meyer and Stockmeyer showed that Integer Expression Inequivalence is  $\Sigma_2^P$ -complete. Other  $\Sigma_2^P$ -completeness results include Huynh's [Huy84] work on inequivalence of certain types of context-free grammars, Schaefer's [Sch01b] work on Ramsey-type problems, and Umans's [Uma98] work on boolean formula minimization problems.  $\Sigma_3^P$  has natural complete problems, for example, Schaefer's [Sch99, Sch00] work on the VC-dimension. Schaefer has written a nice compendium of natural complete problems for  $\Sigma_2^P$ ,  $\Pi_2^P$ ,  $\Sigma_3^P$ ,  $\Pi_3^P$ , etc. [Sch01a].

Papadimitriou [Pap84] showed that natural problems are complete for  $\Delta_2^P$ , including Unique Optimal Traveling Salesperson. Hemaspaandra, Hemaspaandra, and Rothe [HHR97] showed that it is  $\Theta_2^P$ -complete. to check who the winner is in the election system developed in 1876 by Lewis Carroll.

Problems asking when greedy algorithms perform well are also known to be  $\Theta_2^P$ -complete [HR98]. Wagner [Wag87] provided a valuable framework for proving  $\Theta_2^P$ -completeness results.

---

**Fig. A.6** The polynomial hierarchy and PSPACE—part II
 

---

This characterization by alternating quantifiers is handy. When asked the complexity of  $\text{MINIMAL-FORMULAS} = \{F \mid F \text{ is a boolean formula and no equivalent boolean formula is shorter than } F\}$ , we can reflect for a moment on the underlying quantifier structure and quickly note that  $\text{MINIMAL-FORMULAS} \in \Pi_2^P$ . That is, MINIMAL-FORMULAS is the set of all  $F$  such that *for every* shorter formula  $F'$  *there exists* a variable assignment on which  $F$  and  $F'$  differ.

The work of Chandra, Kozen, and Stockmeyer [CKS81] develops machines that accept the languages at each level of the polynomial hierarchy. Known as alternating Turing machines, the action of these machines alternates between existential and universal blocks, and mirrors the underlying quantifier structure of the classes.

We say that the polynomial hierarchy *collapses* if, for some  $k$ ,  $\Sigma_k^P = \Pi_k^P$  (thus  $\Sigma_k^P = \text{PH}$ ). A crucial open question is, does the polynomial hierarchy collapse? That is, is some fixed number of quantifiers powerful enough to simulate all fixed arrangements of quantifiers? Oracles are known for which the hierarchy collapses [BGS75] and for which the hierarchy does not col-

---

**PH, PSPACE, P, NP, coNP, P<sup>NP</sup>, NP<sup>NP</sup>, ... – The Polynomial Hierarchy and Polynomial Space**


---

**Selected Facts and Theorems**

1. For  $k \geq 0$ ,  $\Sigma_k^P \cup \Pi_k^P \subseteq \Theta_{k+1}^P \subseteq \Delta_{k+1}^P \subseteq \Sigma_{k+1}^P \cap \Pi_{k+1}^P$ .
2.  $\text{PH} \subseteq \text{PSPACE}$ .
3.  $\text{PH} = \text{PSPACE} \implies \text{PH}$  collapses.
4. For any  $k \geq 1$ ,  $\Sigma_k^P = \Pi_k^P \implies \Sigma_k^P = \text{PH}$  (Downward Separation). [Sto76]
5. For any  $k \geq 0$ ,  $\Sigma_k^P = \Sigma_{k+1}^P \implies \Sigma_k^P = \text{PH}$  (Downward Separation). [Sto76]
6.  $\text{P}^{\text{NP}} \cap \text{coNP} = \text{NP} \cap \text{coNP}$ .  $\text{NP}^{\text{NP}} \cap \text{coNP} = \text{NP}$ .  
([Sel79,Sch83], see also [Sel74,Lon78,Sel78])
7.  $\Theta_2^P = \text{R}_{tt}^P(\text{NP})$ . [Hem89]
8.  $(\exists A)[P^A = \text{PSPACE}^A]$ . [BGS75]
9.  $\oplus \text{P}^{\text{PH}} \subseteq \text{BPP}^{\oplus \text{P}} \subseteq \text{PP}^{\oplus \text{P}} \subseteq \text{P}^{\# \text{P}[1]} \subseteq \text{P}^{\# \text{P}}$ . (see Chap. 4)
10.  $\text{PP}^{\text{PH}} \subseteq \text{P}^{\# \text{P}}$ . (see Chap. 4)
11.  $\text{IP} = \text{PSPACE}$ . (see Chap. 6)
12.  $(\exists A)[P^A \neq \text{NP}^A \neq \text{NP}^{\text{NP}^A} \neq \dots \neq \text{PSPACE}^A]$ . (see Chap. 8)
13. The following are equivalent:
  - a)  $\text{PH}$  collapses.
  - b) There is a sparse set  $S$  such that  $\text{PH}^S$  collapses.
  - c) For all sparse sets  $S$ ,  $\text{PH}^S$  collapses. ([BBS86], see also [LS86])
14. The following are equivalent:
  - a)  $\text{PH} = \text{PSPACE}$ .
  - b) There is a sparse set  $S$  such that  $\text{PH}^S = \text{PSPACE}^S$ .
  - c) For all sparse sets  $S$ ,  $\text{PH}^S = \text{PSPACE}^S$ . ([BBS86], see also [LS86])
15.  $\text{Prob}_A(\text{PH}^A \subsetneq \text{PSPACE}^A) = 1$ . ([Cai89], see also [Bab87])
16.  $\text{PSPACE} = \text{NPSpace} = \text{Probabilistic-PSPACE}$ . [Sav70,Sim77b]
17. For each  $j$  there is an oracle  $A$  such that  $(\Sigma_j^P)^A \neq (\Sigma_{j+1}^P)^A = \text{PSPACE}^A$ . [Ko89]
18. There is an oracle relative to which, for all  $j \geq 2$ ,  $\Theta_j^P \subsetneq \Delta_j^P \subsetneq \Sigma_j^P$ .  
(see the text and Bibliographic Notes of Chap. 8)
19.  $\text{P}^{\text{NP}^{\text{SPARSE}}} \cap \text{NP} = \text{P}^{\text{NP}}$ . [KS85]
20. If  $S$  is sparse set, then  $\text{P}^{\text{NP}^{\text{NP}^S}} = \text{P}^{\text{NP}^S \oplus \text{SAT}}$ . [Sch86a]
21. If the polynomial hierarchy collapses with probability one relative to a random oracle, then the polynomial hierarchy collapses. [Boo94]
22.  $\text{E} \neq \text{PSPACE}$ . [Boo74a]
23. For each  $k > 1$  it holds that:  $\text{R}_{1-T}^P(\Sigma_k^P) = \text{R}_{2-T}^P(\Sigma_k^P) \implies \Sigma_k^P \cap \Pi_k^P = \text{PH}$ .  
([BF99], see also [HHH99a])

**Open Problems**

- Does the polynomial hierarchy collapse?
- $\text{Prob}_A(P^A \neq \text{NP}^A \neq \text{NP}^{\text{NP}^A} \neq \dots) = 1$ ?
- $\text{Prob}_A(P^A \neq \text{NP}^A \cap \text{coNP}^A) = 1$ ?

**Fig. A.7** The polynomial hierarchy and PSPACE—part III

lapse (Chap. 8). An exponential analog of the polynomial hierarchy collapses [Hem89].

#### A.4.2 Polynomial Space

*To me every hour of the light and dark is a miracle,  
Every cubic inch of space is a miracle.  
—Walt Whitman, Miracles*

PSPACE is the class of languages accepted by polynomial-space Turing machines. PSPACE embodies the power of polynomially bounded quantifiers. A quantified boolean formula is an expression of the form

$$(\exists x_1)(\forall x_2)(\exists x_3) \cdots [f(x_1, x_2, x_3, \dots)],$$

where  $f$  is a quantifier-free boolean formula and the  $x_i$  are boolean variables. QBF, the set of true quantified boolean formulas, is a well-known PSPACE-complete problem, and shows how PSPACE embodies the power of alternating quantifiers [Sto76].

There are many PSPACE-complete problems. Adversary (game) problems are often PSPACE-complete. For example, the generalized versions of GO [LS80] and Othello [IK94] are PSPACE-complete. In a fixed formal system, whether a theorem has a polynomial “proof presentation”—basically, whether given an eraser and a polynomial-sized blackboard one can convince an uncreative, deterministic actor of the truth of the theorem—can be determined in PSPACE [HY84].

#### A.5 E, NE, EXP, and NEXP

$E = \bigcup_{c>0} \text{DTIME}[2^{cn}]$  and  $NE = \bigcup_{c>0} \text{NTIME}[2^{cn}]$  are exponential-time analogs of P and NP. The structure of these exponential-time classes is linked to the structure of polynomial-time classes. In particular, the complexity of tally<sup>2</sup> and sparse sets within NP is tied to the structure of E and NE [Boo74b, HH74, HY84, HIS85, CGH<sup>+</sup>88, CGH<sup>+</sup>89].

$\text{EXP} = \bigcup_{c>0} \text{DTIME}[2^{n^c}]$  and  $\text{NEXP} = \bigcup_{c>0} \text{NTIME}[2^{n^c}]$  are alternate exponential-time analogs of P and NP. They are particularly useful in classifying the complexity of logics. For example, the satisfiability problem of propositional dynamic logic is EXP-complete [Pra79, FL79], as are the satisfiability problems of various attribute-value description formalisms [BS93] and various branching time logics [EH85]. Various logic problems are also known that are complete for NEXP (see [Pap94, Chap. 20]). NEXP has also proven central in understanding the complexity of interactive proof systems (see Chap. 6).

<sup>2</sup>  $T$  is a tally set if  $T \subseteq 1^* = \{\epsilon, 1, 11, 111, \dots\}$ .



---

**E, NE, EXP, and NEXP – Exponential-Time Classes**
**Power**

Exponential-time deterministic and nondeterministic computation.

**Definitions**

$$\begin{aligned}
 E &= \bigcup_c \text{DTIME}[2^{cn}]. \\
 NE &= \bigcup_c \text{NTIME}[2^{cn}]. \\
 EXP &= \bigcup_c \text{DTIME}[2^{n^c}]. \\
 NEXP &= \bigcup_c \text{NTIME}[2^{n^c}].
 \end{aligned}$$

**Background**

The complexity of sparse sets in the polynomial hierarchy is closely related to the structure of exponential-time classes [Boo74b, HH74, HY84, HIS85, CGH<sup>+</sup>89].

**Complete Languages**

All these classes have straightforward canonical complete languages that capture the actions of generic machines (see the techniques of [Har78]). The satisfiability problem of propositional dynamic logic is EXP-complete. Various problems from logic (e.g., whether a given Schönfinkel-Bernays expression has a model) known to be complete for NEXP (see [Pap94, Chap. 20]).

**Selected Facts and Theorems**

1.  $E \subseteq NE$ .  $PSPACE \subseteq EXP \subseteq NEXP$ . [Boo72]
2.  $E \neq NP$ . [Boo72]
3.  $E \subsetneq EXP$ .  $NP \subsetneq NE \subsetneq NEXP$ . [HS65, Coo73, SFM78]
4. The strong exponential hierarchy collapses, i.e.,  $P^{NE} = E \cup NE \cup NP^{NE} \cup NP^{NP^{NE}} \cup \dots$ . [Hem89]
5.  $MIP = NEXP$ . (see Chap. 6)
6.  $E = NE$  if and only if there are no tally sets in  $NP - P$ . [Boo74b, HH74]
7.  $E = NE$  if and only if there are no sparse sets in  $NP - P$ . [HIS85]
8.  $NE = coNE$  if and only if every sparse set in  $NP$  is NP-printable. [HY84]
9. All  $\leq_m^P$ -complete sets for EXP have infinite P subsets. [Ber76]
10.  $E \neq PSPACE$ . [Boo74a]
11. All  $\leq_{1-tt}^P$ -complete sets for E are  $\leq_m^P$ -complete for E. [HKR93]
12. All  $\leq_{1-tt}^P$ -complete sets for NE are  $\leq_m^P$ -complete for NE. [BST93]
13. If  $EXP \subseteq P/poly$ , then  $EXP = MA$ .

(see the Bibliographic Notes of Chap. 6)

**Fig. A.8** E, NE, EXP, and NEXP

---

---

**P/poly – Nonuniform Polynomial Time**
**Power**

Small circuits. Table lookup.

**Definition**

P/poly denotes  $\{L \mid (\exists C \in \mathbf{P}) (\exists \text{ polynomial } f) [L \in C/f]\}$ , where  $C/f$  denotes the class of all sets such that for some function  $h$  satisfying  $(\forall n) [|h(n)| = f(n)]$  it holds that  $L = \{x \mid \langle x, h(|x|) \rangle \in C\}$ .

**Selected Facts and Theorems**

1.  $\mathbf{P/poly} = \{L \mid (\exists \text{ sparse } S) [L \in \mathbf{P}^S]\}$ . (see [BH77])
2.  $\mathbf{P/poly} = \{L \mid (\exists C \in \mathbf{P}) (\exists \text{ polynomial } f) [L \in C/*f]\}$ , where  $C/*f$  denotes the class of all sets such that for some function  $h$  satisfying  $(\forall n) [|h(n)| \leq f(n)]$  it holds that  $L = \{x \mid \langle x, h(|x|) \rangle \in C\}$ .
3.  $\mathbf{P^{P/poly}/poly} = \mathbf{P/poly}$ .
4.  $\mathbf{NP} \subseteq \mathbf{P/poly} \Rightarrow \mathbf{PH} = \mathbf{ZPP^{NP}}$ . [KW98]
5.  $\mathbf{BPP^{\oplus P}} \subseteq \mathbf{P^{\#P[1]}}$ . (see Chap. 4)
6.  $\mathbf{BPP} \subseteq \mathbf{P/poly}$ . (see Chap. 4)
7.  $\mathbf{P\text{-}sel} \subseteq \mathbf{P/poly}$  (indeed, even  $\mathbf{P\text{-}sel} \subseteq \mathbf{P/quadratic}$ ). (see Chap. 3)
8. If  $A \in \mathbf{P/poly}$ , then  $\mathbf{P^{NP^{NP^A}}} \subseteq \mathbf{P^{NP^A \oplus SAT}}$ . In particular,  $\mathbf{P^{NP^{NP^{P/poly}} \cap NP}} = \mathbf{P^{NP^{NP}}}$ . The analogous inclusions hold for  $(\mathbf{NP} \cap \mathbf{coNP})/\mathbf{poly}$ . ([Köb94], see also [Gav95, Köb95])
9. If  $A \in \mathbf{NP/poly} \cap \mathbf{coNP/poly}$ , then  $\mathbf{NP^{NP^{NP^A}}} \subseteq \mathbf{NP^{NP^A \oplus SAT}}$ . In particular,  $\mathbf{NP^{NP^{NP^{(NP \cap coNP)/poly}} \cap NP}} = \mathbf{NP^{NP^{NP}}}$ . [HNOS96b]
10.  $\mathbf{P/poly} \neq \mathbf{E_T^P(SPARSE)}$ . [GW93]

**Fig. A.9** P/Poly

---

**A.6 P/Poly: Small Circuits**

$L$  is in P/poly if and only if  $L$  has small circuits, i.e., there is a family of “representations” (see [Sav72, Sch86b]) of boolean circuits  $C_1, C_2, \dots$  and an integer  $k$  such that:

- $|C_i| \leq i^k + k$ , and
- $x \in L \iff C_{|x|}$  accepts  $x$  [KL80].

More typically, and more generally, this is formalized as follows.

**Definition A.1** [KL80]

1. For any set  $A$  and any function  $f$ ,  $A/f$  denotes the class of all sets  $L$  such that for some function  $h$  satisfying  $(\forall n) [|h(n)| = f(n)]$  it holds that

$$L = \{x \mid \langle x, h(|x|) \rangle \in A\}.$$

2. For any class  $C$  and any function  $f$ ,  $C/f$  denotes

$$\{L \mid (\exists C \in C) [L \in C/f]\}.$$

3. For any set  $A$  and any class of functions  $\mathcal{F}$ ,  $A/\mathcal{F}$  denotes

$$\{L \mid (\exists f \in \mathcal{F}) [L \in A/f]\}.$$

4. For any class  $\mathcal{C}$  and any class of functions  $\mathcal{F}$ ,  $\mathcal{C}/\mathcal{F}$  denotes

$$\{L \mid (\exists C \in \mathcal{C}) (\exists f \in \mathcal{F}) [L \in C/f]\}.$$

Equivalently, a language  $L$  is in P/poly if and only if there is a sparse<sup>3</sup> set  $S$  so that  $L \in P^S$  (this equivalence is due to Meyer, see [BH77, p. 307] and [KL80]).

Intuitively, sets in P/poly are “close” to being in polynomial time. With a small amount of advice (e.g., the circuit description), a polynomial machine can recognize these sets. However, the advice may be terribly hard to compute; thus it is not surprising that P/poly contains sets arbitrarily high in the Kleene hierarchy.

Some important natural sets that are not known to be in P are known to have small circuits. For example, the set of primes is not known to be in P, but has small circuits and belongs to the class ZPP (which itself implies possession of small circuits) [Rab76, Adl78, APR83, GK99]. More generally, any set in the probabilistic class BPP has small circuits.

Karp and Lipton show it unlikely that all NP sets have small circuits: If NP has small circuits (i.e., if  $NP \subseteq P^S$  for some sparse set  $S$ ) then the polynomial hierarchy collapses to its second level. In the wake of their result, a flurry of related research has extended our knowledge of the implications of “ $NP \subseteq P^S$ ,  $S$  sparse,” and of “ $NP \subseteq P^S$ ,  $S$  sparse,  $S \in NP$ ” (see the surveys [HOW92, You92] or the papers [AHH<sup>+</sup>93, KW98]). This line of research is discussed in Chap. 1.

## A.7 L, NL, etc.: Logspace Classes

Much of the polynomial-time world (of P, NP,  $\oplus P$ , etc.) is echoed in the world of logspace computation. L and NL denote the languages acceptable by Turing machines running in, respectively, deterministic and nondeterministic logspace.  $\oplus L$  is the logspace analog of  $\oplus P$ . The study of logspace analogs of modulo classes was initiated by Buntrock et al. ([BDHM92], see also [HRV00]).

The logspace world provides only a partial analogy to the polynomial-time world. For example,  $NP = \text{coNP}$  is a major open question. Nonetheless, the beautiful logspace analog of this,  $NL = \text{coNL}$ , is known to hold, due to work of Immerman [Imm88] and Szelepcsényi [Sze88].

<sup>3</sup> A set  $S$  is sparse if there are at most polynomially many elements of length at most  $n$  in  $S$ , i.e.,  $(\exists k)(\forall n \geq 1)[|\{x \mid x \in S \wedge |x| \leq n\}| \leq n^k]$ .

---

**L, NL, etc.: Logspace Classes**
**Power**

Various types of logspace-bounded computation.

**Definition**

$L$  and  $NL$  denote the sets acceptable by, respectively, deterministic and non-deterministic logspace computation.

$$\begin{aligned}
 UL &= \left\{ L \mid \begin{array}{l} \text{there is a nondeterministic logspace Turing machine } N \\ \text{such that } L = L(N) \text{ and, for all } x, \text{ the computation tree} \\ \text{of } N \text{ on input } x \text{ has at most one accepting path} \end{array} \right\}. \\
 C=L &= \left\{ L \mid \begin{array}{l} \text{there is a nondeterministic logspace Turing machine } N \text{ and} \\ \text{a (deterministic) logspace-computable function } f \text{ so that} \\ \text{for each } x \text{ it holds that } x \in L \text{ if and only if } \#acc_N(x) = \\ f(x) \end{array} \right\}. \\
 PL &= \left\{ L \mid \begin{array}{l} \text{there is a probabilistic logspace-bounded Turing machine} \\ N \text{ so that for each } x \text{ it holds that } x \in L \text{ if and only if } \\ \Pr[N \text{ on input } x \text{ accepts}] \geq 1/2 \end{array} \right\}.
 \end{aligned}$$

**Background**

$PL$  was introduced by Gill [Gil77]. Gill defined  $PL$  as the class of all languages  $L$  for which there exists a probabilistic logarithmic space-bounded machine  $M$  with unlimited computation time such that, for all  $x$ ,  $x \in L$  if and only if the probability that  $M$  on input  $x$  accepts is at least  $\frac{1}{2}$ . Jung [Jun85] proves that a definition in which the machines are required to additionally run in polynomial time in fact gives the same class.  $C=L$  was first studied by Allender and Ogiwara [AO96].  $UL$  was first studied by Buntrock et al. [BJLR91].

**Complete Languages**

It is well known that  $PL$  and  $C=L$  have canonical complete languages. The language  $\{(G, s, t, m) \mid G \text{ is a topologically sorted directed graph } \wedge s, t \text{ are nodes in } G \wedge m \text{ is an integer } \wedge \text{ the number of paths in } G \text{ from } s \text{ to } t \text{ is at least } m\}$  is logspace many-one complete for  $PL$ . With “equal to” in place of “at least,” this language becomes logspace many-one complete for  $C=L$ . Jung [Jun85] presents a  $PL$ -complete problem that is related to the evaluation of polynomials over integer matrices. Allender and Ogiwara [AO96] show the problem of testing singularity of a given integer matrix is complete for  $C=L$ . It is unknown whether  $UL$  has a complete language.

**Selected Facts and Theorems**

1.  $NL = coNL$ . [Imm88, Sze88]
2.  $NL/poly \subseteq UL/poly$  (and so  $NL/poly = UL/poly$ ). (see Chap. 4)
3. The class of languages accepted by probabilistic logspace machines that are required to run in polynomial time exactly equals  $PL$ . [Jun85]
4.  $PL = PL^{PL}$ . (see Chap. 9)
5.  $L^{C=L} = C=L^{C=L}$ . [ABO99]
6.  $RL \subseteq SC^2$ . [Nis94]
7. All sets that are complete for  $NL$  with respect to 1-L (one-way-logspace) reductions are polynomial-time isomorphic. (Analogous results hold for  $NP$  and many other classes.) [All88]

**Fig. A.10**  $L$ ,  $NL$ , and other logspace classes

---

The class PL was defined by Gill [Gil77] as a logarithmic space, unlimited-computation-time version of PP. Jung [Jun85] showed that the polynomial-time version of PL is identical to the unlimited-time version. Various models of relativized nondeterministic/probabilistic logspace computation have been studied in the literature [LL76, Sim77a, RS81, RST84]. A widely used model is the Ruzzo–Simon–Tompia model [RST84], in which the logspace oracle machines are required to behave deterministically whenever query strings are generated. Allender and Ogihara [AO96] showed that Jung’s result relativizes under the Ruzzo–Simon–Tompia model. They also considered the logspace analog of the counting hierarchy.

The oracle hierarchies of PL and  $C=L$  are known to collapse [Ogi98, ABO99]. Damm [Dam91], Toda [Tod91a], Valiant [Val92], and Vinay [Vin91] independently observed that the determinant function is complete for  $\#L$ . Since the determinant function is in  $NC^2$  [BCP83], this implies that the various logspace classes are in  $NC^2$ . Nisan [Nis94] show that randomized logspace, RL, is contained in  $SC^2$ , the class of languages accepted by polynomial-time machines that use  $\mathcal{O}(\log^2 n)$  space. Allender, Beals, and Ogihara [ABO99], Santha and Tan [ST98], and Hoang and Thierauf [HT00] present algebraic problems that are complete for reducibility closures of  $C=L$ .

## A.8 NC, AC, LOGCFL: Circuit Classes

LOGCFL is the logspace many-one reducibility closure of the context-free languages.

A boolean circuit  $C_n$  with  $n$  inputs is a labeled, directed acyclic graph with nodes having in-degree zero or at least two. Nodes with in-degree zero are labeled from the set  $\{0, 1, x_1, \dots, x_n, \bar{x}_1, \dots, \bar{x}_n\}$  and all other nodes are labeled by either  $\wedge$  or  $\vee$  and compute  $\wedge$  or  $\vee$ , respectively. A language  $L$  is accepted by a family  $\mathcal{F} = \{C_n\}_{n \geq 1}$  of boolean circuits if, for every  $x$ ,  $x \in L$  if and only if  $C_{|x|}$  on  $x$  evaluates to 1. A family  $\{C_n\}_{n \geq 1}$  is logspace-uniform (P-uniform) if there exists a logarithmic space-bounded (polynomial time-bounded) Turing machine that computes the description of  $C_n$  given  $1^n$ .

For  $k \geq 1$ ,  $NC^k$  [Pip79] is the class of languages accepted by logspace-uniform,  $\mathcal{O}(\log^k n)$ -depth, polynomial-size, bounded fan-in (all  $\wedge$  and  $\vee$  gates have in-degree two) circuit families.  $NC = \bigcup_{k \geq 1} NC^k$ . For  $k \geq 0$ ,  $AC^k$  [Coo85, CSV84] is the class of languages accepted by logspace-uniform,  $\mathcal{O}(\log^k n)$ -depth, polynomial-size, unbounded fan-in (no restriction on the fan-in) circuit families.  $AC = \bigcup_{k \geq 0} AC^k$ . Moreover,  $SAC^k$  [BCD<sup>+</sup>89] is the class of languages accepted by logspace-uniform,  $\mathcal{O}(\log^k n)$ -depth, polynomial-size, semi-unbounded fan-in (all  $\wedge$  have in-degree two) circuit families.

**NC, AC, and LOGCFL — Polynomial-Size, Polylog-Depth Circuits**

Power

Boolean operations.

fan-in	$\wedge$	$\vee$
bounded	2	2
semi-unbounded	2	unbounded
unbounded	unbounded	unbounded

Definition

$$\begin{aligned} \text{LOGCFL} &= \left\{ L \mid \begin{array}{l} L \text{ is logspace many-one reducible to a context-free} \\ \text{language} \end{array} \right\}. \\ \text{NC}^k &= \left\{ L \mid \begin{array}{l} L \text{ is accepted by a logspace-uniform family of} \\ \text{polynomial-size, } \mathcal{O}(\log^k n)\text{-depth, bounded fan-in} \\ \text{circuit} \end{array} \right\}. \\ \text{AC}^k &= \left\{ L \mid \begin{array}{l} L \text{ is accepted by a logspace-uniform family of} \\ \text{polynomial-size, } \mathcal{O}(\log^k n)\text{-depth, unbounded fan-in} \\ \text{circuit} \end{array} \right\}. \end{aligned}$$

Alternate Definition

$$\begin{aligned} \text{NC}^k &= \left\{ L \mid \begin{array}{l} L \text{ is accepted by a deterministic polynomial-time} \\ \text{Turing machine with } \mathcal{O}(c \log^k n) \text{ reversal} \end{array} \right\}. \\ \text{AC}^k &= \left\{ L \mid \begin{array}{l} L \text{ is accepted by a logspace bounded alternating Tur-} \\ \text{ing machine with } \mathcal{O}(c \log^k n) \text{ alternation} \end{array} \right\}. \end{aligned}$$

$$\text{LOGCFL} = \text{SAC}^1.$$

Selected Facts and Theorems

1.  $\text{AC}^k \subseteq \text{NC}^{k+1}$ ,  $k \geq 0$ . [Ruz80]
2. Sorting can be done in  $\text{NC}^1$ , so, the parity function is in  $\text{NC}^1$ . [AKS83]
3. The parity function is not in  $\text{AC}^0$ , and thus,  $\text{AC}^0 \neq \text{NC}^1$ . Even stronger, no family of constant-depth, superpolynomial-size unbounded-fan-in circuits can compute the parity function. (see Chap. 8)
4. For each  $k \geq 1$ , there is a family of functions  $\mathcal{F} = \{f_n\}_{n \geq 1}$  such that  $\mathcal{F}$  can be computed by a family of depth- $k$ , polynomial-size unbounded-fan-in circuits but cannot be computed by a family of depth- $(k-1)$ , superpolynomial-size unbounded-fan-in circuits. (see Chap. 8)
5.  $\text{PL} \cup \oplus\text{L} \subseteq \text{NC}^2$ . [BCP83]
6.  $\text{NL} \subseteq \text{LOGCFL}$ . [Sud78]
7.  $\text{LOGCFL}$  is closed under complement. [BCD<sup>+</sup>89]
8.  $\text{SAC}^0$  is not closed under complement. However, for each  $k \geq 1$ ,  $\text{SAC}^k$  is closed under complement. [Ven91, BCD<sup>+</sup>89]

**Fig. A.11** NC, AC, and LOGCFL

Cook [Coo85] proposed to use notation  $NC^k$  in the honor of Pippenger, who characterized  $NC^k$  as the languages accepted by reversal bounded Turing machines [Pip79]. Chandra, Stockmeyer, and Vishkin [CSV84] studied the nonuniform version of AC. Cook [Coo85] proposed notation  $AC^k$  for the languages accepted by logspace alternating Turing machines with alternation bound  $\mathcal{O}(\log^k n)$ . Cook pointed out (attributed to Cook and Ruzzo in [Coo85]) that the class is the same as the uniform version of the class studied by Chandra, Stockmeyer, and Vishkin. The letter “A” in AC thus stands for “alternating.” Ruzzo [Ruz80] showed that  $AC^k \subseteq NC^{k+1}$  for all  $k \geq 0$ . Hence,  $AC = NC$ . Sudborough [Sud78] gave a complete characterization of LOGCFL as the languages accepted by nondeterministic auxiliary pushdown automata in logspace and polynomial-time, which characterization yields  $NL \subseteq LOGCFL$ . Ruzzo [Ruz80] showed that the pushdown automata class by Sudborough is included in  $AC^1$  thereby showing that  $LOGCFL \subseteq AC^1$ . Venkateswaran [Ven91] strengthened the upper bound by showing that  $LOGCFL = SAC^1$ .

Though we have used logspace-uniformity as the default uniformity type of  $NC^k$ ,  $AC^k$ , and  $SAC^k$ , many other types of uniformity are also important in the literature, ranging from P-uniformity down to extremely restrictive notions of uniformity. Regarding the former, see for example Allender’s paper [All89b]. Regarding the latter, we mention that  $U_E$ -uniformity, which was introduced by Ruzzo ([Ruz81], see also [Coo85]), is an attempt at capturing what one would mean by “ $NC^1$ -uniformity,” and is often used when studying  $NC^1$ . Uniformity types that are even more restrictive have been proposed and studied by Barrington, Immerman, and Straubing [BIS90]. Ruzzo [Ruz81] has compared various logspace-uniformity conditions and shown that  $NC^k$  with  $k \geq 2$  is robust under the choice of logspace-uniformity conditions.

## A.9 UP, FewP, and US: Ambiguity-Bounded Computation and Unique Computation

$$\begin{aligned}
 UP &= \left\{ L \mid \begin{array}{l} \text{there is a nondeterministic polynomial-time Turing} \\ \text{machine } N \text{ such that } L = L(N) \text{ and, for all } x, N(x) \\ \text{has at most one accepting path} \end{array} \right\}. \\
 US &= \left\{ L \mid \begin{array}{l} \text{there is a nondeterministic polynomial-time Turing} \\ \text{machine } N \text{ such that, for all } x, x \in L \iff N(x) \\ \text{has exactly one accepting path} \end{array} \right\}.
 \end{aligned}$$

Above, the  $N(x)$  is used to denote the computation of machine  $N$  on input  $x$ , and in particular is denoting above, as a shorthand, the computation tree of  $N$  on input  $x$ .

The classes UP and US capture the power of uniqueness (for UP, some prefer the term unambiguity). Given a boolean formula  $f$  a typical US question would be, “Does  $f$  have exactly one solution?” UP has a related but

subtly different nature. UP is the class of problems that have (on some NP machine) unique witnesses. That is, if there is an NP machine  $M$  accepting  $L$  and for every input  $x$  the computation  $M(x)$  has at most one accepting path, then we say  $L \in \text{UP}$ . We call NP machines that accept on at most one path for all inputs *categorical machines*. Valiant started the study of UP and categorical machines [Val76].

UP has come to play a crucial role in both cryptography and complexity theory. In cryptography theory, Grollmann and Selman [GS88] prove that one-to-one one-way functions<sup>4</sup> exist if and only if  $P \neq \text{UP}$ , and one-to-one one-way functions whose range is in  $P$  exist if and only if  $P \neq \text{UP} \cap \text{coUP}$ . Thus we suspect that  $P \neq \text{UP}$  because we suspect that one-to-one one-way functions exist.

A central question in complexity theory, first asked by Berman and Hartmanis [BH77], is "How many NP-complete problems are there?" Berman and Hartmanis conjectured that there is only one NP-complete problem, which appears in many guises. That is, they conjectured that all NP-complete sets are polynomial-time isomorphic (P-isomorphic). Indeed, they showed that all then-known and all paddable NP-complete sets are P-isomorphic ([BH77] and Mahaney and Young [MY85]). Note that the conjectured P-isomorphism of NP-complete sets implies  $P \neq \text{NP}$ .

Kurtz, Mahaney, and Royer [KMR95] have shown that relative to a random oracle there are NP-complete sets that are not P-isomorphic. Fenner, Fortnow, and Kurtz [FFK96] have shown that there is an oracle world in which all NP-complete sets are P-isomorphic.

Joseph and Young found NP-complete " $k$ -creative" sets that are *not* obviously P-isomorphic to SAT. However, if no one-to-one one-way functions exist then these sets are isomorphic to SAT. This led to the following conjecture (see [JY85,KLD86,KMR88,KMR95,Rog97]). Since one-to-one one-way functions exist if and only if  $P \neq \text{UP}$ , this conjecture links  $P = \text{UP}$  to the structure of NP.

**One-Way Conjecture** One-to-one one-way functions exist if and only if non-P-isomorphic NP-complete sets exist.

This coupling between UP and NP has been weakened. Hartmanis and Hemachandra [HH91a] show that there is a relativized world in which the One-way Conjecture fails. That is, there is a world in which there are no one-to-one one-way functions yet there are non-P-isomorphic NP-complete sets.

<sup>4</sup> By  $\text{range}(f)$  we denote  $\bigcup_{i \in \Sigma^*} f(i)$ . A function  $f$  is *honest* if  $(\exists \text{ polynomial } q)(\forall y \in \text{range}(f))(\exists x)[|x| \leq q(|y|) \wedge f(x) = y]$ . A *one-to-one one-way function* is a total, single-valued, one-to-one honest, polynomial-time computable function  $f$  such that  $f^{-1}$  (which will be a partial function if  $\text{range}(f) \neq \Sigma^*$ ) is not computable in polynomial time [GS88].



---

**UP, FewP, and US – Unambiguous Polynomial Time, Polynomial-Ambiguity Polynomial Time, Unique Polynomial Time**


---

**Power**

Categorical acceptance. Unambiguity; polynomial-bounded ambiguity; uniqueness.

**Definition**

$$\begin{aligned}
 \text{UP} &= \left\{ L \left| \begin{array}{l} \text{there is a nondeterministic polynomial-time Turing machine } N \text{ such that } L = L(N) \text{ and, for all } x, N(x) \text{ has at} \\ \text{most one accepting path} \end{array} \right. \right\}. \\
 \text{FewP} &= \left\{ L \left| \begin{array}{l} \text{there is a nondeterministic polynomial-time Turing machine } N \text{ and a polynomial } q \text{ such that } L = L(N) \text{ and,} \\ \text{for all } x, N(x) \text{ has at most } q(|x|) \text{ accepting paths} \end{array} \right. \right\}. \\
 \text{US} &= \left\{ L \left| \begin{array}{l} \text{there is a nondeterministic polynomial-time Turing machine } N \text{ such that, for all } x, x \in L \iff N(x) \text{ has} \\ \text{exactly one accepting path} \end{array} \right. \right\}.
 \end{aligned}$$

**Alternate Definition**

$$\begin{aligned}
 \text{UP} &= \left\{ L \left| \begin{array}{l} \text{there is a polynomial-time predicate } P \text{ and a polynomial } q \text{ such that, for all } x, \\ 1. x \notin L \implies ||\{y \mid |y| \leq q(|x|) \wedge P(x, y)\}|| = 0, \text{ and} \\ 2. x \in L \implies ||\{y \mid |y| \leq q(|x|) \wedge P(x, y)\}|| = 1 \end{array} \right. \right\}. \\
 \text{FewP} &= \left\{ L \left| \begin{array}{l} \text{there is a polynomial-time predicate } P, \text{ a polynomial } q, \\ \text{and a polynomial } q' \text{ such that, for all } x, \\ 1. x \notin L \implies ||\{y \mid |y| \leq q(|x|) \wedge P(x, y)\}|| = 0, \text{ and} \\ 2. x \in L \implies 1 \leq ||\{y \mid |y| \leq q(|x|) \wedge P(x, y)\}|| \leq q'(|x|) \end{array} \right. \right\}. \\
 \text{US} &= \left\{ L \left| \begin{array}{l} \text{there is a polynomial-time predicate } P \text{ and a polynomial } q \text{ such that, for all } x, x \in L \iff ||\{y \mid |y| \leq q(|x|) \wedge P(x, y)\}|| = 1 \end{array} \right. \right\}.
 \end{aligned}$$

**Background**

UP was defined by Valiant [Val76]. US was defined by Blass and Gurevich [BG82]. FewP was defined by Allender and Rubinstein [All86,AR88]. UP is related to cryptography [GS88] and some think it central to conjectures in complexity theory ([JY85,KMR88], but see also [HH91a]).

**Complete Problems**

UP and FewP may not have complete languages. There are relativized worlds in which they do not have complete languages (and indeed in which FewP lacks  $\leq_T^P$ -hard sets for UP) [HH88a,HJV93]. On the other hand, there are relativized worlds in which  $P^A \neq \text{UP}^A \neq \text{NP}^A$  yet  $\text{UP}^A$  does have complete languages [HH88a].

$\text{USAT} = \{f \mid f \text{ has exactly one satisfying assignment}\}$  is complete for US.

---

**Fig. A.12** UP, FewP, and US—part I
 

---

---

**UP, FewP, and US – Unambiguous Polynomial Time, Polynomial-Ambiguity Polynomial Time, Unique Polynomial Time**


---

**Selected Facts and Theorems**

1.  $P \subseteq UP \subseteq NP \cap US \subseteq NP \subseteq coUS$ .
2. UP is closed under intersection.
3.  $FewP \subseteq SPP = SPP^{FewP}$ . [KSTT92,FFK94,FFL96]
4. If UP has sparse  $\leq_T^P$ -hard sets, then all UP sets are Low<sub>2</sub>, i.e., for each set  $L$  in UP it holds that  $NP^{NP^L} = NP^{NP}$ . [HR97]
5.  $P \neq UP \iff$  one-to-one one-way functions exist. (see Chap. 2)
6. Unambiguous (i.e., one-to-one) one-way functions exist if and only if bounded-ambiguity one-way functions exist. Equivalently,  $P \neq UP \iff P \neq UP_{\leq k}$ . (see Chap. 2)
7.  $P \neq FewP \iff$  polynomial-to-one one-way functions exist.  
(see the Bibliographic Notes of Chap. 2)
8.  $P \neq UP \cap coUP \iff$  one-to-one one-way functions whose range is in  $P$  exist. [GS88]
9.  $P = FewP$  if and only if all sparse sets in  $P$  are  $P$ -printable. [AR88]
10. If UP has complete languages then it has a complete language of the form  $L = SAT \cap A$ ,  $A \in P$ . [HH88a]
11.  $(\exists A)[P^A \neq UP^A = NP^A]$ . [Rac82]
12.  $(\exists A)[P^A = UP^A \wedge NP = EXP]$  (and so relative to this oracle  $A$ ,  $NP^A$  not only differs from  $UP^A$  but even is  $P^A$ -immune, as a side effect of the known fact that EXP contains  $P$ -immune sets). [BBF98]
13.  $(\exists A)[P^A = FewP^A \neq NP^A]$ .  $(\exists B)[P^B \neq UP^B \neq FewP^B \neq NP^B]$ . [Rub88]
14.  $(\exists A)[UP^A$  has no complete languages]. [HH88a]
15.  $(\exists A)[P^A \neq UP^A \neq NP^A$  and  $UP^A$  has complete languages]. [HH88a]
16.  $(\forall A)[N_i^A \text{ is categorical}] \implies (\forall A)[L(N_i^A) \in P^{NP \oplus A}]$ . [HH90]
17. There is a reasonable (i.e.,  $P^A \neq NP^A$ ) oracle  $A$  for which  $P^A = UP^A$  (that is, there are no one-to-one one-way functions) yet there are sets that are  $\leq_{m, A}^P$ -complete for  $NP^A$  and are non- $P^A$ -isomorphic. [HH91a]
18.  $P \neq UP \cap coUP$  if and only if there is a set  $S$  so (1)  $S \in P$  and  $S \subseteq SAT$ , and (2)  $f \in S \implies f$  has exactly one solution, and (3) no  $P$  machine can find solutions for all formulas in  $S$ —that is,

$$g(f) = \begin{cases} 0 & f \notin S \\ \text{the unique satisfying assignment of } f & f \in S \end{cases}$$

is not a polynomial-time computable function. [HH88a]

19.  $P$ ,  $UP$ , and  $NP$  all differ with probability one relative to a random oracle. [Bei89,NRRS98]
  20.  $Primes \in UP \cap coUP$ . [FK92]
- 

**Fig. A.13** UP, FewP, and US—part II

Their oracle consists of a collapsing component (PSPACE) unioned with an extraordinarily sparse diagonalizing component.

**Theorem A.2** *There is a reasonable (i.e.,  $P^A \neq NP^A$ ) oracle  $A$  for which  $P^A = UP^A$  (that is, there are no one-to-one one-way functions) yet there are sets that are  $\leq_m^{P^A}$ -complete for  $NP^A$  and are non- $P^A$ -isomorphic.*

This does not imply that the One-Way Conjecture is false, though it does open that possibility. This theorem, however, suggests that the conjecture is unlikely to be proved by standard techniques.

There is a “promise” in the definition of UP. In particular, a UP machine must have the property that on each input, its number of accepting paths is either one or zero. There is no known way to enumerate all machines having this property without also enumerating machines not having this property. Since such enumerations are a central tool in proving the existence of complete sets [Sip82, HH88a, BCS92, Bor94], this precludes the standard method of proving that the class has complete sets. In fact, there are relativized worlds in which UP lacks complete sets ([HH88a], see also [HJV93]).

Attempts to find an NP analog of Rice’s Theorem have instead led to analogs of Rice’s Theorem for UP (unambiguous polynomial-time) and its constant-ambiguity cousins. In particular, all nontrivial counting properties of circuits are hard for these classes ([BS00, HR00], see also [HT]).

The class FewP, defined by Allender and Rubinfeld [All86, AR88], is an analogue of UP that restricts machines not to one accepting path but to at most polynomially many accepting paths. Clearly,  $P \subseteq UP \subseteq \text{FewP} \subseteq NP$ , and Allender and Rubinfeld [AR88] show that  $P = \text{FewP}$  if and only if all sparse sets in  $P$  are  $P$ -printable.<sup>5</sup>

**Definition A.3**  *$L \in \text{FewP}$  if there is a nondeterministic polynomial-time Turing machine  $N$  so that  $N$  accepts language  $L$  and for some polynomial  $q$ ,*

$$(\forall x) [N(x) \text{ has at most } q(|x|) \text{ accepting paths}].$$

Many authors prefer to use the term *unambiguous* computation to refer to UP, and reserve the term *unique* computation for the class US. Note that there is a key difference between UP and FewP on one hand, and US on the other hand. UP and FewP are indeed about computation that has a limit on the ambiguity (the number of solutions, i.e., accepting paths, of the underlying machine). In contrast, though the machine for a US set by definition accepts exactly when there is exactly one accepting computation path, it is legal for the machine on some inputs to have huge numbers of accepting paths—it merely is the case that such inputs are not members of the set.

<sup>5</sup> A set  $S$  is *P-printable* if there is a polynomial-time Turing machine  $M$  such that for each  $n$ ,  $M(1^n)$  prints all elements of  $S$  of length at most  $n$  [HY84].

---

**#P – Sharp P (Counting Solutions)**

Power

Counting solutions.

Definition

$$\#P = \{f \mid (\exists \text{ nondeterministic polynomial-time Turing machine } N) (\forall x) [f(x) = \#acc_N(x)]\}.$$

Background

#P was first studied by Valiant [Val79a], who showed that counting versions not only of NP-complete problems but also of some P problems can be #P-complete.

Complete Problems

#SAT, the function mapping from boolean formulas to their numbers of solutions, is a representative #P function:  $P^{\#P[1]} = P^{\#SAT[1]}$ .

**Fig. A.14** #P—part I

---

**A.10 #P: Counting Solutions**

*One potato, two potato, three potato, four,  
Five potato, six potato, seven potato, more.  
—Children's Rhyme*

$$\#P = \{f \mid (\exists \text{ nondeterministic polynomial-time Turing machine } N) (\forall x) [f(x) = \#acc_N(x)]\},$$

where  $\#acc_N(x)$  denotes the number of accepting paths of  $N(x)$ .

#P is the class of *functions* that count the accepting paths of nondeterministic polynomial-time Turing machines. For example, the function that maps any boolean formula to its number of satisfying assignments is a #P function. To create a language class, as opposed to a function class, we usually discuss  $P^{\#P}$ . Toda [Tod91c] has shown that  $P^{\#P} \supseteq PP^{PH}$  (Chap. 4).

#P is closely related to PP, probabilistic polynomial time:  $P^{PP} = P^{\#P}$  [BBS86].

The possibility of approximating #P functions has been much studied. Stockmeyer [Sto85] shows that  $\Delta_3^P$  machines can approximate #P functions within a tight factor. Cai and Hemachandra ([CH91], see also [CH89]) and, independently, Amir, Beigel, and Gasarch [ABG00], show that the range of #P functions cannot be reduced to polynomial size unless  $P = P^{\#P}$  (Chap. 6).

#P is intimately connected to the complexity of ranking—determining the position of elements in a set [GS91,HR90,Huy90,BGS91].

Though #P intuitively is the counting analog of NP, there are some curious flaws in the analogy. Valiant [Val79a] has shown that even some P

---

**#P – Sharp P (Counting Solutions)**
**Selected Facts and Theorems**

1.  $PP^{PH} \subseteq P^{\#P} \subseteq PSPACE$ . (see Chap. 4)
2.  $P^{PP} = P^{\#P}$ . [BBS86]
3. If #SAT has a polynomial-time computable enumerator then  $P = P^{\#P}$ . (see Chap. 6)
4. If there is an NP-complete set  $L$  that with respect to some polynomial-time witnessing relation for it,  $R_L$ , is not #P-complete, then  $P \neq P^{\#P}$ . [FHT97]
5. If  $P \neq P^{\#P}$  and  $FewP = NP$ , then each NP-complete set has some polynomial-time witnessing relation with respect to which it fails to be #P-complete. [FHT97]
6.  $PP^{\oplus P} \subseteq P^{\#P[1]}$ . (see Chap. 4)
7. #P is closed under addition and multiplication. (see Chap. 5)
8. The following are equivalent:
  - a)  $UP = PP$ .
  - b) #P is closed under proper subtraction.
  - c) #P is closed under integer division.
  - d) #P is closed under every polynomial-time computable operation. (see Chap. 5)
9. If #P is closed under proper decrement, then  $coNP \subseteq SPP$  and  $NP \subseteq FTM_kP$ . (see the text and Bibliographic Notes of Chap. 5)
10. If  $UP = NP$ , then #P is closed under proper decrement. (see Chap. 5)
11. If #P is closed under integer division by two, then  $\oplus P = SPP$  (and so  $PH \subseteq PP$ ). (see Chap. 5)
12. If #P is closed under minimum, then  $NP = UP$  and  $C=P = SPP$ . (see Chap. 5)
13. If #P is closed under maximum, then  $C=P = SPP$ . (see Chap. 5)
14. If #P is closed under integer division by two, then  $\oplus P = SPP$  (and so  $PH \subseteq PP$ ). (see Chap. 5)

**Open Problems**

- $P^{\#P[1]} = P^{\#P}$ ?
- $P^{\#P} = PSPACE$ ?
- Find a complexity class equality that completely characterizes whether #P is closed under proper decrement.

**Fig. A.15 #P—part II**


---

sets have #P-complete counting versions, at least under some reducibilities. And Fischer, Hemaspaandra, and Torenvliet [FHT97] have shown that under certain complexity-theoretic assumptions, not all counting versions of NP-complete sets are  $\leq_{1-T}^P$ -complete for #P.

Goldsmith, Ogihara, and Rothe [GOR00] have studied the complexity of  $\#P_1$  [Val79b], the tally analog of #P.

---

**ZPP, RP, coRP, and BPP – Error-Bounded Probabilism**


---

Power

Error-bounded probabilism.

Definition

$$\begin{aligned}
 \text{BPP} &= \left\{ L \mid \begin{array}{l} \text{there is a probabilistic polynomial-time Turing machine } \\ M \text{ so that for each } x \text{ it holds that (a) if } x \in L \text{ then} \\ \Pr[M(x) \text{ accepts}] \geq 3/4, \text{ and (b) if } x \notin L \text{ then } \Pr[M(x) \\ \text{rejects}] \geq 3/4 \end{array} \right\} . \\
 \text{RP} &= \left\{ L \mid \begin{array}{l} \text{there is a probabilistic polynomial-time Turing machine } \\ M \text{ so that for each } x \text{ it holds that (a) if } x \in L \text{ then} \\ \Pr[M(x) \text{ accepts}] \geq 1/2, \text{ and (b) if } x \notin L \text{ then } \Pr[M(x) \\ \text{rejects}] = 1 \end{array} \right\} . \\
 \text{coRP} &= \{ L \mid \overline{L} \in \text{RP} \} . \\
 \text{ZPP} &= \text{RP} \cap \text{coRP} .
 \end{aligned}$$

Alternate Definition

$$\begin{aligned}
 \text{BPP} &= \left\{ L \mid \begin{array}{l} \text{there is a polynomial-time predicate } P \text{ and a polynomial} \\ q \text{ such that, for all } x, \\ \begin{array}{l} 1. x \notin L \implies ||\{y \mid |y| \leq q(|x|) \wedge P(x, y)\}|| \leq \frac{1}{4} 2^{q(|x|)}, \\ \text{and} \\ 2. x \in L \implies ||\{y \mid |y| \leq q(|x|) \wedge P(x, y)\}|| \geq \frac{3}{4} 2^{q(|x|)} \end{array} \end{array} \right\} . \\
 \text{RP} &= \left\{ L \mid \begin{array}{l} \text{there is a polynomial-time predicate } P \text{ and a polynomial} \\ q \text{ such that, for all } x, \\ \begin{array}{l} 1. x \notin L \implies ||\{y \mid |y| \leq q(|x|) \wedge P(x, y)\}|| = 0, \text{ and} \\ 2. x \in L \implies ||\{y \mid |y| \leq q(|x|) \wedge P(x, y)\}|| \geq 2^{q(|x|)-1} \end{array} \end{array} \right\} .
 \end{aligned}$$

Background

Gill [Gil77] wrote the seminal paper on error-bounded probabilistic computation.

Complete Languages

No complete sets are known for any of these classes. There are relativized worlds in which ZPP has no  $\leq_T^P$ -hard set in BPP (or even IP), and thus in which none of these classes have complete sets ([HJV93], see also [Sip82]).

**Fig. A.16** ZPP, RP, coRP, and BPP—part I

---

**A.11 ZPP, RP, coRP, and BPP: Error-Bounded Probabilism**


---

A language  $L$  is in BPP if there is a probabilistic polynomial-time Turing machine  $M$  (essentially, a Turing machine that can flip an unbiased coin) such that for each  $x \in L$  it holds that  $M(x)$  accepts with probability at least  $3/4$ , and for each  $x \notin L$  it holds that  $M(x)$  rejects with probability at least  $3/4$ .

**ZPP, RP, coRP, and BPP – Error-Bounded Probabilism**

## Selected Facts and Theorems

1.  $P \subseteq ZPP = RP \cap coRP \subseteq_{coRP}^{RP} \subseteq BPP$ . [Gil77]
2.  $RP \subseteq NP$ . [Gil77]
3.  $NP^{BPP} \subseteq ZPP^{NP}$ . [ZF87]
4.  $ZPP^{NP^{BPP}} = ZPP^{NP}$ . ([AK01], see also [ZH86,GZ97])
5.  $BPP^{BPP} = BPP$ . [Ko82,Zac82]
6.  $ZPP^{ZPP} = ZPP$ . [Zac82]
7.  $BPP \subseteq P/poly$ . (see Chap. 4)
8.  $BPP^{\oplus P} \subseteq P^{\#P[1]}$ . (see Chap. 4)
9.  $PH \subseteq BPP^{\oplus P}$ . (see Chap. 4)
10.  $NP \subseteq BPP \implies RP = NP$ . [Ko82]
11.  $NP \subseteq BPP \implies PH \subseteq BPP$ . [Zac88]
12.  $Primes \in ZPP$ . [AH92]
13. If  $A \in BPP$ , then  $NP^{NP^A} \subseteq NP^{A \oplus SAT}$ . In particular,  $NP^{NP^{BPP} \cap NP} = NP^{NP}$ . [Sch86b]
14. If #GA, the function counting the number of automorphisms of graphs has a polynomial-time computable enumerator then the Graph Isomorphism Problem,  $\{\langle G, H \rangle \mid G \text{ and } H \text{ are isomorphic graphs}\}$ , belongs to the class RP. [BCGT99]

**Fig. A.17** ZPP, RP, coRP, and BPP—part II

Most computer scientists, if stopped on the street and asked for a definition of “feasible computation,” would say “P” and walk on. Yet, there is another possibility: BPP. Suppose the error probability of the machine described above is, on each input  $x$ , bounded not by  $1/4$  but rather by  $1/2^{|x|}$ . (It is not hard to show—simply by taking polynomially many trials and rejecting or accepting as the majority do—that each BPP language does have such low-error machines.) For all sufficiently large  $x$  (and, after all, for all other  $x$  we can in theory just use table lookup), the probability the answer is wrong due to this  $1/2^{|x|}$  error probability is less in practice than the probability that an earthquake levels the building or that the physical parts of the computer suddenly fail. Thus, many people accept low-error probabilistic complexity classes (i.e., ZPP, RP, coRP, and BPP) as intuitively “feasible.” Indeed, under a certain very plausible complexity-theoretic assumption, it would even follow that  $P = BPP$  [IW97]. On the other hand, to present a fair picture we should mention that the assumption that a computer can fairly generate random bits is less innocuous than it seems (however, there is interesting work on dealing with biased sources, see, e.g., [VV85]). Also, it is important to stress that BPP is characterized in terms of the *probability* of acceptance being bounded away from  $1/2$ , not by the proportion of accepting paths being bounded away from  $1/2$ . The latter notion seems to define a larger class ([HHT97], see also [JMT96,AFF<sup>+</sup>01]).

RP and coRP are classes similar to BPP, but allow only “one-sided” error (see Fig. A.16). ZPP is the class of languages accepted by zero-error computers whose *expected* running times are polynomial. Equivalently,  $ZPP = RP \cap \text{coRP}$  [Gil77,Zac82].

Probabilistic classes play a central role in complexity theory and computing. For example, though it is not known whether testing primality can be done deterministically in polynomial time, Adleman and Huang have shown that primality is in ZPP [AH92].

## A.12 PP, C=P, and SPP: Counting Classes

A language  $L$  is in PP [Sim75,Gil77] if there is a probabilistic polynomial-time Turing machine  $M$  such that, for each  $x$ ,

$$x \in L \iff M(x) \text{ accepts with probability at least } 1/2.$$

A language  $L$  is in C=P [Sim75,Wag86] if there is a polynomial-time computable function  $f$  and a NPTM  $N$  such that, for each  $x$ ,

$$x \in L \iff \#acc_N(x) = f(x),$$

where  $\#acc_N(x)$  denotes the number of accepting paths of  $N$  on input  $x$ . A language  $L$  is in SPP [OH93,FFK94] if there is a polynomial-time computable function  $f$  and a NPTM  $N$  such that, for each  $x$ ,

$$x \notin L \implies \#acc_N(x) = f(x) - 1, \text{ and}$$

$$x \in L \implies \#acc_N(x) = f(x).$$

Many counting classes have been defined and shown to be important in the study of previously defined notions. The classes usually attempt to extract out the essence of some particular computational task. For example, we may loosely think of PP as encapsulating the power of majority testing, and of C=P as encapsulating the power of exact equality testing.

Though BPP is a quite strong candidate for the title of “outer limit of feasible computation,” PP is not. The reason is that PP has no bound on its error. In fact, for PP machines, the difference between acceptance and rejection is so slight—one over an exponential function of the input—that we would need an exponential number of Monte Carlo tests to get any useful information. However, if one were willing to do an exponential amount of work, one could just as well exactly solve the PP problem by brute force.

PP, however, does have some nice properties. In particular, there is no “promise” built into its definition, and thus it is not hard to show that it has complete sets. The same also holds for C=P. However, in contrast, there is a “promise” (see the discussion in Sect. A.9) in the definition of SPP.



---

**PP, C=P, and SPP – Counting Classes**
**Power**

PP: Unbounded error probabilism.

C=P: Exact counting.

SPP: Generalized UP.

**Definition**

$$\begin{aligned}
 \text{PP} &= \left\{ L \left| \begin{array}{l} \text{there is a probabilistic polynomial-time Turing machine } M \\ \text{so that for each } x \text{ it holds that } x \in L \text{ if and only if} \\ \Pr[M(x) \text{ accepts}] \geq 1/2 \end{array} \right. \right\}. \\
 \text{C=P} &= \left\{ L \left| \begin{array}{l} \text{there is a nondeterministic polynomial-time Turing machine } N \text{ and a (deterministic) polynomial-time} \\ \text{computable function } f \text{ so that for each } x \text{ it holds that } x \in L \\ \text{if and only if } \#acc_N(x) = f(x) \end{array} \right. \right\}. \\
 \text{SPP} &= \left\{ L \left| \begin{array}{l} \text{there is a nondeterministic polynomial-time Turing machine } N \text{ and a (deterministic) polynomial-time} \\ \text{computable function } f \text{ so that for each } x \text{ it holds that (a) if} \\ x \in L \text{ then } \#acc_N(x) = f(x), \text{ and (b) if } x \notin L \text{ then} \\ \#acc_N(x) = f(x) - 1 \end{array} \right. \right\}.
 \end{aligned}$$

**Fig. A.18** PP, C=P, and SPP—part I

---

**A.13 FP, NPSV, and NPMV: Deterministic and Nondeterministic Functions**

We say that a function is in FP if it is computed by some deterministic polynomial-time Turing machine. Functions in FP must be single-valued, but they may potentially be partial.

The classes NPSV and NPMV capture the power of nondeterministic function computation. In particular, consider any nondeterministic polynomial-time Turing machine  $N$ . On any input  $x$ , we will consider  $N$  to have a (possibly empty) set of outputs. Namely, on input  $x$ , each string  $y$  that appears on the worktape of  $N$  along at least one computation path that halts and accepts is considered to belong to the output set. A function  $f$  is said to belong to NPMV if there exists some nondeterministic polynomial-time Turing machine  $N$  such that on each input  $x$  the outputs of  $f$  are exactly the outputs of  $N$ . As a notation, we use  $\text{set-}f(x)$  to denote  $\{a \mid a \text{ is an output of } f(x)\}$ . For example, on inputs  $x$  where the partial function  $f(x)$  is undefined, we have  $\text{set-}f(x) = \emptyset$ . Note that functions in NPMV may be partial or may be total, and may be single-valued or may be multivalued. Note that the multiplicities of appearances of outputs do not concern us here; if, on input  $x$ , machine  $N$  outputs 101 on one path and outputs 0011 on seventeen paths, its set of outputs is simply  $\{101, 0011\}$ .

NPSV denotes the set of all NPMV functions  $f$  that are single-valued, i.e., for each input  $x$ ,  $|\text{set-}f(x)| \leq 1$ .

---

**PP, C=P, and SPP – Counting Classes**
**Alternate Definition**

A language  $L$  is in PP if there exists a polynomial  $q$  and a polynomial-time predicate  $R$  such that, for each  $x$ ,

$$x \in L \iff |\{y \mid |y| = q(|x|) \wedge R(x, y)\}| \geq 2^{q(|x|)-1}.$$

A language  $L$  is in C=P if there exists a polynomial  $q$ , a polynomial-time function  $f$ , and a polynomial-time predicate  $R$  such that, for each  $x$ ,

$$x \in L \iff |\{y \mid |y| = q(|x|) \wedge R(x, y)\}| = f(x).$$

A different definition, which again yields the same class, allows us to fix the function  $f$  to be a of a very simple form. In particular,  $R$  such that, for each C=P has an alternate definition in which the “acceptance” cardinality is set to be exactly half of the total number of possibilities: A language  $L$  is in C=P if there exists a polynomial  $q$  and a polynomial-time predicate  $R$  such that, for each  $x$ ,

$$x \in L \iff |\{y \mid |y| = q(|x|) \wedge R(x, y)\}| = 2^{q(|x|)-1}.$$

A language  $L$  is in SPP if there exists a polynomial  $q$ , a polynomial-time function  $f$ , and a polynomial-time predicate  $R$  such that, for each  $x$ ,

1.  $x \in L \implies |\{y \mid |y| = q(|x|) \wedge R(x, y)\}| = f(x)$ , and
2.  $x \notin L \implies |\{y \mid |y| = q(|x|) \wedge R(x, y)\}| = f(x) - 1$ .

A different definition, which again yields the same class, allows us to fix the function  $f$  to be a of a very simple form. A language  $L$  is in SPP if there exists a polynomial  $q$ , a polynomial  $p$ , and a polynomial-time predicate  $R$  such that, for each  $x$ ,

1.  $x \in L \implies |\{y \mid |y| = q(|x|) \wedge R(x, y)\}| = 2^{p(|x|)} + 1$ , and
2.  $x \notin L \implies |\{y \mid |y| = q(|x|) \wedge R(x, y)\}| = 2^{p(|x|)}$ .

---

**Fig. A.19** PP, C=P, and SPP—part II

We need some natural way to speak of reducing the number of outputs that a function has. Refinement captures this notion.

**Definition A.4** *We say that a multivalued function  $f$  is a refinement of multivalued function  $g$  if*

1.  $(\forall x) [\text{set-}f(x) = \emptyset \iff \text{set-}g(x) = \emptyset]$ , and
2.  $(\forall x) [\text{set-}f(x) \subseteq \text{set-}g(x)]$ .

Intuitively, a refinement of a function is the function except on each input some (but not all) outputs may be missing. Note that an NPSV refinement of  $g$  is  $g$  with, on each input, all but one output removed. The question of whether all NPMV functions have NPSV refinements is central in Chap. 3.

We note that whether all NPMV functions have NPSV refinements seems to be a different issue than whether  $\text{UP} = \text{NP}$ . That is, it is not currently

---

**PP, C=P, and SPP – Counting Classes**
**Background**

PP was introduced in the work of Simon [Sim75] and Gill [Gil77]. C=P was introduced (though not assigned a name) by Simon [Sim75], who also proved that  $C=P \subseteq PP$ .

**Complete Languages**

Simon [Sim75] proved that MajSat is PP-complete, where  $\text{MajSat} = \{f \mid f \text{ is satisfied by more than half of the possible variable assignments}\}$ . C=P has canonical complete sets, but is not currently known to have any particularly natural complete sets. Toda [Tod94] and Ogiwara [Ogi92] have studied, respectively, canonical complete problems for  $P^{PP}$  and  $P^{C=P}$ . SPP is a “promise”-like class and thus seems to lack complete sets. Mundhenk, Goldsmith, Lusena, and Allender [MGLA00] have shown that  $PP^{NP}$  has natural complete sets.

**Selected Facts and Theorems**

1.  $P \subseteq UP \subseteq SPP \subseteq \oplus P \subseteq PP$ .
2.  $US \subseteq C=P \subseteq PP \subseteq P^{\#P} \subseteq PSPACE$ .
3. Both C=P and  $coC=P$  are closed downward under positive Turing reductions. [Ogi94b]
4.  $R_{tt}^P(C=P) = P\text{-uniform}NC^1(C=P) = P\text{-uniform}AC^1(C=P)$ . [Ogi95a]
5.  $P^{PP} = P^{\#P}$ . [BBS86]
6.  $NP^{\#P} = NP^{C=P}$ . [Tor91]
7.  $PP^{PH} \subseteq BPP^{C=P} \subseteq P^{PP}$ . (see Chap. 4)
8.  $\Theta_2^P \subseteq PP$ . [BHW91]
9. There is an oracle  $A$  such that  $P^{NP^A} \not\subseteq PP^A$ . [Bei94]
10. PP is closed downward under truth-table reductions. (see Chap. 9)
11.  $SPP^{SPP} = SPP^{\text{FewP}} = PP$ . [FFK94]
12.  $C=P^{SPP} = C=P^{\text{FewP}} = C=P$ . [FFK94, KSTT92]
13.  $PP^{SPP} = PP^{\text{FewP}} = PP$ . [FFK94, KSTT92]
14. There exists an oracle  $A$  such that  $SPP^A$  strictly contains  $PH^A$  yet  $PH^A$  does not collapse. [For99]

---

**Fig. A.20** PP, C=P, and SPP—part III
 

---

known that either implies the other. The barrier to proving “UP = NP if all NPMV functions have NPSV refinements” is that NPSV functions, though possessing at most one output, may potentially output that one output on many accepting paths; thus, the computation is in no way UP-like. Regarding the other direction, though

$$UP = NP \implies \text{all NPMV functions have } FP^{UP} \text{ refinements,}$$

there seems to be no obvious way to use  $UP = NP$  to obtain NPSV refinements of NPMV functions.

---

**FP, NPSV, and NPMV – Deterministic and Nondeterministic Functions**
**Power**

Function versions of P and NP.

**Definition**

FP denotes the class of (single-valued, possibly partial) functions computed by deterministic polynomial-time Turing machines. Given a nondeterministic polynomial-time Turing machine, we may view it as computing a (possibly partial, possibly multivalued) function as follows. Each rejecting path is viewed as outputting no value. Each accepting path is viewed as outputting whatever value it has written on its worktape. The machine, on input  $x$ , maps from  $x$  to the set of all values that are an output of some accepting computation path. The set of all such functions is denoted NPMV. The class of all NPMV functions that on no input take on more than one value is denoted NPSV.  $\text{set-}f(x)$  denotes  $\{a \mid a \text{ is an output of } f(x)\}$ . We say that a multivalued function  $f$  is a *refinement* of multivalued function  $g$  if

1.  $(\forall x) [\text{set-}f(x) = \emptyset \iff \text{set-}g(x) = \emptyset]$ , and
2.  $(\forall x) [\text{set-}f(x) \subseteq \text{set-}g(x)]$ .

**Background**

NPSV and NPMV were introduced by Book, Long, and Selman [BLS84,BLS85].

**Sample Functions**

Let  $f_{\text{SAT}}$  represent the function such that  $\text{set-}f_{\text{SAT}}(F)$  is empty if  $F$  is unsatisfiable, and is  $\{a \mid a \text{ is a satisfying assignment of } F\}$  if  $F$  is satisfiable.  $f_{\text{SAT}}$  is in NPMV.

**Selected Facts and Theorems**

1.  $\text{FP} \subseteq \text{NPSV} \subsetneq \text{NPMV}$ .
2.  $\text{P} = \text{NP} \iff \text{FP} = \text{NPSV}$ . (see [Sel94])
3. For every  $k \geq 1$ ,  $\text{P}_{k\text{-tt}}^{\text{NP}} = \text{P}_{k\text{-tt}}^{\text{NPSV}}$  and  $\text{P}_{k\text{-T}}^{\text{NP}} = \text{P}_{k\text{-T}}^{\text{NPSV}}$ . [FHOS97]
4. If every NPMV function has an NPSV refinement, then  $\text{PH} = \text{ZPP}^{\text{NP}}$ . (see Chap. 3)

**Fig. A.21** FP, NPSV, and NPMV

---

## A.14 P-Sel: Semi-feasible Computation

$$\text{P-sel} = \left\{ L \left| \begin{array}{l} \text{there is a polynomial-time 2-ary function } f \text{ such that} \\ \text{for each } x \text{ and } y \text{ it holds that (a) } f(x, y) \in \{x, y\}, \text{ and} \\ \text{(b) } \{x, y\} \cap L \neq \emptyset \implies f(x, y) \in L \end{array} \right. \right\}.$$

P denotes the class of sets that have polynomial-time membership algorithms. P-sel, the class of P-selective sets, denotes the class of sets that have polynomial-time *semi-membership* algorithms. A semi-membership algorithm for a set is a function—called a *selector function*—that, given two inputs, chooses one that is “logically more likely” (or, to be more accurate, “logically no less likely”) to be in the set in the sense that if exactly one of the two inputs is in the set then the algorithm chooses that input.

---

**P-sel – Semi-feasible Computation**
**Power**

Semi-feasible computation.

**Definition**

$$\text{P-sel} = \left\{ L \mid \begin{array}{l} \text{there is a polynomial-time 2-ary function } f \text{ such that} \\ \text{for each } x \text{ and } y \text{ it holds that (a) } f(x, y) \in \{x, y\} \text{ and} \\ \text{(b) } \{x, y\} \cap L \neq \emptyset \implies f(x, y) \in L \end{array} \right\}.$$

**Background**

The P-selective sets were introduced by Selman [Sel79, Sel82b, Sel82a] as a polynomial-time analog of the semi-recursive sets from recursive function theory [Joc68]. The NPSV-selective sets were first studied by Hemaspaandra, Naik, Ogihara, and Selman [HNOS96b].

**Sample Language**

For any real number  $0 \leq a < 1$ , the left cut of  $a$  is a P-selective set, where the left cut of a number in the range  $[0, 1)$  is the set  $\{b_1 b_2 b_3 \cdots b_k \mid k \geq 0 \wedge a > 0.b_1 b_2 b_3 \cdots b_k\}$  and  $0.b_1 b_2 b_3 \cdots b_k$  denotes the binary fraction denoted by the given bits.

**Fig. A.22** P-sel—part I

---

In the 1960s, Jockusch [Joc68] first studied semi-membership algorithms, by studying the class of languages—the semi-recursive sets—having recursive selector functions.

Around 1980, Selman [Sel79, Sel82a, Sel82b] introduced the P-selective sets—the class of sets having polynomial-time selector functions. Selman and other researchers obtained many important foundational results [Sel79, Sel82a, Sel82b, Ko83]. There followed a half-decade in which relatively little attention was paid to the P-selective sets. Then, around 1990, there was an abrupt and intense renewal of interest in the P-selective sets. In a flurry of progress (surveyed by Denny-Brown, Han, Hemaspaandra, and Torenvliet [DHHT94]), longstanding open problems were resolved and new variants were introduced and explored. Of particular interest to this book are the NPSV-selective sets of Hemaspaandra, Naik, Ogihara, and Selman [HNOS96b].

**Definition A.5** *A set  $L$  is NPSV-selective if there is a function  $f \in \text{NPSV}$  such that*

1.  $(\forall x, y) [\text{set-}f(x, y) \subseteq \{x, y\}]$ , and
2.  $(\forall x, y) [\{x, y\} \cap L \neq \emptyset \implies (\text{set-}f(x, y) = \{x\} \vee \text{set-}f(x, y) = \{y\})]$ .

The motivations for studying selectivity are varied. One motivation is as a relaxation of P. Given that membership in P is open for a wide range of important sets, it is natural to define generalizations of P and see whether these generalizations capture such important sets. A great variety of such classes

---

**P-sel – Semi-feasible Computation**
**Selected Facts and Theorems**

1.  $P \subseteq \text{P-sel} \not\subseteq \text{RECURSIVE}$ .
2. The P-selective sets are closed under complementation.
3. The P-selective sets are closed downward under  $\leq_{pos}^P$ -reductions. [BTvEB93]
4. The P-selective sets are closed under exactly  $2k + 2$  of the  $k$ -ary boolean functions properties, namely, those that are either completely degenerate or almost-completely degenerate. In particular, the P-selective sets are closed under neither union nor intersection. [HJ95b]
5.  $NP \subseteq \text{P-sel} \implies P = NP$ . Indeed,  $NP \subseteq R_{tt}^P(\text{P-sel}) \implies P = NP$ . [Sel79,AA96,BKS95,Ogi95b]
6. a) If all sets in UP are  $\leq_{tt}^P$ -reducible to P-selective sets then  $P = UP$ .  
b) If all sets in NP are  $\leq_{tt}^P$ -reducible to P-selective sets then  $P = \text{FewP}$  and  $RP = NP$ .  
c) If all sets in  $P^{NP}$  are  $\leq_{tt}^P$ -reducible to P-selective sets then  $P = NP$ . [Tod91b]
7. If there exists a P-selective set that is truth-table-hard for NP then, for all  $k > 0$ ,  $\text{SAT} \in \text{DTIME}[2^{n/\log^k n}]$ . [NS99]
8.  $\text{P-sel} \subseteq P/\text{poly}$  (indeed, even  $\text{P-sel} \subseteq P/\text{quadratic}$ ). (see Chap. 3)
9.  $\text{P-sel} \subseteq NP/\text{linear} \cap \text{coNP}/\text{linear}$ . (see Chap. 3)
10.  $\text{P-sel} \not\subseteq NP/n$ . (see Chap. 3)
11. If  $NP \subseteq \text{NPSV-sel}$  then the polynomial hierarchy collapses. (see Chap. 3)
12.  $\text{NPSV-sel} \cap NP \subseteq (NP \cap \text{coNP})/\text{poly}$ . (see Chap. 3)
13.  $\text{NPSV-sel} \subseteq NP/\text{poly} \cap \text{coNP}/\text{poly}$ . [HNOS96b]
14.  $\text{P-sel} \subsetneq R_{1-T}^P(\text{P-sel}) = R_{1-tt}^P(\text{P-sel}) = E_{1-T}^P(\text{P-sel}) = E_{1-tt}^P(\text{P-sel}) \subsetneq R_{2-tt}^P(\text{P-sel}) \subsetneq \dots \subsetneq R_{k-tt}^P(\text{P-sel}) \subsetneq R_{(k+1)-tt}^P(\text{P-sel}) \subsetneq \dots$  [HHO96]
15.  $\text{P-sel} \subsetneq E_{1-T}^P(\text{P-sel}) \subsetneq E_{2-T}^P(\text{P-sel}) \subsetneq \dots \subsetneq E_{k-T}^P(\text{P-sel}) \subsetneq E_{(k+1)-T}^P(\text{P-sel}) \subsetneq \dots$  [HHO96]
16. Any Turing self-reducible P-selective set is in P. [BT96b]
17. If  $P = PP$  then for every non-empty P-selective set  $A$  there exists a standard left-cut  $L(r)$  such that  $A \equiv_m^P L(r)$ . [HNOS96a]
18. If  $A$  is a P-selective set, then  $NP^{NP^A} \subseteq NP^{A \oplus \text{SAT}}$ . In particular,  $NP^{NP^{\text{P-sel}} \cap NP} = NP^{NP}$ . [KS85,ABG00]
19.  $E_T^P(\text{P-sel}) \subseteq \text{EXP}/\text{linear}$ . [BL97]

**Fig. A.23** P-sel—part II

---

have been defined: the P-selective sets, the P-close sets [Sch86a], the near-testable sets [GHJY91], the nearly near-testable sets [HH91b], the almost polynomial-time sets [MP79], etc. However, the P-selective sets stand out from the crowd in their centrality in complexity theory. The NPSV-selective sets are, somewhat curiously, best motivated simply as a tool. In particular, they offer the key bridge to proving that unique solutions collapse the polynomial hierarchy (Chap. 3).

The NPSV-selective sets are not the only generalization of the P-selective sets. Many other generalizations or variations have been developed and studied [HHN<sup>+</sup>95,Ogi95b,HZZ96,Wan95,HJR97,Zim98,ABG00,Nic00].

Finally, it is very useful to be able to assume that selector functions are oblivious to the order of their arguments. Let  $f'(\cdot, \cdot)$  be a selector function for  $L$ . Note that that  $f(x, y) = f'(\min\{x, y\}, \max\{x, y\})$  is also a selector function for  $L$ .

### Proposition A.6

1. If  $L$  is a P-selective set, then  $L$  is P-selective via some selector function  $f \in \mathbf{P}$  such that  $(\forall x, y) [f(x, y) = f(y, x)]$ .
2. If  $L$  is a NPSV-selective set, then  $L$  is NPSV-selective via some selector function  $f \in \mathbf{NPSV}$  such that  $(\forall x, y) [\text{set-}f(x, y) = \text{set-}f(y, x)]$ .

## A.15 $\oplus \mathbf{P}$ , $\text{Mod}_k \mathbf{P}$ : Modulo-Based Computation

$$\text{Mod}_k \mathbf{P} = \left\{ L \mid \left( \begin{array}{l} (\exists \text{ NPTM } N) (\forall x) [x \in L \iff \\ \#_{\text{acc}_N}(x) \not\equiv 0 \pmod{k}] \end{array} \right) \right\}.$$

$$\oplus \mathbf{P} = \text{Mod}_2 \mathbf{P}.$$

$\oplus \mathbf{P}$ , introduced independently by Papadimitriou and Zachos [PZ83] and Goldschlager and Parberry [GP86], captures the power of parity. Cai and Hemachandra [CH90] and Beigel, Gill, and Hertrampf [Her90,Bei91b,BG92] generalized the class to modulus other than two. There are oracles relative to which  $\oplus \mathbf{P}$  does not even contain NP, as shown by Torán [Tor91,Tor88]. Nonetheless, Toda [Tod91c] proved that  $\text{BPP}^{\oplus \mathbf{P}}$  contains the entire polynomial hierarchy, and Tarui [Tar93] shows that  $\mathbf{R} \cdot \mathbf{PP}$  contains the entire polynomial hierarchy (and even  $\mathbf{PP}^{\text{PH}}$ ).

## A.16 SpanP, OptP: Output-Cardinality and Optimization Function Classes

The counting class  $\# \mathbf{P}$  captures the notion of the number of accepting paths of NP machines. This class plays a very central role in complexity theory. However, it is not the only function class that plays a central role.

The function class OptP, introduced by Krentel ([Kre88], see also [BJY91]), seeks to capture the notion of maximizing over the set of output values of nondeterministic machines. Our model is as follows. We by convention say that any path that does not explicitly output a non-negative integer has implicitly output the integer 0. A function  $f$  is an OptP function if there is some such machine,  $N$ , for which, on each  $x$ ,  $f(x) = \max\{i \in \mathbb{N} \mid \text{some path of } N(x) \text{ has } i \text{ as its output}\}$ .

---

 **$\oplus P$ ,  $\text{Mod}_k P$  — Modulo-Based Computation**

Power

Modulo-based computation.

Definition

$$\begin{aligned} \text{Mod}_k P &= \{L \mid (\exists \text{ NPTM } N) (\forall x) [x \in L \iff \# \text{acc}_N(x) \not\equiv 0 \pmod{k}]\}. \\ \oplus P &= \text{Mod}_2 P. \end{aligned}$$

Alternate Definition

$$\text{Mod}_k P = \left\{ L \mid \begin{array}{l} \text{there is a polynomial-time predicate } P \text{ and a poly-} \\ \text{nomial } q \text{ such that, for all } x, x \in L \iff ||\{y \mid |y| \leq} \\ q(|x|) \wedge P(x, y)\}|| \not\equiv 0 \pmod{k} \end{array} \right\}.$$

Background

$\oplus P$  was introduced by Papadimitriou and Zachos [PZ83] and Goldschlager and Parberry [GP86].  $\text{Mod}_k P$  was introduced by Cai and Hemachandra ([CH90], see also [Her90, BG92]).

Complete Languages

$\oplus \text{SAT} = \{f \mid f \text{ has an even number of satisfying assignments}\}$  is complete for  $\oplus P$ . Analogous complete sets exist for  $\text{Mod}_k P$ .

Selected Facts and Theorems

1.  $\text{UP} \subseteq \text{SPP} \subseteq \oplus P \cap \text{C}_{=}P$ .
2.  $\oplus P^{\text{SPP}} = \oplus P^{\oplus P} = \oplus P^{\text{Few}P} = \oplus P$ . [PZ83, KSTT92, FFK94]
3. For any  $k$  that is a prime power,  $\text{Mod}_k P = \text{coMod}_k P$ . [BG92]
4. For any integer  $k > 1$ ,  $\text{Mod}_k P = \text{Mod}_{\pi(k)} P$ , where  $\pi(k)$  denotes the product of all primes that are divisors of  $k$ , e.g.,  $\pi(12) = 2 \cdot 3 = 6$ . In particular, for any  $k$  that is a prime power and  $i \geq 1$ ,  $\text{Mod}_{k^i} P = \text{Mod}_k P$ . ([Her90], see also [BG92])
5. For any  $k$ ,  $\text{Mod}_k P$  is closed under union. [Her90]
6.  $\oplus P^{\text{PH}} \subseteq \text{BPP}^{\oplus P}$ . (see Chap. 4)
7. There exists an oracle  $A$  relative to which  $P^A = \text{NP}^A = \text{PH}^A \neq \oplus P^A = \text{EXP}^A$ . [BM99]
8. For each  $k \geq 2$ , the existence of sparse  $\leq_{btt}^P$ -hard sets for  $\text{Mod}_k P$  implies  $P = \text{Mod}_k P$ . [OL93]
9. For each  $k \geq 2$ ,  $\text{Mod}_k P \subseteq \text{R}_{btt}^P(P\text{-sel}) \implies P = \text{Mod}_k P$ . [AA96, Ogi95b]
10. There exists an oracle  $A$  relative to which  $\oplus P \not\subseteq \text{PP}^{\text{PH}}$ .  
(see the text and Bibliographic Notes of Chap. 8)

Open Problem

- $\text{NP} \subseteq \oplus P \implies \text{PH} \subseteq \oplus P?$

---

**Fig. A.24**  $\oplus P$  and  $\text{Mod}_k P$ 


---



So,  $\#P$  focuses on the number of accepting paths of NP machines and OptP focuses on the largest output value of an NP machine. The class SpanP seeks to capture the functions giving the richness of the output value set. That is, a function  $f$  is in SpanP if there is some NP Turing Machine  $N$  such that, on each  $x$ ,  $f(x)$  is the cardinality of the set of strings output on the accepting paths of  $N$ . (By looking at only the accepting paths, we allow the possibility that on some inputs  $f$  may take on the value 0.) SpanP was introduced by Köbler, Schöning, and Torán ([KST89], see also [Köb89,Sch90]). SpanP is a quite flexible class; it is easy to see that SpanP contains both  $\#P$  and OptP.

## A.17 IP and MIP: Interactive Proof Classes

A verifier is a polynomial time-bounded probabilistic Turing machine  $V$  with a special state called query state and  $k$  special read/write tapes called communication tapes, where  $k \geq 1$ . Through the  $k$  communication tapes, the verifier interacts with  $k$  adversaries  $P_1, \dots, P_k$  called provers, which have unlimited computational power and can use randomness but cannot communicate among others. The interaction with provers is invoked when  $V$  enters query state. At that moment, for each  $i, 1 \leq i \leq k$ , the string held on the  $i$ th communication tape is sent to  $P_i$ . In the next move, for each  $i, 1 \leq i \leq k$ ,  $P_i$  supplies an answer, which depends on (1) the input to  $V$ , (2) the query, (3) the questions and answers passed through the tape so far, and (4)  $P_i$ 's probability distribution.

**Definition A.7** *Let  $k \geq 1$ . A language  $L$  has a  $k$ -prover interactive proof system if there exist a verifier with  $k$  communication tapes and  $k$  provers  $P_1, \dots, P_k$  such that for every  $x$ , the following two conditions (1) and (2), respectively called the correctness condition and the soundness condition, are met:*

1. *if  $x \in L$ , then  $V$  on  $x$  with  $P_1, \dots, P_k$  accepts with probability greater than  $\frac{3}{4}$  and*
2. *if  $x \notin L$ , then for any provers  $P'_1, \dots, P'_k$ ,  $V$  on  $x$  with  $P'_1, \dots, P'_k$  rejects with probability greater than  $\frac{3}{4}$ .*

IP (respectively, MIP) is the class of all languages  $L$  that have one-prover interactive proof systems (respectively,  $k$ -prover interactive proof systems for some  $k$ ).

We will sometimes omit the word “one-prover” but, by convention, it will be implicit.

---

**IP and MIP (Interactive Proof Systems)**

Power

Probabilistic verification.

Definition

$$\text{IP} = \left\{ L \left| \begin{array}{l} \text{there is a verifier } V \text{ and a prover } P \text{ such that for every } x, \\ \text{it holds that (a) if } x \in L, \text{ then the probability that } V \text{ on } x \\ \text{accepts with prover } P \text{ is greater than } \frac{3}{4} \text{ and (b) if } x \in L, \\ \text{then for any prover } P', \text{ the probability that } V \text{ on } x \text{ accepts} \\ \text{with prover } P' \text{ is less than } \frac{3}{4} \end{array} \right. \right\}.$$

$$\text{MIP} = \left\{ L \left| \begin{array}{l} \text{there exist some } k \geq 1, \text{ a } k\text{-communication-tape verifier} \\ V, \text{ and provers } P_1, \dots, P_k \text{ such that for every } x, \text{ it holds} \\ \text{that (a) if } x \in L, \text{ then the probability that } V \text{ on } x \text{ accepts} \\ \text{provers } P_1, \dots, P_k \text{ is greater than } \frac{3}{4}, \text{ and (b) if } x \in L, \\ \text{then for any provers } P'_1, \dots, P'_k, \text{ the probability that } V \text{ on} \\ x \text{ accepts with } P'_1, \dots, P'_k \text{ is less than } \frac{3}{4} \end{array} \right. \right\}.$$

Background

IP was studied independently by Babai [Bab85] and by Goldwasser, Micali, and Rackoff [GMR89], with different models and terminology (with Babai in particular defining the class AM, Arthur-Merlin). The difference between these two models is the treatment of the coin tosses of the verifier, which are sent to the prover in Babai's model and kept secret in Goldwasser, Micali, and Rackoff's model. Goldwasser and Sipser showed that these two models are equivalent [GS89]. MIP was introduced by Ben-Or, Goldwasser, Kilian, and Wigderson [BOGKW88].

Selected Facts and Theorems

1.  $\text{NP} \subseteq \text{IP}$ . [GMR89]
2.  $\text{P}^{\text{PP}} \subseteq \text{IP}$ . (see Chap. 6)
3.  $\text{IP} = \text{PSPACE}$ . (see Chap. 6)
4.  $\text{MIP} = \text{NEXP}$ . (see Chap. 6)
5. With probability one relative to a random oracle, IP and PSPACE differ. [CCG<sup>+</sup>94]

---

**Fig. A.25** IP and MIP

**A.18 PBP, SF, SSF: Branching Programs and Bottleneck Computation**

A width- $k$  branching program over variables  $x_1, \dots, x_n$  is a sequence  $P = \{(i_j, \mu_j^0, \mu_j^1)\}_{j=1}^m$  such that for each  $j, 1 \leq j \leq m$ , it holds that

1.  $1 \leq i_j \leq n$ , and
2.  $\mu_j^0$  and  $\mu_j^1$  are mappings of  $\{1, \dots, k\}$  to itself.

The triples  $(i_j, \mu_j^0, \mu_j^1)$  are called instructions. Given a bitstring  $x \in \Sigma^n$ , whose  $n$  bits we will refer to as  $x_1, x_2, \dots, x_n$  (i.e.,  $x$  is the concatenation  $x_1x_2 \cdots x_n$ ), the product of  $P$  with respect to  $x$ , denoted by  $P[x]$ , is

**PBP, SF, and SSF—Branching Programs, Bottleneck Turing Machines**

Power

Distributing computations into smaller tasks.

Definition

$$\begin{aligned}
k\text{-PBP} &= \left\{ L \mid \begin{array}{l} L \text{ is accepted by width-}k \text{ polynomial-size} \\ \text{branching programs} \end{array} \right\}. \\
\text{SF}_k &= \left\{ L \mid \begin{array}{l} L \text{ is accepted by a width-}k \text{ bottleneck Turing} \\ \text{machines} \end{array} \right\}. \\
\text{SSF}_k &= \left\{ L \mid \begin{array}{l} L \text{ is accepted by a width-}k \text{ symmetric bottle-} \\ \text{neck Turing machines} \end{array} \right\}. \\
\text{ProbabilisticSSF}_k &= \left\{ L \mid \begin{array}{l} L \text{ is accepted by a width-}k \text{ probabilistic sym-} \\ \text{metric bottleneck Turing machines} \end{array} \right\}.
\end{aligned}$$

Selected Facts and Theorems

1.  $\text{Nonuniform-NC}^1 = 5\text{-PBP} = \bigcup_{k \geq 2} k\text{-PBP}$ . (see Chap. 7)
2.  $\text{Nonuniform-NC}^1$  is the class of languages accepted by a family of polynomial-size NUDFA programs on some monoid. (see the text and Bibliographic Notes of Chap. 7)
3.  $\text{Nonuniform-AC}^0$  is the class of languages accepted by a family of polynomial-size NUDFA programs on some aperiodic monoid. (see the text and Bibliographic Notes of Chap. 7)
4.  $\text{PSPACE} = \text{SF}_5 = \bigcup_{k \geq 2} \text{SF}_k$ . (see Chap. 7)
5.  $\text{PH} \subseteq \text{SF}_4$ . ([Ogi94a], see also [Her97, Her00])
6.  $\text{SF}_4 \subseteq \text{BP} \cdot \oplus \text{P}^{\text{Mod}_3 \text{P} \oplus \text{P}^{\text{Mod}_3 \text{P} \oplus \text{P}}}$  ([Ogi94a], see also [Her97, Her00])
7. For  $k \geq 2$ , languages in  $\text{SSF}_k$  are many-one reducible to languages in  $\text{coMod}_k \text{P}$  by functions polynomial-time computable with at most  $k^k$  parallel queries to languages in NP. (see Chap. 7)
8.  $\text{ProbabilisticSSF}_2 = \text{NP}^{\text{PP}}$ . (see Chap. 7)

**Fig. A.26** PBP, SF, and SSF

$$\mu_m^{x_{im}} \circ \dots \circ \mu_1^{x_{i1}},$$

where the product is evaluated from the right to the left. That is, a triple  $(i_j, \mu_j^0, \mu_j^1)$  in  $P$  in effect says that  $\mu_j^0$  has to be multiplied into the product if the  $i_j$ th bit of  $x$  is a 0, and  $\mu_j^1$  has to be multiplied into the product otherwise. Program  $P$  accepts  $x$  if  $P[x]$  is a mapping that maps 1 to something else; i.e.,  $P[x](1) \neq 1$ . A language  $L$  is accepted by polynomial-size width- $k$  branching programs if there exists a family  $\{P_n\}_{n \geq 1}$  of width- $k$  branching programs such that (1) there exists a polynomial  $p$  such that for every  $n$ , the length of  $P_n$  is at most  $p(n)$  and (2) for every  $x$ ,  $x$  belongs to  $L$  if and only if  $P_{|x|}$  accepts  $x$ . The class of languages accepted by polynomial-size width- $k$  branching programs is denoted by  $k\text{-PBP}$ .

For  $k \geq 2$ , a width- $k$  bottleneck Turing machine is a polynomial time-bounded deterministic Turing machine  $M$  with an auxiliary input called the counter and a special device called the safe-storage, where the counter holds a binary integer of length  $p(|x|)$ , for some fixed polynomial  $p$  not dependent on  $x$ , and the safe-storage is a read/write cell that holds a number from the set  $\{1, \dots, k\}$ . For an input  $x$ , an auxiliary input  $y$ , and a value  $d \in \{1, \dots, k\}$ , let  $V(x, y, d)$  denote the value of the safe-storage when  $M$  finishes its computation. Collection  $V(x, v, d), 1 \leq d \leq k$ , can then be viewed as a mapping of  $\{1, \dots, k\}$  to itself, so, we will use  $f(x, y)$  to denote the mapping. A language  $L$  is accepted by  $M$  if for every  $x$ , it holds that (here we are showing an expansion that assumes that  $p(|x|) \geq 2$ , but the general case is similarly clear):

$$x \in L \iff f(x, 1^{p(|x|)}) \circ f(x, 1^{p(|x|)-1}0) \circ f(x, 1^{p(|x|)-2}01) \circ f(x, 1^{p(|x|)-2}00) \circ \dots \circ f(x, 0^{p(|x|)-2}10) \circ f(x, 0^{p(|x|)-1}1) \circ f(x, 0^{p(|x|)}) \text{ maps } 1 \text{ to } 1,$$

where the product is from right to left.  $\text{SF}_k$  is the class of languages accepted by bottleneck Turing machines of width  $k$ .

A bottleneck Turing machine  $M$  is symmetric if, for every  $x$  and every permutation  $\pi$  over  $\Sigma^{p(|x|)}$ , it holds that:

$$(f(x, 1^{p(|x|)}) \circ \dots \circ f(x, 0^{p(|x|)}))(1) = 1 \iff (f(x, \pi(1^{p(|x|)})) \circ \dots \circ f(x, \pi(0^{p(|x|)})))(1) = 1.$$

$\text{SSF}_k$  is the class of all languages accepted by a symmetric bottleneck Turing machine of width  $k$ .

A probabilistic width- $k$  symmetric bottleneck Turing machine is defined by endowing  $M$  the power of flipping coins to determine what to store in the storage-value, which turns function  $f$  into a probability distribution over the set of all mappings of  $\{1, \dots, k\}$  to itself. A language  $L$  is accepted by  $M$  if for every  $x$  and every permutation  $\pi$  over  $\Sigma^{p(|x|)}$ , it holds that:

$$x \in L \iff \Pr[(f(x, \pi(1^{p(|x|)})) \circ \dots \circ f(x, \pi(0^{p(|x|)})))(1) = 1] = \frac{1}{2}.$$

$\text{ProbabilisticSSF}_k$  is the class of all languages accepted by probabilistic width- $k$  symmetric bottleneck Turing machines.

Branching programs were introduced by Lee [Lee59]. For every  $k \geq 2$ ,  $k$ -PBP can be viewed as a class of those languages many-one reducible to languages accepted by  $k$ -state automata via nonuniform (the output depends only on the input length) functions each of whose output bits is one of: an input bit, the negation of an input bit, the constant 0, or the constant 1 (such reductions in general are studied by Skyum and Valiant [SV85]). Barrington [Bar89] showed that for every  $k \geq 5$ , polynomial-size width- $k$  branching programs capture nonuniform  $\text{NC}^1$  and for  $k = 2, 3, 4$ ,  $k$ -PBP are subclasses of languages accepted by polynomial-size constant-depth circuits with modulo 2 gates and modulo 3 gates.

Bottleneck Turing machines were introduced by Cai and Furst [CF91], who showed that  $SF_5 = PSPACE$ . The power of  $SF_k$  and its subclasses with  $k = 2, 3, 4$  were studied by Ogihara [Ogi94a] and Hertrampf ([Her97], see also [Her00]). Ogihara [Ogi94a], among other results, exactly classified  $SF_2$  as equaling the class  $\oplus OptP$ , which was originally defined by Hemachandra and Hoene [HH91b] to capture the notion of efficient implicit membership testing. Hertrampf ([Her97], see also [Her00]) expressed, using the notion of query-order-based classes ([HHW99], see also [HHH97]), exact classifications of the power of  $SF_3$  and  $SF_4$ .

Symmetric bottleneck Turing machines as well as probabilistic versions of them are introduced and studied by Hemaspaandra and Ogihara [HO97], and have been further investigated by Hertrampf ([Her99], see also [Her00]).



## B. A Rogues' Gallery of Reductions

If one can solve problem  $A$  using a black box that solves problem  $B$ , one can reasonably say that problem  $A$  is “not much harder than” problem  $B$ , the degree of the “not much” being linked to how powerful and extensive the use of  $B$  is. Thus, reductions provide a means of classifying the relative hardness of sets. If  $A$  reduces to  $B$  and  $B$  reduces to  $A$ , then we can reasonably say that  $A$  and  $B$  are of “about the same” hardness. Of course, the closeness of the relationship between  $A$  and  $B$  will again depend on how powerful the reduction is. The more computationally weak the reduction is, the stronger the claim we can make about the similarity of hardness of  $A$  and  $B$ . Over the years, a rich collection of reductions has been developed to aid in classifying the relative hardness of sets. In this chapter, we define the key reductions, mention some standard notational shorthands, and then present some comments about reductions and their relative powers. We also discuss a centrally important reduction, Cook’s reduction, which is the reduction that proves that SAT is NP-complete.

### B.1 Reduction Definitions: $\leq_m^p$ , $\leq_T^p$ , ...

$\leq_m^p$  (Many-one reductions)

$$A \leq_m^p B \iff (\exists f \in \text{FP})(\forall x)[x \in A \iff f(x) \in B].$$

$\leq_T^p$  (Turing reductions)

$$A \leq_T^p B \iff A \in \text{P}^B \text{ (see Sect. A.3).}$$

$\leq_{tt}^p$  (Truth-table reductions)

$$A \leq_{tt}^p B \iff (\exists g \in \text{FP})(\exists L \in \text{P})(\forall x)[(\exists \ell)(\exists z_1, z_2, \dots, z_\ell)[g(x) = z_1 \# z_2 \# \dots \# z_\ell \#] \wedge (x \in A \iff x \# \chi_B(z_1) \chi_B(z_2) \dots \chi_B(z_\ell) \in L)].$$

$\leq_{dtt}^p$  (Disjunctive truth-table reductions)

$$A \leq_{dtt}^p B \iff (\exists g \in \text{FP})(\forall x)[(\exists \ell)(\exists z_1, z_2, \dots, z_\ell)[g(x) = z_1 \# z_2 \# \dots \# z_\ell \#] \wedge (x \in A \iff z_1 \in B \vee z_2 \in B \vee \dots \vee z_\ell \in B)]. \text{ (By convention, when the machine asks no questions the input is rejected.)}$$

$\leq_{ctt}^p$  (Conjunctive truth-table reductions)

$$A \leq_{ctt}^p B \iff (\exists g \in \text{FP})(\forall x)[(\exists \ell)(\exists z_1, z_2, \dots, z_\ell)[g(x) = z_1 \# z_2 \# \dots \# z_\ell \#] \wedge (x \in A \iff z_1 \in B \wedge z_2 \in B \wedge \dots \wedge z_\ell \in B)]. \text{ (By convention, when the machine asks no questions the input is accepted.)}$$

$\leq_d^p$  (Disjunctive Turing reductions)

$A \leq_d^p B \iff A \leq_T^p B$  via some deterministic polynomial-time machine  $M$  that on each input accepts if and only if at least one oracle query is made and at least one query that  $M$  makes is in  $B$ . (Note in particular that when the machine asks no questions to its oracle, the input is rejected.)

$\leq_c^p$  (Conjunctive Turing reductions)

$A \leq_c^p B \iff A \leq_T^p B$  via some deterministic polynomial-time machine that on each input accepts if and only if all queries it makes to its oracle are in  $B$ . (Note in particular that when the machine asks no questions to its oracle, the input is accepted.)

$\leq_{pos}^p$  (Positive Turing reductions)

$A \leq_{pos}^p B \iff A \leq_T^p B$  via some deterministic Turing machine  $M$  that for some polynomial  $q$  runs for all oracles in time  $q(n)$  (where  $n$  is the input length) and that satisfies the additional property that:

$$(\forall C, D)[C \subseteq D \implies L(M^C) \subseteq L(M^D)].$$

$\leq_{locpos}^p$  (Locally positive Turing reductions)

$A \leq_{locpos}^p B \iff A \leq_T^p B$  via some deterministic polynomial-time machine  $M$  that has the following properties:

1.  $(\forall C : C \supseteq B)[L(M^C) \supseteq L(M^B)]$ , and
2.  $(\forall C : C \subseteq B)[L(M^C) \subseteq L(M^B)]$ .

$\leq_{f(n)-tt}^p$  ( $f(n)$ -truth-table reductions)

$A \leq_{f(n)-tt}^p B \iff$   
 $(\exists g \in \text{FP})(\exists L \in \text{P})(\forall x)[(\exists \ell : \ell \leq f(|x|))(\exists z_1, z_2, \dots, z_\ell)[g(x) =$   
 $z_1 \# z_2 \# \dots \# z_\ell \#] \wedge (x \in A \iff x \# \chi_B(z_1) \chi_B(z_2) \dots \chi_B(z_\ell) \in L)].$

$\leq_{btt}^p$  (Bounded truth-table reductions)

$A \leq_{btt}^p B \iff (\exists k)[A \leq_{k-tt}^p B]$ .

$\leq_{f(n)-T}^p$  ( $f(n)$ -Turing reductions)

$A \leq_{f(n)-T}^p B \iff A \leq_T^p B$  via some deterministic polynomial-time machine that on each input  $x$  makes at most  $f(|x|)$  oracle queries.

$\leq_m^L$  (Many-one logspace reductions)

$A \leq_m^L B \iff (\exists f : \text{function } f \text{ can be computed by a logspace machine})$   
 $(\forall x)[x \in A \iff f(x) \in B]$ .

$\leq_m^{conp}$  (Many-one coNP reductions)

$A \leq_m^{conp} B \iff A = B = \emptyset \vee (\exists f \in \text{NPMV})(\forall x)[x \in A \iff \emptyset \neq \text{set-}f(x) \subseteq B]$ .

$\leq_T^C$  (Generalized Turing reductions)

$A \leq_T^C B \iff A \in \mathcal{C}^B$ . Notes: This is defined only for classes  $\mathcal{C}$  for which relativization has been defined. Clearly,  $\leq_T^p \equiv \leq_T^p$ . By tradition, the notation  $\leq_T^{sn}$  denotes  $\leq_T^{\text{NP} \cap \text{coNP}}$ , i.e.,  $A \leq_T^{sn} B \iff A \in \text{NP}^B \cap \text{coNP}^B$ .

**Note on Combining Mechanisms and Bounds** Interpretation types (disjunctive, conjunctive, positive, etc.) and bounds on the number of queries are often combined in the natural way. For example,



$$A \leq_{f(n)\text{-ctt}}^p B \iff (\exists g \in \text{FP})(\forall x)[(\exists \ell : \ell \leq f(|x|))(\exists z_1, z_2, \dots, z_\ell)[g(x) = z_1 \# z_2 \# \dots \# z_\ell \#] \wedge (x \in A \iff z_1 \in B \wedge z_2 \in B \wedge \dots \wedge z_\ell \in B)]].$$

## B.2 Shorthands: R and E

**Definition B.1** For any text strings  $a$  and  $b$  for which  $\leq_a^b$  is a reduction type that has been defined:

1. For any set  $C$ ,  $R_a^b(C) = \{L \mid L \leq_a^b C\}$ .
2. For any class  $C$ ,  $R_a^b(C) = \{L \mid (\exists C \in C)[L \leq_a^b C]\}$ .

**Definition B.2** For any text strings  $a$  and  $b$  for which  $\leq_a^b$  is a reduction type that has been defined:

1. For any set  $C$ ,  $E_a^b(C) = \{L \mid L \leq_a^b C \wedge C \leq_a^b L\}$ .
2. For any class  $C$ ,  $E_a^b(C) = \{L \mid (\exists C \in C)[L \leq_a^b C \wedge C \leq_a^b L]\}$ .

## B.3 Facts about Reductions

**Proposition B.3** For any sets  $A$  and  $B$  neither of which is  $\emptyset$  or  $\Sigma^*$ , the following hold.

1.  $A \leq_m^L B \implies A \leq_m^p B \implies A \leq_{dtt}^p B \implies A \leq_{tt}^p B \implies A \leq_T^p B \implies A \leq_T^{sn} B$ ,  
 $B$ ,
2.  $(A \leq_c^p B \vee A \leq_d B) \implies A \leq_{pos}^p B \implies A \leq_{locpos}^p B \implies A \leq_T^p B$ , and
3.  $A \leq_{ctt}^p B \implies A \leq_m^{conp} B$ .
4. If  $B$  is sparse and  $A \leq_{btt}^p B$  then  $A \leq_{dtt}^p B$ .

All the above implications—except that  $A \leq_{btt}^p B \implies A \leq_{dtt}^p B$  if  $B$  is sparse—are immediately clear from the definitions.

It is not hard to see that  $\leq_c^p$  is equivalent to  $\leq_{ctt}^p$ , and that  $\leq_d^p$  is equivalent to  $\leq_{dtt}^p$ .

Many complexity classes respect reductions. For example, though  $R_{btt}^p(\text{NP}) = \text{NP} \iff \text{NP} = \text{coNP}$ , nonetheless  $\text{NP}$  is closed downward under  $\leq_{pos}^p$  reductions, i.e.,

$$\text{NP} = R_{pos}^p(\text{NP}).$$

Similarly,  $R_T^{\text{BPP}}(\text{BPP}) = \text{BPP}$  and  $R_T^{sn}(\text{NP} \cap \text{coNP}) = \text{NP} \cap \text{coNP}$ .

Some reductions are powerful enough to bridge the differences between seemingly—or absolutely—different classes. For example,  $\text{SPARSE} \supsetneq \text{TALLY}$ , but it can be shown that  $R_{ctt}^p(\text{SPARSE}) = R_{ctt}^p(\text{TALLY})$ . As another example,  $\text{E} \subsetneq \text{EXP}$ , but clearly (via padding)  $R_m^p(\text{E}) = R_m^p(\text{EXP}) = \text{EXP}$ . Finally, though we suspect that  $\text{NP} \neq \text{coNP}$ , clearly  $R_{1-T}^p(\text{NP}) = R_{1-T}^p(\text{coNP})$ .

## B.4 Circuit-Based Reductions: $\text{NC}^k$ and $\text{AC}^k$

### $\text{AC}^k$

For  $k \geq 0$ , a language  $L$  is  $\text{AC}^k$  reducible to a language  $A$  if there exists a family of polynomial-size  $\mathcal{O}(\log^k n)$  depth circuits with oracle gates, which that this family used with oracle  $A$  accepts  $L$ , where an oracle gate with  $m$  inputs contributes 1 to the depth. Furthermore,  $L$  is logspace-uniform (P-uniform)  $\text{AC}^k$  reducible to  $A$  if there is a logspace (polynomial-time) algorithm to compute the description of the circuit for  $\Sigma^n$  given  $1^n$ .

### $\text{NC}^k$

For  $k \geq 1$ , a language  $L$  is  $\text{NC}^k$  reducible to a language  $A$  if there exists a family of polynomial-size,  $\mathcal{O}(\log^k n)$  depth, bounded fan-in (all  $\wedge$  and  $\vee$  gates have in-degree two) circuits with oracle gates, such that this family used with oracle  $A$  accepts  $L$ , where an oracle gate with  $m$  inputs contributes  $\lceil \log m \rceil$  to the depth. Furthermore,  $L$  is logspace-uniform (P-uniform)  $\text{NC}^k$  reducible to  $A$  if there is a logspace (polynomial-time) algorithm to compute the description of the circuit for  $\Sigma^n$  given  $1^n$ .

## B.5 Bibliographic Notes

Ladner, Lynch, and Selman's seminal paper is the best source on polynomial-time reductions [LLS75]. Ladner and Lynch [LL76] is the best source on logspace reductions. Positive Turing reductions were introduced by Selman [Sel82b], and locally positive Turing reductions were introduced by Hemachandra and Jain [HJ91]. Many-one coNP reductions were introduced by Beigel, Chang, and Ogiwara [BCO93]. Strong nondeterministic Turing reductions were introduced by Selman ([Sel78], see also [Lon82]). Allender et al. [AHOW92] proved that if  $B$  is sparse then  $A \leq_{btt}^p B \implies A \leq_{dtt}^p B$ . Cook's reduction is due (not surprisingly) to Cook [Coo71], though we state it here (see Fig. A.3) in a known stronger form. Buhrman, Hemaspaandra, and Longpré [BHL95] proved that  $\text{SPARSE} \subseteq R_{ctt}^p(\text{TALLY})$ , from which  $R_{ctt}^p(\text{SPARSE}) = R_{ctt}^p(\text{TALLY})$  clearly follows.  $R_T^{\text{BPP}}(\text{BPP}) = \text{BPP}$  is due to Ko and Zachos [Ko82,Zac82]. The  $\text{AC}^0$  reductions were introduced by Chandra, Stockmeyer, and Vishkin [CSV84] as reductions among functions. Cook [Coo85] introduced the  $\text{NC}^1$  many-one reductions as reductions of among functions. Language versions of the  $\text{NC}^k$  reducibility as well as the  $\text{AC}^k$  reducibility were introduced by Wilson [Wil85,Wil90]. In general, for every  $k \geq 0$ ,  $\text{NC}^{k+1}$  reductions are as powerful as  $\text{AC}^k$ -reductions. Ogiwara [Ogi95a] showed for certain classes such as NP and  $\text{C=P}$  that, for each  $k \geq 0$ , the P-uniform  $\text{AC}^k$  reducibility closure and the P-uniform  $\text{NC}^{k+1}$  reducibility closure coincide.

# References

- [AA96] M. Agrawal and V. Arvind. Quasi-linear truth-table reductions to P-selective sets. *Theoretical Computer Science*, 158(1–2):361–370, 1996.
- [ABG00] A. Amir, R. Beigel, and W. Gasarch. Some connections between bounded query classes and non-uniform complexity. Technical Report TR00-024, Electronic Colloquium on Computational Complexity, <http://www.eccc.uni-trier.de/eccc/>, May 2000.
- [ABO84] M. Ajtai and M. Ben-Or. A theorem on probabilistic constant depth computations. In *Proceedings of the 16th ACM Symposium on Theory of Computing*, pages 471–474. ACM Press, April 1984.
- [ABO99] E. Allender, R. Beals, and M. Ogiwara. The complexity of matrix rank and feasible systems of linear equations. *Computational Complexity*, 8(2):99–126, 1999.
- [ACG<sup>+</sup>99] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and their Approximability Properties*. Springer-Verlag, 1999.
- [Adl78] L. Adleman. Two theorems on random polynomial time. In *Proceedings of the 19th IEEE Symposium on Foundations of Computer Science*, pages 75–83. IEEE Computer Society, October 1978.
- [AFF<sup>+</sup>01] J. Aspnes, D. Fischer, M. Fischer, M. Kao, and A. Kumar. Towards understanding the predictability of stock markets from the perspective of computational complexity. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 745–754, January 2001.
- [AFK89] M. Abadi, J. Feigenbaum, and J. Kilian. On hiding information from an oracle. *Journal of Computer and System Sciences*, 39(1):21–50, 1989.
- [AH92] L. Adleman and M. Huang. *Primality Testing and Abelian Varieties over Finite Fields*. Springer-Verlag *Lecture Notes in Mathematics* #1512, 1992.
- [AH94] E. Allender and U. Hertrampf. Depth reduction for circuits of unbounded fan-in. *Information and Computation*, 112(2):217–238, 1994.
- [AHH<sup>+</sup>93] V. Arvind, Y. Han, L. Hemachandra, J. Köbler, A. Lozano, M. Mundhenk, M. Ogiwara, U. Schöning, R. Silvestri, and T. Thierauf. Reductions to sets of low information content. In K. Ambos-Spies, S. Homer, and U. Schöning, editors, *Complexity Theory*, pages 1–45. Cambridge University Press, 1993.
- [AHOW92] E. Allender, L. Hemachandra, M. Ogiwara, and O. Watanabe. Relating equivalence and reducibility to sparse sets. *SIAM Journal on Computing*, 21(3):521–539, 1992.

- [AHU74] A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.
- [AIK84] A. Adachi, S. Iwata, and T. Kasai. Some combinatorial game problems require  $\Omega(n^k)$  time. *Journal of the ACM*, 31(2):361–376, 1984.
- [Ajt83] M. Ajtai.  $\Sigma_1^1$ -formulae on finite structures. *Annals of Pure and Applied Logic*, 24(1):1–48, 1983.
- [AK01] V. Arvind and J. Köbler. On pseudorandomness and resource-bounded measure. *Theoretical Computer Science*, 255(1–2):205–221, 2001.
- [AKS83] M. Ajtai, J. Komlós, and E. Szemerédi. Sorting in  $c \log n$  parallel steps. *Combinatorica*, 3(1):1–19, 1983.
- [AL97] S. Arora and C. Lund. Hardness of approximations. In D. Hochbaum, editor, *Approximation algorithms for NP-hard problems*, pages 399–456. PWS Publishing Company, 1997.
- [All86] E. Allender. The complexity of sparse sets in P. In *Proceedings of the 1st Structure in Complexity Theory Conference*, pages 1–11. Springer-Verlag *Lecture Notes in Computer Science #223*, June 1986.
- [All88] E. Allender. Isomorphisms and 1-L reductions. *Journal of Computer and System Sciences*, 36(6):336–350, 1988.
- [All89a] E. Allender. A note on the power of threshold circuits. In *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science*, pages 580–584. IEEE Computer Society Press, October/November 1989.
- [All89b] E. Allender. P-uniform circuit complexity. *Journal of the ACM*, 36(4):912–928, 1989.
- [All89c] E. Allender. Some consequences of the existence of pseudorandom generators. *Journal of Computer and System Sciences*, 39(1):101–124, 1989.
- [All90] E. Allender. Oracles versus proof techniques that do not relativize. In *Proceedings of the 1990 SIGAL International Symposium on Algorithms*, pages 39–52. Springer-Verlag *Lecture Notes in Computer Science #450*, August 1990.
- [All91] E. Allender. Limitations of the upward separation technique. *Mathematical Systems Theory*, 24(1):53–67, 1991.
- [ALM<sup>+</sup>98] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998.
- [Ang80] D. Angluin. On counting problems and the polynomial-time hierarchy. *Theoretical Computer Science*, 12(2):161–173, 1980.
- [AO96] E. Allender and M. Ogihara. Relationships among PL, #L, and the determinant. *RAIRO Theoretical Informatics and Applications*, 30(1):1–21, 1996.
- [APR83] L. Adleman, C. Pomerance, and R. Rumely. On distinguishing prime numbers from composite numbers. *Annals of Mathematics*, 117(1):173–206, 1983.
- [AR88] E. Allender and R. Rubinfeld. P-printable sets. *SIAM Journal on Computing*, 17(6):1193–1202, 1988.
- [Aro94] S. Arora. *Probabilistic Checking of Proofs and Hardness of Approximation Problems*. PhD thesis, University of California at Berkeley, 1994.
- [ARZ99] E. Allender, K. Reinhardt, and S. Zhou. Isolation, matching, and counting: Uniform and nonuniform upper bounds. *Journal of Computer and System Sciences*, 59(2):164–181, 1999.

- [AS98] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM*, 45(1):70–122, 1998.
- [ASV00] A. Ambainis, L. Schulman, and U. Vazirani. Computing with highly mixed states. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, pages 697–704. ACM Press, May 2000.
- [AW90] E. Allender and C. Wilson. Downward translations of equality. *Theoretical Computer Science*, 75(3):335–346, 1990.
- [Bab85] L. Babai. Trading group theory for randomness. In *Proceedings of the 17th ACM Symposium on Theory of Computing*, pages 421–429. ACM Press, May 1985.
- [Bab87] L. Babai. A random oracle separates PSPACE from the polynomial hierarchy. *Information Processing Letters*, 26(1):51–53, 1987.
- [Bab90] L. Babai. E-mail and the unexpected power of interaction. In *Proceedings of the 5th Structure in Complexity Theory Conference*, pages 30–44. IEEE Computer Society Press, July 1990.
- [Bar85] D. Barrington. Width-3 permutation branching programs. Technical Memorandum TM-293, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 1985.
- [Bar89] D. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC<sup>1</sup>. *Journal of Computer and System Sciences*, 38(1):150–164, 1989.
- [BBF98] R. Beigel, H. Buhrman, and L. Fortnow. NP might not be as easy as detecting unique solutions. In *Proceedings of the 30th ACM Symposium on Theory of Computing*, pages 203–208. ACM Press, May 1998.
- [BBS86] J. Balczár, R. Book, and U. Schöning. The polynomial-time hierarchy and sparse oracles. *Journal of the ACM*, 33(3):603–617, 1986.
- [BCD<sup>+</sup>89] A. Borodin, S. Cook, P. Dymond, W. Ruzzo, and M. Tompa. Two applications of inductive counting for complementation problems. *SIAM Journal on Computing*, 18(3):559–578, 1989. Erratum appears in the same journal, 18(6):1283.
- [BCG<sup>+</sup>96] N. Bshouty, R. Cleve, R. Gavalda, S. Kannan, and C. Tamon. Oracles and queries that are sufficient for exact learning. *Journal of Computer and System Sciences*, 52(3):421–433, 1996.
- [BCGT99] R. Beals, R. Chang, W. Gasarch, and J. Torán. On finding the number of graph automorphisms. *Chicago Journal of Theoretical Computer Science*, volume 1999, article 1, 1999.
- [BCKT94] N. Bshouty, R. Cleve, S. Kannan, and C. Tamon. Oracles and queries that are sufficient for exact learning. In *Proceedings of the 7th ACM Conference on Computational Learning Theory*, pages 130–139. ACM Press, July 1994.
- [BCO93] R. Beigel, R. Chang, and M. Ogiwara. A relationship between difference hierarchies and relativized polynomial hierarchies. *Mathematical Systems Theory*, 26(3):293–310, 1993.
- [BCP83] A. Borodin, S. Cook, and N. Pippenger. Parallel computation for well-endowed rings and space-bounded probabilistic machines. *Information and Control*, 58(1–3):113–136, 1983.
- [BCS92] D. Bovet, P. Crescenzi, and R. Silvestri. A uniform approach to define complexity classes. *Theoretical Computer Science*, 104(2):263–283, 1992.
- [BCS95] D. Bovet, P. Crescenzi, and R. Silvestri. Complexity classes and sparse oracles. *Journal of Computer and System Sciences*, 50(3):382–390, 1995.

- [BD76] A. Borodin and A. Demers. Some comments on functional self-reducibility and the NP hierarchy. Technical Report TR 76-284, Department of Computer Science, Cornell University, Ithaca, NY, July 1976.
- [BDFP86] A. Borodin, D. Dolev, F. Fich, and W. Paul. Bounds for width two branching programs. *SIAM Journal on Computing*, 15(2):549–560, 1986.
- [BDG95] J. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I*. EATCS Texts in Theoretical Computer Science. Springer-Verlag, 2nd edition, 1995.
- [BDHM92] G. Buntrock, C. Damm, U. Hertrampf, and C. Meinel. Structure and importance of logspace-MOD classes. *Mathematical Systems Theory*, 25(3):223–237, 1992.
- [Bea94] P. Beame. A switching lemma primer. Technical Report UW-CSE-95-07-01, Department of Computer Science and Engineering, Univisity of Washington, 1994.
- [Bei89] R. Beigel. On the relativized power of additional accepting paths. In *Proceedings of the 4th Structure in Complexity Theory Conference*, pages 216–224. IEEE Computer Society Press, June 1989.
- [Bei91a] R. Beigel. Bounded queries to SAT and the boolean hierarchy. *Theoretical Computer Science*, 84(2):199–223, 1991.
- [Bei91b] R. Beigel. Relativized counting classes: Relations among thresholds, parity, and mods. *Journal of Computer and System Sciences*, 42(1):76–96, 1991.
- [Bei94] R. Beigel. Perceptrons, PP, and the polynomial hierarchy. *Computational Complexity*, 4(4):339–349, 1994.
- [Bei97] R. Beigel. Closure properties of GapP and #P. In *Proceedings of the 5th Israeli Symposium on Theory of Computing and Systems*, pages 144–146. IEEE Computer Society Press, June 1997.
- [Bel96] M. Bellare. Proof checking and approximation: Towards tight results. *SIGACT News*, 27(1):2–13, 1996. Extended version available at: <http://www-cse.ucsd.edu/users/mihir/pcp.html>.
- [Ber76] L. Berman. On the structure of complete sets. In *Proceedings of the 17th IEEE Symposium on Foundations of Computer Science*, pages 76–80. IEEE Computer Society, October 1976.
- [Ber77] L. Berman. *Polynomial Reducibilities and Complete Sets*. PhD thesis, Cornell University, Ithaca, NY, 1977.
- [Ber78] P. Berman. Relationship between density and deterministic complexity of NP-complete languages. In *Proceedings of the 5th International Colloquium on Automata, Languages, and Programming*, pages 63–71. Springer-Verlag Lecture Notes in Computer Science #62, July 1978.
- [BF90] D. Beaver and J. Feigenbaum. Hiding instances in multioracle queries. In *Proceedings of the 7th Annual Symposium on Theoretical Aspects of Computer Science*, pages 37–48. Springer-Verlag Lecture Notes in Computer Science #415, February 1990.
- [BF91] L. Babai and L. Fortnow. Arithmetization: a new method in structural complexity theory. *Computational Complexity*, 1(1):41–66, 1991.
- [BF99] H. Buhrman and L. Fortnow. Two queries. *Journal of Computer and System Sciences*, 59(2):182–194, 1999.
- [BF00] R. Beigel and B. Fu. Circuits over PP and PL. *Journal of Computer and System Sciences*, 60(2):422–441, 2000.
- [BFH78] G. Brassard, S. Fortune, and J. Hopcroft. A note on cryptography and  $NP \cap coNP = P$ . Technical Report TR-338, Department of Computer Science, Cornell University, Ithaca, NY, April 1978.

- [BFL91] L. Babai, L. Fortnow, and C. Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1(1):3–40, 1991.
- [BFLS91] L. Babai, L. Fortnow, L. Levin, and M. Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the 23rd ACM Symposium on Theory of Computing*, pages 21–31. ACM Press, May 1991.
- [BFNW93] L. Babai, L. Fortnow, N. Nisan, and A. Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3(4):307–318, 1993.
- [BG81] C. Bennett and J. Gill. Relative to a random oracle  $A$ ,  $P^A \neq NP^A \neq coNP^A$  with probability 1. *SIAM Journal on Computing*, 10(1):96–113, 1981.
- [BG82] A. Blass and Y. Gurevich. On the unique satisfiability problem. *Information and Control*, 55(1–3):80–88, 1982.
- [BG92] R. Beigel and J. Gill. Counting classes: Thresholds, parity, mods, and fewness. *Theoretical Computer Science*, 103(1):3–23, 1992.
- [BG98] R. Beigel and J. Goldsmith. Downward separation fails catastrophically for limited nondeterminism classes. *SIAM Journal on Computing*, 27(5):1420–1429, 1998.
- [BGS75] T. Baker, J. Gill, and R. Solovay. Relativizations of the  $P=?NP$  question. *SIAM Journal on Computing*, 4(4):431–442, 1975.
- [BGS91] A. Bertoni, M. Goldwurm, and N. Sabadini. The complexity of computing the number of strings of given length in context-free languages. *Theoretical Computer Science*, 86(2):325–342, 1991.
- [BH77] L. Berman and J. Hartmanis. On isomorphisms and density of NP and other complete sets. *SIAM Journal on Computing*, 6(2):305–322, 1977.
- [BHHR99] A. Beygelzimer, L. Hemaspaandra, C. Homan, and J. Rothe. One-way functions in worst-case cryptography: Algebraic and security properties are on the house. *SIGACT News*, 30(4):25–40, 1999.
- [BHL95] H. Buhrman, E. Hemaspaandra, and L. Longpré. SPARSE reduces conjunctively to TALLY. *SIAM Journal on Computing*, 24(4):673–681, 1995.
- [BHW91] R. Beigel, L. Hemachandra, and G. Wechsung. Probabilistic polynomial time is closed under parity reductions. *Information Processing Letters*, 37(2):91–94, 1991.
- [BHZ87] R. Boppana, J. Håstad, and S. Zachos. Does co-NP have short interactive proofs? *Information Processing Letters*, 25(2):127–132, 1987.
- [BIP98] P. Beame, R. Impagliazzo, and T. Pitassi. Improved depth lower bounds for small distance connectivity. *Computational Complexity*, 7(4):325–345, 1998.
- [BIS90] D. Barrington, N. Immerman, and H. Straubing. On uniformity within  $NC^1$ . *Journal of Computer and System Sciences*, 41(3):274–306, 1990.
- [BJLR91] G. Buntrock, B. Jenner, K. Lange, and P. Rossmanith. Unambiguous and fewness for logarithmic space. In *Proceedings of the 8th Conference on Fundamentals of Computation Theory*, pages 168–179. Springer-Verlag Lecture Notes in Computer Science #529, September 1991.
- [BJY91] D. Bruschi, D. Joseph, and P. Young. A structural overview of NP optimization problems. *Algorithms Review*, 2(1):1–26, 1991.
- [BK95] M. Blum and S. Kannan. Designing programs that check their work. *Journal of the ACM*, 42(1):269–291, 1995.
- [BKS95] R. Beigel, M. Kummer, and F. Stephan. Approximable sets. *Information and Computation*, 120(2):304–314, 1995.

- [BL97] H. Burtschick and W. Lindner. On sets Turing reducible to P-selective sets. *Theory of Computing Systems*, 30(2):135–143, 1997.
- [BLR93] M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, 47(3):549–595, December 1993.
- [BLS84] R. Book, T. Long, and A. Selman. Quantitative relativizations of complexity classes. *SIAM Journal on Computing*, 13(3):461–487, 1984.
- [BLS85] R. Book, T. Long, and A. Selman. Qualitative relativizations of complexity classes. *Journal of Computer and System Sciences*, 30(3):395–413, 1985.
- [BM99] R. Beigel and A. Maciel. Circuit lower bounds collapse relativized complexity classes. In *Proceedings of the 14th Annual IEEE Conference on Computational Complexity*, pages 222–226. IEEE Computer Society Press, May 1999.
- [BOC92] M. Ben-Or and R. Cleve. Computing algebraic formulas using a constant number of registers. *SIAM Journal on Computing*, 21(1):54–58, 1992.
- [BOGKW88] M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson. Multi-prover interactive proof: How to remove intractability assumptions. In *Proceedings of the 20th ACM Symposium on Theory of Computing*, pages 113–131. ACM Press, May 1988.
- [Boo72] R. Book. On languages accepted in polynomial time. *SIAM Journal on Computing*, 1(4):281–287, 1972.
- [Boo74a] R. Book. Comparing complexity classes. *Journal of Computer and System Sciences*, 3(9):213–229, 1974.
- [Boo74b] R. Book. Tally languages and complexity classes. *Information and Control*, 26(2):186–193, 1974.
- [Boo94] R. Book. On collapsing the polynomial-time hierarchy. *Information Processing Letters*, 52(5):235–237, 1994.
- [Bor94] B. Borchert. *Predicate Classes, Promise Classes, and the Acceptance Power of Regular Languages*. PhD thesis, Mathematisches Institut, Universität Heidelberg, Heidelberg, Germany, 1994.
- [Bra79] G. Brassard. A note on the complexity of cryptography. *IEEE Transactions on Information Theory*, 25(2):232–233, 1979.
- [BRS91] R. Beigel, N. Reingold, and D. Spielman. The perceptron strikes back. In *Proceedings of the 6th Structure in Complexity Theory Conference*, pages 286–291. IEEE Computer Society Press, June/July 1991.
- [BRS95] R. Beigel, N. Reingold, and D. Spielman. PP is closed under intersection. *Journal of Computer and System Sciences*, 50(2):191–202, 1995.
- [Bru92] D. Brusch. Strong separations of the polynomial hierarchy with oracles: Constructive separations by immune and simple sets. *Theoretical Computer Science*, 102(2):215–252, 1992.
- [BS93] P. Blackburn and E. Spaan. A modal perspective on the computational complexity of attribute value grammar. *Journal of Logic, Language, and Information*, 2(2):129–169, 1993.
- [BS95] R. Beigel and H. Straubing. The power of local self-reductions. In *Proceedings of the 10th Structure in Complexity Theory Conference*, pages 277–285. IEEE Computer Society Press, June 1995.
- [BS00] B. Borchert and F. Stephan. Looking for an analogue of Rice’s Theorem in circuit complexity theory. *Mathematical Logic Quarterly*, 46(4):489–504, 2000.
- [BST90] D. Barrington, H. Straubing, and D. Thérien. Non-uniform automata over groups. *Information and Computation*, 89(2):109–132, 1990.



- [BST93] H. Buhrman, E. Spaan, and L. Torenvliet. Bounded reductions. In K. Ambos-Spies, S. Homer, and U. Schöning, editors, *Complexity Theory*, pages 83–99. Cambridge University Press, 1993.
- [BT88] D. Barrington and D. Thérien. Finite monoids and the fine structure of  $NC^1$ . *Journal of the ACM*, 35(4):941–952, 1988.
- [BT94] R. Beigel and J. Tarui. On ACC. *Computational Complexity*, 4(4):350–366, 1994.
- [BT96a] H. Buhrman and T. Thierauf. The complexity of generating and checking proofs of membership. In *Proceedings of the 13th Annual Symposium on Theoretical Aspects of Computer Science*, pages 75–86. Springer-Verlag *Lecture Notes in Computer Science #1046*, February 1996.
- [BT96b] H. Buhrman and L. Torenvliet. P-selective self-reducible sets: A new characterization of P. *Journal of Computer and System Sciences*, 53(2):210–217, 1996.
- [BTvEB93] H. Buhrman, L. Torenvliet, and P. van Emde Boas. Twenty questions to a P-selector. *Information Processing Letters*, 48(4):201–204, 1993.
- [BU98] C. Berg and S. Ulfberg. A lower bound for perceptrons and an oracle separation of the  $PP^H$  hierarchy. *Journal of Computer and System Sciences*, 56(3):263–271, 1998.
- [Cai89] J. Cai. With probability one, a random oracle separates PSPACE from the polynomial-time hierarchy. *Journal of Computer and System Sciences*, 38(1):68–85, 1989.
- [Cai01] J. Cai.  $S_2^P \subseteq ZPP^{NP}$ . In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*. IEEE Computer Society Press, October 2001. To appear.
- [Can96] R. Canetti. More on BPP and the polynomial-time hierarchy. *Information Processing Letters*, 57(5):237–241, 1996.
- [CCG<sup>+</sup>94] R. Chang, B. Chor, O. Goldreich, J. Hartmanis, J. Håstad, and D. Ranjan. The Random Oracle Hypothesis is false. *Journal of Computer and System Sciences*, 49(1):24–39, 1994.
- [CCHO01] J. Cai, V. Chakaravarthy, L. Hemaspaandra, and M. Ogihara. Some Karp–Lipton-type theorems based on  $S_2$ . Technical Report TR-759, Department of Computer Science, University of Rochester, Rochester, NY, September 2001.
- [CCL94] J. Cai, A. Condon, and R. Lipton. PSPACE is provable by two provers in one round. *Journal of Computer and System Sciences*, 48(1):183–193, 1994.
- [CF91] J. Cai and M. Furst. PSPACE survives constant-width bottlenecks. *International Journal of Foundations of Computer Science*, 2(1):67–76, 1991.
- [CFL85] A. Chandra, S. Fortune, and R. Lipton. Unbounded fan-in circuits and associative functions. *Journal of Computer and System Sciences*, 30(2):222–235, 1985.
- [CGH<sup>+</sup>88] J. Cai, T. Gundermann, J. Hartmanis, L. Hemachandra, V. Sewelson, K. Wagner, and G. Wechsung. The boolean hierarchy I: Structural properties. *SIAM Journal on Computing*, 17(6):1232–1252, 1988.
- [CGH<sup>+</sup>89] J. Cai, T. Gundermann, J. Hartmanis, L. Hemachandra, V. Sewelson, K. Wagner, and G. Wechsung. The boolean hierarchy II: Applications. *SIAM Journal on Computing*, 18(1):95–111, 1989.
- [CH89] J. Cai and L. Hemachandra. Enumerative counting is hard. *Information and Computation*, 82(1):34–44, 1989.

- [CH90] J. Cai and L. Hemachandra. On the power of parity polynomial time. *Mathematical Systems Theory*, 23(2):95–106, 1990.
- [CH91] J. Cai and L. Hemachandra. A note on enumerative counting. *Information Processing Letters*, 38(4):215–219, 1991.
- [Chu36] A. Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58(2):345–363, 1936.
- [Chu41] A. Church. *The Calculi of Lambda-Conversion*. Annals of Mathematics Studies #6. Princeton University Press, 1941.
- [CHW99] J. Cai, L. Hemaspaandra, and G. Wechsung. Robust reductions. *Theory of Computing Systems*, 32(6):625–647, 1999.
- [CK] P. Crescenzi and V. Kann. A compendium of NP optimization problems. <http://www.nada.kth.se/~viggo/problemist/compendium.html>.
- [CKS81] A. Chandra, D. Kozen, and L. Stockmeyer. Alternation. *Journal of ACM*, 26(1), 1981.
- [CLRS01] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press/McGraw Hill, second edition, 2001.
- [CMTV98] H. Caussinus, P. McKenzie, D. Thérien, and H. Vollmer. Nondeterministic  $NC^1$  computation. *Journal of Computer and System Sciences*, 57(2):200–212, 1998.
- [CNS95] J. Cai, A. Naik, and D. Sivakumar. On the existence of hard sparse sets under weak reductions. Technical Report 95-31, Department of Computer Science, State University of New York at Buffalo, Buffalo, NY, July 1995.
- [CNS96] J. Cai, A. Naik, and D. Sivakumar. On the existence of hard sparse sets under weak reductions. In *Proceedings of the 13th Annual Symposium on Theoretical Aspects of Computer Science*, pages 307–318. Springer-Verlag *Lecture Notes in Computer Science #1046*, February 1996.
- [CO97] J. Cai and M. Ogihara. Sparse sets versus complexity classes. In L. Hemaspaandra and A. Selman, editors, *Complexity Theory Retrospective II*, pages 53–80. Springer-Verlag, 1997.
- [Cob64] A. Cobham. The intrinsic computational difficulty of functions. In *Proceedings of the 1964 International Congress for Logic Methodology and Philosophy of Science*, pages 24–30. North Holland, 1964.
- [Coo71] S. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd ACM Symposium on Theory of Computing*, pages 151–158. ACM Press, May 1971.
- [Coo73] S. Cook. A hierarchy for nondeterministic time complexity. *Journal of Computer and System Sciences*, 7(4):343–353, 1973.
- [Coo85] S. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64(1–3):2–22, 1985.
- [CRS95] S. Chari, P. Rohatgi, and A. Srinivasan. Randomness-optimal unique element isolation, with applications to perfect matching and related problems. *SIAM Journal on Computing*, 24(5):1036–1050, 1995.
- [CS99] J. Cai and D. Sivakumar. Sparse hard sets for P: Resolution of a conjecture of Hartmanis. *Journal of Computer and System Sciences*, 58(2):280–296, 1999.
- [CSV84] A. Chandra, L. Stockmeyer, and U. Vishkin. Constant depth reducibility. *SIAM Journal on Computing*, 13(2):423–439, 1984.
- [Dam91] C. Damm.  $DET = L^{\#L}$ ? Informatik-Preprint 8, Fachbereich Informatik der Humboldt-Universität zu Berlin, 1991.
- [Dav58] M. Davis. *Computability and Unsolvability*. Dover, 1958.

- [DHHT94] D. Denny-Brown, Y. Han, L. Hemaspaandra, and L. Torenvliet. Semi-membership algorithms: Some recent advances. *SIGACT News*, 25(3):12–23, 1994.
- [DHK00] A. Durand, M. Hermann, and P. Kolaitis. Subtractive reductions and complete problems for counting complexity classes. In *Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science*, pages 323–332. Springer-Verlag *Lecture Notes in Computer Science #1893*, August/September 2000.
- [DL78] R. DeMillo and R. Lipton. A probabilistic remark on algebraic program testing. *Information Processing Letters*, 7(4):193–195, 1978.
- [DT90] J. Díaz and J. Torán. Classes of bounded nondeterminism. *Mathematical Systems Theory*, 23(1):21–32, 1990.
- [Edm65] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17(3):449–467, 1965.
- [EFF82] P. Erdős, P. Frankl, and Z. Füredi. Families of finite sets in which no set is covered by the union of two others. *Journal of Combinatorial Theory, Series A*, 33(2):158–166, 1982.
- [EFF85] P. Erdős, P. Frankl, and Z. Füredi. Families of finite sets in which no set is covered by the union of  $r$  others. *Israel Journal of Mathematics*, 51(1–2):79–89, 1985.
- [EH85] E. Emerson and J. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *Journal of Computer and System Sciences*, 30(1):1–24, 1985.
- [Fel68] W. Feller. *An Introduction to Probability Theory and Its Applications*. J. Wiley and Sons, 1968.
- [FFK94] S. Fenner, L. Fortnow, and S. Kurtz. Gap-definable counting classes. *Journal of Computer and System Sciences*, 48(1):116–148, 1994.
- [FFK96] S. Fenner, L. Fortnow, and S. Kurtz. The Isomorphism Conjecture holds relative to an oracle. *SIAM Journal on Computing*, 25(1):193–206, 1996.
- [FFL96] S. Fenner, L. Fortnow, and L. Li. Gap-definability as a closure property. *Information and Computation*, 130(1):1–17, 1996.
- [FGJ<sup>+</sup>78] A. Fraenkel, M. Garey, D. Johnson, T. Schaefer, and Y. Yesha. The complexity of checkers on an  $N \times N$  board—Preliminary report. In *Proceedings of the 19th IEEE Symposium on Foundations of Computer Science*, pages 55–64. IEEE Computer Society, October 1978.
- [FGL<sup>+</sup>96] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Interactive proofs and the hardness of approximating cliques. *Journal of the ACM*, 43(2):268–292, 1996.
- [FHL80] M. Furst, J. Hopcroft, and E. Luks. Polynomial-time algorithms for permutation groups. In *Proceedings of the 21st IEEE Symposium on Foundations of Computer Science*, pages 36–41. IEEE Computer Society, October 1980.
- [FHOS97] S. Fenner, S. Homer, M. Ogiwara, and A. Selman. Oracles that compute values. *SIAM Journal on Computing*, 26(4):1043–1065, 1997.
- [FHT97] S. Fischer, L. Hemaspaandra, and L. Torenvliet. Witness-isomorphic reductions and local search. In A. Sorbi, editor, *Complexity, Logic, and Recursion Theory*, pages 207–223. Marcel Dekker, Inc., 1997.
- [FK92] M. Fellows and N. Koblitz. Self-witnessing polynomial-time complexity and prime factorization. In *Proceedings of the 7th Structure in Complexity Theory Conference*, pages 107–110. IEEE Computer Society Press, June 1992.

- [FL79] M. Fischer and R. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194–211, 1979.
- [FL92] U. Feige and L. Lovász. Two-prover one-round proof systems: Their power and their problems. In *Proceedings of the 24th ACM Symposium on Theory of Computing*, pages 733–744. ACM Press, May 1992.
- [For79] S. Fortune. A note on sparse complete sets. *SIAM Journal on Computing*, 8(3):431–433, 1979.
- [For94] L. Fortnow. The role of relativization in complexity theory. *Bulletin of the EATCS*, 52:229–244, 1994.
- [For99] L. Fortnow. Relativized worlds with an infinite hierarchy. *Information Processing Letters*, 69(6):309–313, 1999.
- [FR74] M. Fischer and M. Rabin. Super-exponential complexity of Presburger arithmetic. In R. Karp, editor, *Complexity of Computation*, SIAM-AMS Proceedings 7: Proceedings of a Symposium in Applied Mathematics of the American Mathematical Society and the Society for Industrial and Applied Mathematics. American Mathematical Society, 1974.
- [FR96] L. Fortnow and N. Reingold. PP is closed under truth-table reductions. *Information and Computation*, 124(1):1–6, 1996.
- [FRS94] L. Fortnow, J. Rompel, and M. Sipser. On the power of multi-prover interactive protocols. *Theoretical Computer Science*, 134(2):545–557, 1994.
- [FSS84] M. Furst, J. Saxe, and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, 1984.
- [Gav95] R. Gavaldà. Bounding the complexity of advice functions. *Journal of Computer and System Sciences*, 50(3):468–475, 1995.
- [GB91] R. Gavaldà and J. Balcázar. Strong and robustly strong polynomial time reducibilities to sparse sets. *Theoretical Computer Science*, 88(1):1–14, 1991.
- [Gef94] V. Geffert. A hierarchy that does not collapse: Alternations in low level space. *RAIRO Theoretical Informatics and Applications*, 28(5):465–512, 1994.
- [GH96] J. Goldsmith and S. Homer. Scalability and the isomorphism problem. *Information Processing Letters*, 57(3):137–143, 1996.
- [GH00] C. Glaßer and L. Hemaspaandra. A moment of perfect clarity II: Consequences of sparse sets hard for NP with respect to weak reductions. *SIGACT News*, 31(4):39–51, 2000.
- [GHJY91] J. Goldsmith, L. Hemachandra, D. Joseph, and P. Young. Near-testable sets. *SIAM Journal on Computing*, 20(3):506–523, 1991.
- [GHK92] J. Goldsmith, L. Hemachandra, and K. Kunen. Polynomial-time compression. *Computational Complexity*, 2(1):18–39, 1992.
- [GHR95] R. Greenlaw, H. Hoover, and W. Ruzzo. *Limits to Parallel Computation: P-Completeness Theory*. Oxford University Press, 1995.
- [Gil77] J. Gill. Computational complexity of probabilistic Turing machines. *SIAM Journal on Computing*, 6(4):675–695, 1977.
- [GJ79] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.
- [GK99] S. Goldwasser and J. Kilian. Primality testing using elliptic curves. *Journal of the ACM*, 46(4):450–472, 1999.
- [GKR<sup>+</sup>95] F. Green, J. Köbler, K. Regan, T. Schwentick, and J. Torán. The power of the middle bit of a #P function. *Journal of Computer and System Sciences*, 50(4):456–467, 1995.

- [Gla00] C. Glaßer. Consequences of the existence of sparse sets hard for NP under a subclass of truth-table reductions. Technical Report TR 245, Institut für Informatik, Universität Würzburg, Würzburg, Germany, January 2000.
- [GLR<sup>+</sup>91] P. Gemmell, R. Lipton, R. Rubinfeld, M. Sudan, and A. Wigderson. Self-testing/correcting for polynomials and for approximate functions. In *Proceedings of the 23rd ACM Symposium on Theory of Computing*, pages 31–42. ACM Press, May 1991.
- [GLST98] V. Guruswami, D. Lewin, M. Sudan, and L. Trevisan. A tight characterization of NP with 3-query PCPs. In *Proceedings of the 39th IEEE Symposium on Foundations of Computer Science*, pages 8–17. IEEE Computer Society Press, November 1998.
- [GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(2):186–208, 1989.
- [GMW91] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(3):691–729, 1991.
- [GNW90] T. Gundermann, N. Nasser, and G. Wechsung. A survey on counting classes. In *Proceedings of the 5th Structure in Complexity Theory Conference*, pages 140–153. IEEE Computer Society Press, July 1990.
- [Göd31] K. Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, 38:173–198, 1931.
- [Gol01] O. Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, 2001.
- [GOR00] J. Goldsmith, M. Ogihara, and J. Rothe. Tally NP sets and easy census functions. *Information and Computation*, 158(1):29–52, 2000.
- [Got95] G. Gottlob. NP trees and Carnap’s modal logic. *Journal of the ACM*, 42(2):421–457, 1995.
- [GP86] L. Goldschlager and I. Parberry. On the construction of parallel computers from various bases of boolean functions. *Theoretical Computer Science*, 43(1):43–58, 1986.
- [Gre91] F. Green. An oracle separating  $\oplus P$  from  $PP^{PH}$ . *Information Processing Letters*, 37(3):149–153, 1991.
- [Gru99] J. Gruska. *Quantum Computing*. McGraw Hill, 1999.
- [GS88] J. Grollmann and A. Selman. Complexity measures for public-key cryptosystems. *SIAM Journal on Computing*, 17(2):309–335, 1988.
- [GS89] S. Goldwasser and M. Sipser. Private coins versus public coins in interactive proof systems. In S. Micali, editor, *Randomness and Computation*, pages 73–90. Advances in Computing Research #5, JAI Press Inc., 1989.
- [GS91] A. Goldberg and M. Sipser. Compression and ranking. *SIAM Journal on Computing*, 20(3):524–536, 1991.
- [Gup92] S. Gupta. On the closure of certain function classes under integer division by polynomially bounded functions. *Information Processing Letters*, 44(2):205–210, 1992.
- [Gup93] S. Gupta. On bounded probability operators and  $C=P$ . *Information Processing Letters*, 48(2):93–98, 1993.
- [Gup95] S. Gupta. Closure properties and witness reduction. *Journal of Computer and System Sciences*, 50(3):412–432, 1995.

- [Gur83]      Y. Gurevich. Algebras of feasible functions. In *Proceedings of the 24th IEEE Symposium on Foundations of Computer Science*, pages 210–214. IEEE Computer Society Press, November 1983.
- [GW93]      R. Gavaldà and O. Watanabe. On the computational complexity of small descriptions. *SIAM Journal on Computing*, 22(6):1257–1274, 1993.
- [GW96]      A. Gál and A. Wigderson. Boolean complexity classes versus their arithmetic analogs. *Random Structures and Algorithms*, 9(1–2):99–111, 1996.
- [GZ97]      O. Goldreich and D. Zuckerman. Another proof that  $BPP \subseteq PH$  (and more). Technical Report TR97-045, Electronic Colloquium on Computational Complexity, <http://www.eccc.uni-trier.de/eccc/>, October 1997.
- [Har78]      J. Hartmanis. *Feasible Computations and Provable Complexity Properties*. CBMS-NSF Regional Conference Series in Applied Mathematics #30. SIAM, 1978.
- [Har83]      J. Hartmanis. On sparse sets in  $NP-P$ . *Information Processing Letters*, 16(2):55–60, 1983.
- [Har85]      J. Hartmanis. Solvable problems with conflicting relativizations. *Bulletin of the EATCS*, 27:40–49, 1985.
- [Har89]      J. Hartmanis. Gödel, von Neumann, and the  $P=?NP$  problem. *Bulletin of the EATCS*, 38:101–107, 1989.
- [Har91]      J. Hartmanis. Notes on  $IP = PSPACE$ . Manuscript, 1991.
- [Hås87]      J. Håstad. *Computational Limitations of Small-Depth Circuits*. MIT Press, 1987.
- [Hås89]      J. Håstad. Almost optimal lower bounds for small depth circuits. In S. Micali, editor, *Randomness and Computation*, pages 143–170. Advances in Computing Research #5, JAI Press Inc., 1989.
- [HCC+92]      J. Hartmanis, R. Chang, S. Chari, D. Ranjan, and P. Rohatgi. Relativization: A revisionistic retrospective. *Bulletin of the EATCS*, 47:144–153, 1992.
- [HCRR90]      J. Hartmanis, R. Chang, D. Ranjan, and P. Rohatgi. Structural complexity theory: Recent surprises. In *Proceedings of the 2nd Scandinavian Workshop on Algorithm Theory*, pages 1–12. Springer-Verlag Lecture Notes in Computer Science #447, July 1990.
- [Hem89]      L. Hemachandra. The strong exponential hierarchy collapses. *Journal of Computer and System Sciences*, 39(3):299–322, 1989.
- [Hem94]      L. Hemaspaandra. The not-ready-for-prime-time conjectures. *SIGACT News*, 24(2):5–10, 1994.
- [Her90]      U. Hertrampf. Relations among MOD-classes. *Theoretical Computer Science*, 74(3):325–328, 1990.
- [Her97]      U. Hertrampf. Acceptance by transformation monoids (with an application to local self-reductions). In *Proceedings of the 12th Annual IEEE Conference on Computational Complexity*, pages 213–224. IEEE Computer Society Press, June 1997.
- [Her99]      U. Hertrampf. Generalized regular counting classes. In *Proceedings of the 24th International Symposium on Mathematical Foundations of Computer Science*, pages 419–429. Springer-Verlag Lecture Notes in Computer Science #1672, August 1999.
- [Her00]      U. Hertrampf. Algebraic acceptance mechanisms for polynomial-time machines. *SIGACT News*, 31(2):22–33, 2000.

- [HH74] J. Hartmanis and H. Hunt. The LBA problem and its importance in the theory of computing. In R. Karp, editor, *Complexity of Computation*, SIAM-AMS Proceedings 7: Proceedings of a Symposium in Applied Mathematics of the American Mathematical Society and the Society for Industrial and Applied Mathematics, pages 1–26. American Mathematical Society, 1974.
- [HH88a] J. Hartmanis and L. Hemachandra. Complexity classes without machines: On complete languages for UP. *Theoretical Computer Science*, 58(1–3):129–142, 1988.
- [HH88b] J. Hartmanis and L. Hemachandra. On sparse oracles separating feasible complexity classes. *Information Processing Letters*, 28(6):291–295, 1988.
- [HH90] J. Hartmanis and L. Hemachandra. Robust machines accept easy sets. *Theoretical Computer Science*, 74(2):217–226, 1990.
- [HH91a] J. Hartmanis and L. Hemachandra. One-way functions and the non-isomorphism of NP-complete sets. *Theoretical Computer Science*, 81(1):155–163, 1991.
- [HH91b] L. Hemachandra and A. Hoene. On sets with efficient implicit membership tests. *SIAM Journal on Computing*, 20(6):1148–1156, 1991.
- [HH96] Y. Han and L. Hemaspaandra. Pseudorandom generators and the frequency of simplicity. *Journal of Cryptology*, 9(4):251–261, 1996.
- [HHH97] E. Hemaspaandra, L. Hemaspaandra, and H. Hempel. An introduction to query order. *Bulletin of the EATCS*, 63:93–107, 1997.
- [HHH98] E. Hemaspaandra, L. Hemaspaandra, and H. Hempel. Downward collapse from a weaker hypothesis. In *Proceedings of the 6th Italian Conference on Theoretical Computer Science*, pages 253–264. World Scientific, November 1998.
- [HHH99a] E. Hemaspaandra, L. Hemaspaandra, and H. Hempel. A downward collapse within the polynomial hierarchy. *SIAM Journal on Computing*, 28(2):383–393, 1999.
- [HHH99b] E. Hemaspaandra, L. Hemaspaandra, and H. Hempel. Extending downward collapse from 1-versus-2 queries to  $j$ -versus- $j + 1$  queries. In *Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science*, pages 270–280. Springer-Verlag *Lecture Notes in Computer Science* #1563, March 1999.
- [HHN<sup>+</sup>95] L. Hemaspaandra, A. Hoene, A. Naik, M. Ogiwara, A. Selman, T. Thierauf, and J. Wang. Nondeterministically selective sets. *International Journal of Foundations of Computer Science*, 6(4):403–416, 1995.
- [HHO96] L. Hemaspaandra, A. Hoene, and M. Ogiwara. Reducibility classes of P-selective sets. *Theoretical Computer Science*, 155(2):447–457, 1996. Erratum appears in the same journal, 234(1–2):323.
- [HHR97] E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Exact analysis of Dodgson elections: Lewis Carroll’s 1876 voting system is complete for parallel access to NP. *Journal of the ACM*, 44(6):806–825, 1997.
- [HHSY91] L. Hemachandra, A. Hoene, D. Siefkes, and P. Young. On sets polynomially enumerable by iteration. *Theoretical Computer Science*, 80(2):203–226, 1991.
- [HHT97] Y. Han, L. Hemaspaandra, and T. Thierauf. Threshold computation and cryptographic security. *SIAM Journal on Computing*, 26(1):59–78, 1997.
- [HHW99] L. Hemaspaandra, H. Hempel, and G. Wechsung. Query order. *SIAM Journal on Computing*, 28(2):637–651, 1999.

- [HI85] J. Hartmanis and N. Immerman. On complete problems for  $NP \cap coNP$ . In *Proceedings of the 12th International Colloquium on Automata, Languages, and Programming*, pages 250–259. Springer-Verlag *Lecture Notes in Computer Science* #194, July 1985.
- [HILL99] J. Håstad, R. Impagliazzo, L. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.
- [HIS85] J. Hartmanis, N. Immerman, and V. Sewelson. Sparse sets in  $NP-P$ : EXPTIME versus NEXPTIME. *Information and Control*, 65(2–3):159–181, 1985.
- [HJ91] L. Hemachandra and S. Jain. On the limitations of locally robust positive reductions. *International Journal of Foundations of Computer Science*, 2(3):237–255, 1991.
- [HJ95a] L. Hemaspaandra and S. Jha. Defying upward and downward separation. *Information and Computation*, 121(1):1–13, 1995.
- [HJ95b] L. Hemaspaandra and Z. Jiang. P-selectivity: Intersections and indices. *Theoretical Computer Science*, 145(1–2):371–380, 1995.
- [HJRW97] L. Hemaspaandra, Z. Jiang, J. Rothe, and O. Watanabe. Polynomial-time multi-selectivity. *Journal of Universal Computer Science*, 3(3):197–229, 1997.
- [HJV93] L. Hemaspaandra, S. Jain, and N. Vereshchagin. Banishing robust Turing completeness. *International Journal of Foundations of Computer Science*, 4(3):245–265, 1993.
- [HKR93] S. Homer, S. Kurtz, and J. Royer. On 1-truth-table-hard languages. *Theoretical Computer Science*, 115(2):383–389, 1993.
- [HL94] S. Homer and L. Longpré. On reductions of NP sets to sparse sets. *Journal of Computer and System Sciences*, 48(2):324–336, 1994.
- [HLS<sup>+</sup>93] U. Hertrampf, C. Lautemann, T. Schwentick, H. Vollmer, and K. Wagner. On the power of polynomial time bit-reductions. In *Proceedings of the 8th Structure in Complexity Theory Conference*, pages 200–207. IEEE Computer Society Press, May 1993.
- [HM80] J. Hartmanis and S. Mahaney. An essay about research on sparse NP complete sets. In *Proceedings of the 9th Symposium on Mathematical Foundations of Computer Science*, pages 40–57. Springer-Verlag *Lecture Notes in Computer Science* #88, September 1980.
- [HNOS96a] E. Hemaspaandra, A. Naik, M. Ogihara, and A. Selman. P-selective sets and reducing search to decision vs. self-reducibility. *Journal of Computer and System Sciences*, 53(2):194–209, 1996.
- [HNOS96b] L. Hemaspaandra, A. Naik, M. Ogihara, and A. Selman. Computing solutions uniquely collapses the polynomial hierarchy. *SIAM Journal on Computing*, 25(4):697–708, 1996.
- [HNP98] L. Hemaspaandra, C. Nasipak, and K. Parkins. A note on linear-nondeterminism, linear-sized, Karp-Lipton advice for the P-selective sets. *Journal of Universal Computer Science*, 4(8):670–674, 1998.
- [HO93] L. Hemachandra and M. Ogiwara. Is  $\#P$  closed under subtraction? In G. Rozenberg and A. Salomaa, editors, *Current Trends in Theoretical Computer Science: Essays and Tutorials*, pages 523–536. World Scientific, 1993.
- [HO97] L. Hemaspaandra and M. Ogihara. Universally serializable computation. *Journal of Computer and System Sciences*, 55(3):547–560, 1997.
- [Hof82] C. Hoffmann. *Group-Theoretic Algorithms and Graph Isomorphism. Lecture Notes in Computer Science* #136. Springer-Verlag, 1982.



- [Hom00] C. Homan. Low ambiguity in strong, total, associative, one-way functions. Technical Report TR-734, Department of Computer Science, University of Rochester, Rochester, NY, August 2000.
- [Hop81] J. Hopcroft. Recent directions in algorithmic research. In *Proceedings 5th GI Conference on Theoretical Computer Science*, pages 123–134. Springer-Verlag *Lecture Notes in Computer Science #104*, March 1981.
- [HOW92] L. Hemachandra, M. Ogiwara, and O. Watanabe. How hard are sparse sets? In *Proceedings of the 7th Structure in Complexity Theory Conference*, pages 222–238. IEEE Computer Society Press, June 1992.
- [HOW00] L. Hemaspaandra, M. Ogiwara, and G. Wechsung. Reducing the number of solutions of NP functions. In *Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science*, pages 394–404. Springer-Verlag *Lecture Notes in Computer Science #1893*, August/September 2000.
- [HPR01] L. Hemaspaandra, K. Pasanen, and J. Rothe. If  $P \neq NP$  then some strongly noninvertible functions are invertible. In *Proceedings of the 13th International Symposium on Fundamentals of Computation Theory*. Springer-Verlag *Lecture Notes in Computer Science*, August 2001. To appear.
- [HR90] L. Hemachandra and S. Rudich. On the complexity of ranking. *Journal of Computer and System Sciences*, 41(2):251–271, 1990.
- [HR97] L. Hemaspaandra and J. Rothe. Unambiguous computation: Boolean hierarchies and sparse Turing-complete sets. *SIAM Journal on Computing*, 26(3):634–653, 1997.
- [HR98] E. Hemaspaandra and J. Rothe. Recognizing when greed can approximate maximum independent sets is complete for parallel access to NP. *Information Processing Letters*, 65(3):151–156, 1998.
- [HR99] L. Hemaspaandra and J. Rothe. Creating strong, total, commutative, associative one-way functions from any one-way function in complexity theory. *Journal of Computer and System Sciences*, 58(3):648–659, 1999.
- [HR00] L. Hemaspaandra and J. Rothe. A second step towards complexity-theoretic analogs of Rice’s Theorem. *Theoretical Computer Science*, 244(1–2):205–217, 2000.
- [HRV00] U. Hertrampf, S. Reith, and H. Vollmer. A note on closure properties of logspace MOD classes. *Information Processing Letters*, 75(3):91–93, 2000.
- [HRZ95] L. Hemaspaandra, A. Ramachandran, and M. Zimand. Worlds to die for. *SIGACT News*, 26(4):5–15, 1995.
- [HS65] J. Hartmanis and R. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117(5):285–306, 1965.
- [HT] L. Hemaspaandra and M. Thakur. Rice’s theorem for polynomial-ambiguity computation. In preparation.
- [HT96] L. Hemaspaandra and L. Torenvliet. Optimal advice. *Theoretical Computer Science*, 154(2):367–377, 1996.
- [HT00] T. Hoang and T. Thierauf. The complexity of verifying the characteristic polynomial and testing similarity. In *Proceedings of the 15th Annual IEEE Conference on Computational Complexity*, pages 87–95. IEEE Computer Society Press, July 2000.
- [HU79] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

- [Huy84] D. Huynh. Deciding the inequivalence of context-free grammars with 1-letter terminal alphabet is  $\Sigma_2^P$ -complete. *Theoretical Computer Science*, 33(2-3):305-326, 1984.
- [Huy90] D. Huynh. The complexity of ranking simple languages. *Mathematical Systems Theory*, 23(1):1-20, 1990.
- [HVW95] U. Hertrampf, H. Vollmer, and K. Wagner. On the power of number-theoretic operations with respect to counting. In *Proceedings of the 10th Structure in Complexity Theory Conference*, pages 299-314. IEEE Computer Society Press, June 1995.
- [HW79] G. Hardy and E. Wright. *An Introduction to the Theory of Numbers*. Oxford University Press, 5th edition, 1979.
- [HW01] J. Håstad and A. Wigderson. Simple analysis of graph tests for linearity and PCP. In *Proceedings of the 16th Annual IEEE Conference on Computational Complexity*, pages 244-254. IEEE Computer Society Press, June 2001.
- [HY84] J. Hartmanis and Y. Yesha. Computation times of NP sets of different densities. *Theoretical Computer Science*, 34(1-2):17-32, 1984.
- [HZ93] L. Hemaspaandra and M. Zimand. Strong forms of balanced immunity. Technical Report TR-480, Department of Computer Science, University of Rochester, Rochester, NY, December 1993. Revised, May 1994.
- [HZZ96] L. Hemaspaandra, M. Zaki, and M. Zimand. Polynomial-time semi-rankable sets. In *Journal of Computing and Information*, 2(1), Special Issue: *Proceedings of the 8th International Conference on Computing and Information*, pages 50-67, 1996. CD-ROM ISSN 1201-8511/V2/#1.
- [IK94] S. Iwata and T. Kasai. The Othello game on an  $n \times n$  board is PSPACE-complete. *Theoretical Computer Science*, 123(2):329-340, 1994.
- [IM89] N. Immerman and S. Mahaney. Relativizing relativized computations. *Theoretical Computer Science*, 68(3):267-276, 1989.
- [Imm88] N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17(5):935-938, 1988.
- [IT89] R. Impagliazzo and G. Tardos. Decision versus search problems in super-polynomial time. In *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science*, pages 222-227. IEEE Computer Society Press, October/November 1989.
- [IW97] R. Impagliazzo and A. Wigderson.  $P = BPP$  if  $E$  requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, pages 220-229. ACM Press, May 1997.
- [JL76] N. Jones and W. Lasser. Complete problems for deterministic polynomial time. *Theoretical Computer Science*, 3(1):105-118, 1976.
- [JMT96] B. Jenner, P. McKenzie, and D. Thérien. Logspace and logtime leaf languages. *Information and Computation*, 129(1):21-33, 1996.
- [Joc68] C. Jockusch. Semirecursive sets and positive reducibility. *Transactions of the AMS*, 131(2):420-436, 1968.
- [Jun85] H. Jung. On probabilistic time and space. In *Proceedings of the 12th International Colloquium on Automata, Languages, and Programming*, pages 281-291. Springer-Verlag *Lecture Notes in Computer Science* #194, July 1985.
- [JY85] D. Joseph and P. Young. Some remarks on witness functions for non-polynomial and non-complete sets in NP. *Theoretical Computer Science*, 39(2-3):225-237, 1985.

- [Kad89] J. Kadin.  $P^{NP[\log n]}$  and sparse Turing-complete sets for NP. *Journal of Computer and System Sciences*, 39(3):282–298, 1989.
- [KAI79] T. Kasai, A. Adachi, and S. Iwata. Classes of pebble games and complete problems. *SIAM Journal on Computing*, 8(4):574–587, 1979.
- [Käm91] J. Kämper. Non-uniform proof systems: A new framework to describe non-uniform and probabilistic complexity classes. *Theoretical Computer Science*, 85(2):305–331, 1991.
- [Kar72] R. Karp. Reducibilities among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103, 1972.
- [KF80] C. Kintala and P. Fisher. Refining nondeterminism in relativized polynomial-time bounded computations. *SIAM Journal on Computing*, 9(1):46–53, 1980.
- [KL80] R. Karp and R. Lipton. Some connections between nonuniform and uniform complexity classes. In *Proceedings of the 12th ACM Symposium on Theory of Computing*, pages 302–309. ACM Press, April 1980. An extended version has also appeared as: Turing machines that take advice, *L'Enseignement Mathématique*, 2nd series, 28, 1982, pages 191–209.
- [KLD86] K. Ko, T. Long, and D. Du. On one-way functions and polynomial-time isomorphisms. *Theoretical Computer Science*, 47(3):263–276, 1986.
- [Kle52] S. Kleene. *Introduction to Metamathematics*. D. van Nostrand Company, Inc., 1952.
- [KMR88] S. Kurtz, S. Mahaney, and J. Royer. Collapsing degrees. *Journal of Computer and System Sciences*, 37(2):247–268, 1988.
- [KMR95] S. Kurtz, S. Mahaney, and J. Royer. The Isomorphism Conjecture fails relative to a random oracle. *Journal of the ACM*, 42(2):401–420, 1995.
- [Ko82] K. Ko. Some observations on the probabilistic algorithms and NP-hard problems. *Information Processing Letters*, 14(1):39–43, 1982.
- [Ko83] K. Ko. On self-reducibility and weak P-selectivity. *Journal of Computer and System Sciences*, 26(2):209–221, 1983.
- [Ko85] K. Ko. On some natural complete operators. *Theoretical Computer Science*, 37(1):1–30, 1985.
- [Ko89] K. Ko. Relativized polynomial time hierarchies having exactly  $k$  levels. *SIAM Journal on Computing*, 18(2):392–408, 1989.
- [Köb89] J. Köbler. *Strukturelle Komplexität von Anzahlproblemen*. PhD thesis, University of Stuttgart, Stuttgart, Germany, 1989.
- [Köb94] J. Köbler. Locating P/poly optimally in the extended low hierarchy. *Theoretical Computer Science*, 134(2):263–285, 1994.
- [Köb95] J. Köbler. On the structure of low sets. In *Proceedings of the 10th Structure in Complexity Theory Conference*, pages 246–261. IEEE Computer Society Press, June 1995.
- [Kos00] S. Kosub. On NP-partitions over posets with an application to reducing the set of solutions of NP problems. In *Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science*, pages 467–476. Springer-Verlag *Lecture Notes in Computer Science* #1893, August/September 2000.
- [Koz92] D. Kozen. *The Design and Analysis of Algorithms*. Springer-Verlag, 1992.
- [Kre88] M. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36(3):490–509, 1988.

- [KS85] K. Ko and U. Schöning. On circuit-size complexity and the low hierarchy in NP. *SIAM Journal on Computing*, 14(1):41–51, 1985.
- [KS97] J. Köbler and U. Schöning. High sets for NP. In D. Zu and K. Ko, editors, *Advances in Algorithms, Languages, and Complexity*, pages 139–156. Kluwer Academic Publishers, 1997.
- [KST89] J. Köbler, U. Schöning, and J. Torán. On counting and approximation. *Acta Informatica*, 26(4):363–379, 1989.
- [KSTT92] J. Köbler, U. Schöning, S. Toda, and J. Torán. Turing machines with few accepting computations and low sets for PP. *Journal of Computer and System Sciences*, 44(2):272–286, 1992.
- [KVVY93] R. Kannan, H. Venkateswaran, V. Vinay, and A. Yao. A circuit-based proof of Toda’s Theorem. *Information and Computation*, 104(2):271–276, 1993.
- [KW98] J. Köbler and O. Watanabe. New collapse consequences of NP having small circuits. *SIAM Journal on Computing*, 28(1):311–324, 1998.
- [Lad75a] R. Ladner. The circuit value problem is log space complete for P. *SIGACT News*, 7(1):18–20, 1975.
- [Lad75b] R. Ladner. On the structure of polynomial time reducibility. *Journal of the ACM*, 22(1):155–171, 1975.
- [Lan53] H. Landau. On dominance relations and the structure of animal societies, III: The condition for score structure. *Bulletin of Mathematical Biophysics*, 15(2):143–148, 1953.
- [Lan87] S. Lang. *Linear Algebra*. Springer-Verlag, 3rd edition, 1987.
- [Lee59] C. Lee. Representation of switching circuits by binary-decision programs. *Bell Systems Technical Journal*, 38(4):985–1000, 1959.
- [Lev75] L. Levin. Universal sequential search problems. *Problems of Information Transmission*, 9(3):265–266, 1975. March 1975 translation into English of Russian article originally published in 1973.
- [LFKN92] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM*, 39(4):859–868, 1992.
- [Lip91] R. Lipton. New directions in testing. In J. Feigenbaum and M. Merritt, editors, *Distributed Computing and Cryptography*, pages 191–202. DIMACS series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1991.
- [LL76] R. Ladner and N. Lynch. Relativization of questions about log space computability. *Mathematical Systems Theory*, 10(1):19–32, 1976.
- [LLS75] R. Ladner, N. Lynch, and A. Selman. A comparison of polynomial time reducibilities. *Theoretical Computer Science*, 1(2):103–124, 1975.
- [Lon78] T. Long. *On Some Polynomial Time Reducibilities*. PhD thesis, Purdue University, Lafayette, IN, 1978.
- [Lon82] T. Long. Strong nondeterministic polynomial-time reducibilities. *Theoretical Computer Science*, 21(1):1–25, 1982.
- [LR96] M. Liśkiewicz and R. Reischuk. The sublogarithmic alternating space world. *SIAM Journal on Computing*, 25(4):828–861, 1996.
- [LR97] M. Liśkiewicz and R. Reischuk. Computing with sublogarithmic space. In L. Hemaspaandra and A. Selman, editors, *Complexity Theory Retrospective II*, pages 197–224. Springer-Verlag, 1997.
- [LS80] D. Lichtenstein and M. Sipser. GO is polynomial-space hard. *Journal of the ACM*, 27(2):393–401, April 1980.
- [LS86] T. Long and A. Selman. Relativizing complexity classes with sparse oracles. *Journal of the ACM*, 33(3):618–627, 1986.

- [LS97] D. Lapidot and A. Shamir. Fully parallelized multi-prover protocols for NEXP-time. *Journal of Computer and System Sciences*, 54(2):215–220, 1997.
- [LSH65] P. Lewis, R. Stearns, and J. Hartmanis. Memory bounds for recognition of context-free and context-sensitive languages. In *Proceedings of the 6th IEEE Symposium on Switching Circuit Theory and Logical Design*, pages 191–202, 1965.
- [Lub96] M. Luby. *Pseudorandomness and Cryptographic Applications*. Princeton University Press, 1996.
- [Lup61] O. Lupanov. Implementing the algebra of logic functions in terms of constant-depth formulas in the basis  $+$ ,  $*$ ,  $-$ . *Soviet Physics Doklady*, 6(2):474–479, 1961.
- [Mah82] S. Mahaney. Sparse complete sets for NP: Solution of a conjecture of Berman and Hartmanis. *Journal of Computer and System Sciences*, 25(2):130–143, 1982.
- [Mah86] S. Mahaney. Sparse sets and reducibilities. In R. Book, editor, *Studies in Complexity Theory*, pages 63–118. John Wiley and Sons, 1986.
- [Mah89] S. Mahaney. The Isomorphism Conjecture and sparse sets. In J. Hartmanis, editor, *Computational Complexity Theory*, pages 18–46. American Mathematical Society, 1989. Proceedings of Symposia in Applied Mathematics #38.
- [Mas76] W. Masek. A fast algorithm for the string editing problem and decision graph complexity. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, May 1976.
- [MGLA00] M. Mundhenk, J. Goldsmith, C. Lusena, and E. Allender. Complexity of finite-horizon Markov decision process problems. *Journal of the ACM*, 47(4):681–720, July 2000.
- [MP79] A. Meyer and M. Paterson. With what frequency are apparently intractable problems difficult? Technical Report MIT/LCS/TM-126, Laboratory for Computer Science, MIT, Cambridge, MA, 1979.
- [MP88] M. Minsky and S. Papert. *Perceptrons*. MIT Press, 1988.
- [MS72] A. Meyer and L. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *Proceedings of the 13th IEEE Symposium on Switching and Automata Theory*, pages 125–129, October 1972.
- [MVV87] K. Mulmuley, U. Vazirani, and V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.
- [MY85] S. Mahaney and P. Young. Reductions among polynomial isomorphism types. *Theoretical Computer Science*, 39(2–3):207–224, 1985.
- [New64] D. Newman. Rational approximation to  $|\alpha|$ . *Michigan Mathematics Journal*, 11(1):11–14, 1964.
- [Nic00] A. Nickelsen. *Polynomial-Time Partial Information Classes*. PhD thesis, Technische Universität Berlin, Berlin, Germany, 2000.
- [Nis94] N. Nisan.  $RL \subseteq SC$ . *Computational Complexity*, 4(1):1–11, 1994.
- [NRRS98] A. Naik, J. Rogers, J. Royer, and A. Selman. A hierarchy based on output multiplicity. *Theoretical Computer Science*, 207(1):131–157, 1998.
- [NRS95] A. Naik, K. Regan, and D. Sivakumar. On quasilinear-time complexity theory. *Theoretical Computer Science*, 148(2):325–349, 1995.
- [NS99] A. Naik and A. Selman. Adaptive versus nonadaptive queries to NP and P-selective sets. *Computational Complexity*, 8(2):169–187, 1999.

- [NW94] N. Nisan and A. Wigderson. Hardness vs. randomness. *Journal of Computer and System Sciences*, 49(2):149–167, 1994.
- [Ogi92] M. Ogiwara. A characterization of  $P^{C=P}$ . *IEICE Transactions on Communications, Electronics, Information, and Systems*, E75-D(1):44–49, 1992.
- [Ogi94a] M. Ogiwara. On serializable languages. *International Journal of Foundations of Computer Science*, 5(3–4):303–318, 1994.
- [Ogi94b] M. Ogiwara. Generalized theorems on the relationships among reducibility notions to certain complexity classes. *Mathematical Systems Theory*, 27(3):189–200, 1994.
- [Ogi95a] M. Ogiwara. Equivalence of  $NC^k$  and  $AC^{k-1}$  closures of NP and other classes. *Information and Computation*, 120(1):55–58, 1995.
- [Ogi95b] M. Ogiwara. Polynomial-time membership comparable sets. *SIAM Journal on Computing*, 24(5):1068–1081, 1995.
- [Ogi96a] M. Ogiwara. Functions computable with limited access to NP. *Information Processing Letters*, 58(1):35–38, 1996.
- [Ogi96b] M. Ogiwara. Sparse hard sets for P yield space-efficient algorithms. *Chicago Journal of Theoretical Computer Science*, volume 1996, article 2, 1996.
- [Ogi98] M. Ogiwara. The PL hierarchy collapses. *SIAM Journal on Computing*, 27(5):1430–1437, 1998.
- [OH93] M. Ogiwara and L. Hemachandra. A complexity theory for feasible closure properties. *Journal of Computer and System Sciences*, 46(3):295–325, 1993.
- [OL93] M. Ogiwara and A. Lozano. On sparse hard sets for counting classes. *Theoretical Computer Science*, 112(2):255–275, 1993.
- [OTTW96] M. Ogiwara, T. Thierauf, S. Toda, and O. Watanabe. On closure properties of #P in the context of  $P^{O\#P}$ . *Journal of Computer and System Sciences*, 53(2):171–179, 1996.
- [OW91] M. Ogiwara and O. Watanabe. On polynomial-time bounded truth-table reducibility of NP sets to sparse sets. *SIAM Journal on Computing*, 20(3):471–483, June 1991.
- [Pap84] C. Papadimitriou. On the complexity of unique solutions. *Journal of the ACM*, 31(2):392–400, 1984.
- [Pap94] C. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
- [PBI93] T. Pitassi, P. Beame, and R. Impagliazzo. Exponential lower bounds for the Pigeonhole Principle. *Computational Complexity*, 3(2):97–140, 1993.
- [Pip79] N. Pippenger. On simultaneous resource bounds. In *Proceedings of the 20th IEEE Symposium on Foundations of Computer Science*, pages 307–311. IEEE Computer Society, October 1979.
- [Pos46] E. Post. A variant of a recursively unsolvable problem. *Bulletin of the AMS*, 52(4):264–268, 1946.
- [Pra75] V. Pratt. Every prime has a succinct certificate. *SIAM Journal on Computing*, 4(3):214–220, 1975.
- [Pra79] V. Pratt. Models of program logics. In *Proceedings of the 20th IEEE Symposium on Foundations of Computer Science*, pages 115–122. IEEE Computer Society, October 1979.
- [PS94] S. Paturi and M. Saks. Approximating threshold circuits by rational functions. *Information and Computation*, 112(2):257–272, 1994.

- [PZ83] C. Papadimitriou and S. Zachos. Two remarks on the power of counting. In *Proceedings 6th GI Conference on Theoretical Computer Science*, pages 269–276. Springer-Verlag *Lecture Notes in Computer Science* #145, January 1983.
- [RA99] K. Reinhardt and E. Allender. Making nondeterminism unambiguous. *SIAM Journal on Computing*, 29(4):1118–1131, 1999.
- [Rab76] M. Rabin. Probabilistic algorithms. In J. Traub, editor, *Algorithms and Complexity*, pages 21–39. Academic Press, 1976.
- [Rac82] C. Rackoff. Relativized questions involving probabilistic algorithms. *Journal of the ACM*, 29(1):261–268, 1982.
- [Reg85] K. Regan. Enumeration problems. Manuscript, 1982; revised, 1985.
- [Reg01] K. Regan, July 2001. Personal communication.
- [Rog67] H. Rogers, Jr. *The Theory of Recursive Functions and Effective Computability*. McGraw-Hill, 1967.
- [Rog97] J. Rogers. The Isomorphism Conjecture holds and one-way functions exist relative to an oracle. *Journal of Computer and System Sciences*, 54(3):412–423, 1997.
- [RRW94] R. Rao, J. Rothe, and O. Watanabe. Upward separation for FewP and related classes. *Information Processing Letters*, 52(4):175–180, 1994. Corrigendum appears in the same journal, 74(1–2):89.
- [RS81] C. Rackoff and J. Seiferas. Limitations on separating nondeterministic complexity classes. *SIAM Journal on Computing*, 10(4):742–745, 1981.
- [RS93] M. Rabi and A. Sherman. Associative one-way functions: A new paradigm for secret-key agreement and digital signatures. Technical Report CS-TR-3183/UMIACS-TR-93-124, Department of Computer Science, University of Maryland, College Park, Maryland, 1993.
- [RS96] R. Rubinfeld and M. Sudan. Robust characterization of polynomials. *SIAM Journal on Computing*, 25(2):252–271, 1996.
- [RS97] M. Rabi and A. Sherman. An observation on associative one-way functions in complexity theory. *Information Processing Letters*, 64(5):239–244, 1997.
- [RS98] A. Russell and R. Sundaram. Symmetric alternation captures BPP. *Computational Complexity*, 7(2):152–162, 1998.
- [RST84] W. Ruzzo, J. Simon, and M. Tompa. Space-bounded hierarchies and probabilistic computations. *Journal of Computer and System Sciences*, 28(2):216–230, 1984.
- [Rub88] R. Rubinfeld. *Structural Complexity Classes of Sparse Sets: Intractability, Data Compression and Printability*. PhD thesis, Northeastern University, Boston, MA, August 1988.
- [Rus85] D. Russo. *Structural Properties of Complexity Classes*. PhD thesis, University of California at Santa Barbara, Santa Barbara, CA, 1985.
- [Ruz80] W. Ruzzo. Tree-size bounded alternation. *Journal of Computer and System Sciences*, 21(2):218–235, 1980.
- [Ruz81] W. Ruzzo. On uniform circuit complexity. *Journal of Computer and System Sciences*, 22(3):365–383, 1981.
- [Sav70] W. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.
- [Sav72] J. Savage. Computational work and time on finite machines. *Journal of the ACM*, 19(4):660–674, 1972.
- [Sch80] J. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27(4):701–717, 1980.

- [Sch83]      U. Schöning. A low and a high hierarchy within NP. *Journal of Computer and System Sciences*, 27(1):14–28, 1983.
- [Sch86a]    U. Schöning. Complete sets and closeness to complexity classes. *Mathematical Systems Theory*, 19(1):29–42, 1986.
- [Sch86b]    U. Schöning. *Complexity and Structure*. Springer Verlag *Lecture Notes in Computer Science* #211, 1986.
- [Sch88]      U. Schöning. Graph isomorphism is in the low hierarchy. *Journal of Computer and System Sciences*, 37(3):312–323, 1988.
- [Sch89]      U. Schöning. Probabilistic complexity classes and lowness. *Journal of Computer and System Sciences*, 39(1):84–100, 1989.
- [Sch90]      U. Schöning. The power of counting. In A. Selman, editor, *Complexity Theory Retrospective*, pages 204–223. Springer-Verlag, 1990.
- [Sch99]      M. Schaefer. Deciding the VC-dimension is  $\Sigma_3^P$ -complete. *Journal of Computer and System Sciences*, 58(1):177–182, 1999.
- [Sch00]      M. Schaefer. Deciding the VC-dimension is  $\Sigma_3^P$ -complete II. Technical Report TR00-006, School of CTI, DePaul University, Chicago, IL, 2000.
- [Sch01a]    M. Schaefer. Completeness in the polynomial-time hierarchy. Technical Report TR01-009, School of CTI, DePaul University, Chicago, IL, July 2001.
- [Sch01b]    M. Schaefer. Graph Ramsey theory and the polynomial hierarchy. *Journal of Computer and System Sciences*, 62(2):290–322, 2001.
- [Scu62]      V. Scully. *The Earth, the Temple, and the Gods*. Yale University Press, 1962.
- [Sel74]      A. Selman. On the structure of NP. *Notices of the AMS*, 21(5):A-498, 1974. Erratum in the same journal, 21(6):310.
- [Sel78]      A. Selman. Polynomial time enumeration reducibility. *SIAM Journal on Computing*, 7(4):440–457, 1978.
- [Sel79]      A. Selman. P-selective sets, tally languages, and the behavior of polynomial time reducibilities on NP. *Mathematical Systems Theory*, 13(1):55–65, 1979.
- [Sel82a]    A. Selman. Analogues of semirecursive sets and effective reducibilities to the study of NP complexity. *Information and Control*, 52(1):36–51, 1982.
- [Sel82b]    A. Selman. Reductions on NP and P-selective sets. *Theoretical Computer Science*, 19(3):287–304, 1982.
- [Sel92]      A. Selman. A survey of one-way functions in complexity theory. *Mathematical Systems Theory*, 25(3):203–221, 1992.
- [Sel94]      A. Selman. A taxonomy of complexity classes of functions. *Journal of Computer and System Sciences*, 48(2):357–381, 1994.
- [SFM78]    J. Seiferas, M. Fischer, and A. Meyer. Separating nondeterministic time complexity classes. *Journal of the ACM*, 25(1):146–167, 1978.
- [Sha92]    A. Shamir.  $IP = PSPACE$ . *Journal of the ACM*, 39(4):869–877, 1992.
- [She86]    A. Sherman. *Cryptology and VLSI (a Two-Part Dissertation)*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1986. Available as Technical Report MIT/LCS/TR-381.
- [SHL65]    R. Stearns, J. Hartmanis, and P. Lewis. Hierarchies of memory limited computations. In *Proceedings of the 6th IEEE Symposium on Switching Circuit Theory and Logical Design*, pages 179–190, 1965.
- [Sim75]    J. Simon. *On Some Central Problems in Computational Complexity*. PhD thesis, Cornell University, Ithaca, N.Y., January 1975. Available as Cornell Department of Computer Science Technical Report TR75-224.



- [Sim77a] I. Simon. *On Some Subrecursive Reducibilities*. PhD thesis, Stanford University, Palo Alto, CA, April 1977. Available as Stanford University Computer Science Department Technical Report STAN-CS-77-608.
- [Sim77b] J. Simon. On the difference between one and many. In *Proceedings of the 4th International Colloquium on Automata, Languages, and Programming*, pages 480–491. Springer-Verlag *Lecture Notes in Computer Science* #52, July 1977.
- [Sip82] M. Sipser. On relativization and the existence of complete sets. In *Proceedings of the 9th International Colloquium on Automata, Languages, and Programming*, pages 523–531. Springer-Verlag *Lecture Notes in Computer Science* #140, July 1982.
- [Sip83] M. Sipser. Borel sets and circuit complexity. In *Proceedings of the 15th ACM Symposium on Theory of Computing*, pages 61–69. ACM Press, April 1983.
- [Sip92] M. Sipser. The history and status of the P versus NP question. In *Proceedings of the 24th ACM Symposium on Theory of Computing*, pages 603–618. ACM Press, May 1992.
- [Siv00] D. Sivakumar, May 6, 2000. Personal communication.
- [SL94] M. Sheu and T. Long. The extended low hierarchy is an infinite hierarchy. *SIAM Journal on Computing*, 23(3):488–509, 1994.
- [SS83] J. Schwartz and M. Sharir. On the piano movers' problem III: Coordinating the motion of several independent bodies: The special case of circular bodies moving amidst polygonal barriers. *International Journal of Robotics Research*, 2(3):46–75, 1983.
- [ST98] M. Santha and S. Tan. Verifying the determinant. *Computational Complexity*, 7(2):128–151, 1998.
- [ST00] A. Samorodnitsky and L. Trevisan. A PCP characterization of NP with optimal amortized query complexity. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, pages 191–199. ACM Press, May 2000.
- [Sto76] L. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.
- [Sto85] L. Stockmeyer. On approximation algorithms for #P. *SIAM Journal on Computing*, 14(4):849–861, 1985.
- [Sud78] I. Sudborough. On the tape complexity of deterministic context-free languages. *Journal of the ACM*, 25(3):405–414, 1978.
- [Sud92] M. Sudan. *Efficient Checking of Polynomials and Proofs and the Hardness of Approximation Problems*. PhD thesis, University of California at Berkeley, 1992. Also appears as an ACM Distinguished Thesis, Lecture Notes in Computer Science, Volume 1001, Springer-Verlag, 1996.
- [SV85] S. Skyum and L. Valiant. A complexity theory based on boolean algebra. *Journal of the ACM*, 32(2):484–502, 1985.
- [Sze88] R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26(3):279–284, 1988.
- [Tar93] J. Tarui. Randomized polynomials,  $AC^0$  functions, and the polynomial hierarchy. *Theoretical Computer Science*, 113(1):167–183, 1993.
- [Th 81] D. Th rien. Classification of finite monoids: The language approach. *Theoretical Computer Science*, 14(2):195–208, 1981.
- [TO92] S. Toda and M. Ogiwara. Counting classes are at least as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 21(2):316–328, 1992.

- [Tod91a] S. Toda. Counting problems computationally equivalent to computing the determinant. Technical Report CSIM 91-07, Department of Computer Science, University of Electro-Communications, Tokyo, Japan, May 1991.
- [Tod91b] S. Toda. On polynomial-time truth-table reducibilities of intractable sets to P-selective sets. *Mathematical Systems Theory*, 24(2):69–82, 1991.
- [Tod91c] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991.
- [Tod94] S. Toda. Simple characterizations of  $P(\#P)$  and complete problems. *Journal of Computer and System Sciences*, 49(1):1–17, 1994.
- [Tor88] J. Torán. *Structural Properties of the Counting Hierarchies*. PhD thesis, Universitat Politècnica de Catalunya, Barcelona, Spain, 1988.
- [Tor91] J. Torán. Complexity classes defined by counting quantifiers. *Journal of the ACM*, 38(3):753–774, 1991.
- [Tur36] A. Turing. On computable numbers, with an application to the *Entscheidungsproblem*. *Proceedings of the London Mathematical Society*, series 2(42):230–265, 1936. Correction appears in the same journal as series 2(43):544–546.
- [Ukk83] E. Ukkonen. Two results on polynomial time truth-table reductions to sparse sets. *SIAM Journal on Computing*, 12(3):580–587, 1983.
- [Uma98] C. Umans. The minimum equivalent DNF problem and shortest implicants. In *Proceedings of the 39th IEEE Symposium on Foundations of Computer Science*, pages 556–563. IEEE Computer Society Press, November 1998. To appear in *Journal of Computer and System Sciences*.
- [Val76] L. Valiant. The relative complexity of checking and evaluating. *Information Processing Letters*, 5(1):20–23, 1976.
- [Val79a] L. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.
- [Val79b] L. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.
- [Val92] L. Valiant. Why is boolean complexity theory difficult? In M. Paterson, editor, *Boolean Function Complexity*, pages 84–94. London Mathematical Society, Lecture Note Series 169, Cambridge University Press, 1992.
- [vBGR93] B. von Braunmühl, R. Gengler, and R. Rettinger. The alternation hierarchy for sublogarithmic space is infinite. *Computational Complexity*, 3(3):207–230, 1993.
- [vBGR94] B. von Braunmühl, R. Gengler, and R. Rettinger. The alternation hierarchy for machines with sublogarithmic space is infinite. In *Proceedings of the 11th Annual Symposium on Theoretical Aspects of Computer Science*, pages 85–96. Springer-Verlag *Lecture Notes in Computer Science* #775, February 1994.
- [vdW70] B. van der Waerden. *Algebra*. Frederick Ungar Publishing Company, 1970. Translated by F. Blum and H. Schulenberg.
- [Ven91] H. Venkateswaran. Properties that characterize LOGCFL. *Journal of Computer and System Sciences*, 43(2):380–404, 1991.
- [Ver94] N. Vereshchagin. Relativizable and nonrelativizable theorems in the polynomial theory of algorithms. *Russian Academy of Sciences—Izvestiya–Mathematics*, 42(2):261–298, 1994.
- [Vin91] V. Vinay. Counting auxiliary pushdown automata and semi-unbounded arithmetic circuits. In *Proceedings of the 6th Structure in*

- Complexity Theory Conference*, pages 270–284. IEEE Computer Society Press, June/July 1991.
- [vM96] D. van Melkebeek. Reducing P to a sparse set using a constant number of queries collapses P to L. In *Proceedings of the 11th Annual IEEE Conference on Computational Complexity*, pages 88–96. IEEE Computer Society Press, May 1996.
- [vM97] D. van Melkebeek, 1997. Personal communication.
- [vMO97] D. van Melkebeek and M. Ogiwara. Sparse hard sets for P. In D. Du and K. Ko, editors, *Advances in Algorithms, Languages, and Complexity*, pages 191–208. Kluwer Academic Publishers, 1997.
- [VV85] U. Vazirani and V. Vazirani. Random polynomial time is equal to slightly-random polynomial time. In *Proceedings of the 26th IEEE Symposium on Foundations of Computer Science*, pages 417–428. IEEE Computer Society Press, October 1985.
- [VV86] L. Valiant and V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47(3):85–93, 1986.
- [Wag86] K. Wagner. The complexity of combinatorial problems with succinct input representations. *Acta Informatica*, 23(3):325–356, 1986.
- [Wag87] K. Wagner. More complicated questions about maxima and minima, and some closures of NP. *Theoretical Computer Science*, 51(1–2):53–80, 1987.
- [Wag90] K. Wagner. Bounded query classes. *SIAM Journal on Computing*, 19(5):833–846, 1990.
- [Wag93] K. Wagner. The alternation hierarchy for sublogarithmic space: An exciting race to STACS '93 (editorial note). In *Proceedings of the 10th Annual Symposium on Theoretical Aspects of Computer Science*, pages 25–27. Springer-Verlag *Lecture Notes in Computer Science* #665, February 1993.
- [Wan95] J. Wang. Some results on selectivity and self-reducibility. *Information Processing Letters*, 55(2):81–87, 1995.
- [Wat88] O. Watanabe. On hardness of one-way functions. *Information Processing Letters*, 27(3):151–157, 1988.
- [Wes96] D. West. *Introduction to Graph Theory*. Prentice Hall, 1996.
- [Wig94] A. Wigderson.  $NL/poly \subseteq \oplus L/poly$ . In *Proceedings of the 9th Structure in Complexity Theory Conference*, pages 59–62. IEEE Computer Society Press, June/July 1994.
- [Wil85] C. Wilson. Relativized circuit complexity. *Journal of Computer and System Sciences*, 31(2):169–181, 1985.
- [Wil90] C. Wilson. On the decomposability of NC and AC. *SIAM Journal on Computing*, 19(2):384–396, 1990.
- [Wra76] C. Wrathall. Complete sets and the polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):23–33, 1976.
- [WT93] O. Watanabe and S. Toda. Structural analysis of the complexity of inverse functions. *Mathematical Systems Theory*, 26(2):203–214, 1993.
- [Yao85] A. Yao. Separating the polynomial-time hierarchy by oracles. In *Proceedings of the 26th IEEE Symposium on Foundations of Computer Science*, pages 1–10. IEEE Computer Society Press, October 1985.
- [Yao90] A. Yao. On ACC and threshold circuits. In *Proceedings of the 31st IEEE Symposium on Foundations of Computer Science*, pages 619–627. IEEE Computer Society Press, October 1990.
- [Yap83] C. Yap. Some consequences of non-uniform conditions on uniform classes. *Theoretical Computer Science*, 26(3):287–300, 1983.

- [Yes83]      Y. Yesha. On certain polynomial-time truth-table reducibilities of complete sets to sparse sets. *SIAM Journal on Computing*, 12(3):411–425, 1983.
- [You92]      P. Young. How reductions to sparse sets collapse the polynomial-time hierarchy: A primer. *SIGACT News*, 23, 1992. Part I (#3, pages 107–117), Part II (#4, pages 83–94), and Corrigendum to Part I (#4, page 94).
- [Zac82]      S. Zachos. Robustness of probabilistic complexity classes under definitional perturbations. *Information and Computation*, 54(3):143–154, 1982.
- [Zac88]      S. Zachos. Probabilistic quantifiers and games. *Journal of Computer and System Sciences*, 36(3):433–451, 1988.
- [Zan91]      V. Zankó. #P-completeness via many-one reductions. *International Journal of Foundations of Computer Science*, 2(1):76–82, 1991.
- [ZF87]      S. Zachos and M. Furer. Probabilistic quantifiers vs. distrustful adversaries. In *Proceedings of the 7th Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 443–455. Springer-Verlag *Lecture Notes in Computer Science* #287, December 1987.
- [ZH86]      S. Zachos and H. Heller. A decisive characterization of BPP. *Information and Control*, 69(1–3):125–135, 1986.
- [Zim98]      M. Zimand. On the size of classes with weak membership properties. *Theoretical Computer Science*, 209(1–2):225–235, 1998.
- [Zip79]      R. Zippel. Probabilistic algorithms for sparse polynomials. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, pages 216–226. Springer-Verlag *Lecture Notes in Computer Science* #72, 1979.

# Index

- $\Delta_k^p$  29, 72, 194, 271–273, 286, *see* hierarchy, polynomial, *see* set,  $\Delta_2^p$ -complete
- $\Pi_k^p$  20, 25, 28, 271–273, *see* hierarchy, polynomial
- $\Pi_k^{p,C}$  213, 214
- $\Sigma_k^p$  19–21, 25, 28, 73, 108, 271–273, *see* hierarchy, polynomial, *see* set,  $\Sigma_2^p$ -complete
- $\Sigma_k^{p,C}$  213–216, 228
- $\Theta_k^p$  18–20, 27, 29, 268, 271–273, 293, *see* hierarchy, polynomial, *see* set,  $\Theta_2^p$ -complete
- closure properties *see* closure, of  $\Theta_k^p \dots$
- $\leq_T^C$  306
- $\leq_m^L$  83, 265, 306, 307, *see* reduction, logspace many-one
- $\leq_{\text{randomized}}$  70, 72, 268, *see* reduction, randomized
- $\leq_m^{\text{comp}}$  241, 306, *see* reduction, coNP-many-one
- $\leq_{NP \cap \text{coNP}}$  306
- $\leq_T^p, A$  60, 284, 285
- $\leq_m^p$  1, 18–20, 22, 60, 182, 268, 269, 305, 306, *see* reduction, polynomial-time Turing
- closure of  $C=P$  downward under *see* closure, of  $C=P$  downward under  $\leq_T^p$
- closure of  $\text{co}C=P$  downward under *see* closure, of  $\text{co}C=P$  downward under  $\leq_T^p$
- closure of  $\oplus P$  downward under *see* closure, of  $\oplus P$  downward under  $\leq_T^p$
- $\leq_{\text{btt}}^p$  1, 8–11, 26, 268, 296, 298, 306–308, *see* reduction, polynomial-time bounded-truth-table
- closure of  $P$  downward under *see* closure, of  $P$  downward under  $\leq_{\text{btt}}^p$
- closure of  $PP$  downward under *see* closure, of  $PP$  downward under  $\leq_{\text{btt}}^p$
- $\leq_c^p$  306, *see* reduction, polynomial-time conjunctive Turing
- $\leq_{\text{ctt}}^p$  26, 187, 260, 305, *see* reduction, polynomial-time conjunctive truth-table
- closure of  $C=P$  downward under *see* closure, of  $C=P$  downward under  $\leq_{\text{ctt}}^p$
- closure of  $\text{coNP}$  downward under *see* closure, of  $\text{coNP}$  downward under  $\leq_{\text{ctt}}^p$
- $\leq_d^p$  306, *see* reduction, polynomial-time disjunctive Turing
- $\leq_{\text{dtt}}^p$  26, 29, 268, 305, 307, *see* reduction, polynomial-time disjunctive truth-table
- closure of  $C=P$  downward under *see* closure, of  $C=P$  downward under  $\leq_{\text{dtt}}^p$
- closure of  $\text{coNP}$  downward under *see* closure, of  $\text{coNP}$  downward under  $\leq_{\text{dtt}}^p$
- $\leq_{f(n)-T}^p$  306
- $\leq_{f(n)-\text{tt}}^p$  306
- $\leq_{k-\text{tt}}^p$  10, 11, 245, 275, 306, *see*  $\leq_{\text{btt}}^p$ , *see* reduction, polynomial-time bounded-truth-table
- $\leq_{\text{locpos}}^p$  306, 307, *see* reduction, polynomial-time locally positive Turing
- $\leq_m^p$  1–6, 8, 9, 26, 60, 187, 194, 245, 267–269, 275, 305, *see* reduction, polynomial-time many-one
- closure of  $PP$  downward under *see* closure, of  $PP$  downward under  $\leq_m^p$
- closure of  $SF_5$  downward under *see* closure, of  $SF_5$  downward under  $\leq_m^p$
- closure of the context free languages downward under *see* closure, of the context free languages downward under  $\leq_m^p$

- $\leq_{pos}^p$  296, 306, 307, *see* reduction, polynomial-time positive Turing
- closure of  $C=P$  downward under *see* closure, of  $C=P$  downward under  $\leq_{pos}^p$
- closure of  $coC=P$  downward under *see* closure, of  $coC=P$  downward under  $\leq_{pos}^p$
- closure of NP downward under *see* closure, of NP downward under  $\leq_{pos}^p$
- closure of P-sel downward under *see* closure, of P-sel downward under  $\leq_{pos}^p$
- $\leq_{tt}^p$  248, 251, 260, 296, 305, *see* reduction, polynomial-time truth-table
- closure of PP downward under *see* closure, of PP downward under  $\leq_{tt}^p$
- $\leq_{tt[k]}^p$  249, 251, *see* reduction, polynomial-time constant-round truth-table
- $\leq_I^{sn}$  306, *see* reduction, strong nondeterministic
- $\oplus E$  28
- $\oplus L$  82, 87, 89, 277, 280, *see* class, modulo-based, logspace-analogs
- $\oplus L/poly$  82, 89, *see* advice
- $\oplus OptP$  181, 183, 192, 303
- $\oplus P$  28, 72, 73, 76–82, 97, 101, 181–183, 185, 186, 194, 213, 232, 259, 273, 276, 277, 287, 289, 293, 297, 298, 301
- closure properties *see* closure, of  $\oplus P \dots$
- exponential-time analog of *see* analog, exponential-time, of  $\oplus P$ , *see* analog, exponential-time, of UP, FewP,  $\oplus P$ , ZPP, RP, or BPP
- $\oplus P/poly$  78, 82, *see* advice
- $\oplus SAT$  298, *see* computation, modulo-based
- $\#L$  87, 253, 279, *see* set,  $\#L$ -complete
- $\#P$  76, 78–81, 87, 88, 91–108, 110, 112–115, 119, 121, 163, 164, 237, 238, 240, 265, 273, 276, 286, 287, 289, 293, 297, 299, *see* class, counting, *see* function,  $\#P$ , *see* set,  $\#P$ -complete
- closure properties *see* closure, of  $\#P \dots$
- $\#P$  function *see* function,  $\#P$
- $\#P_1$  287
- $\#SAT$  286
- $\#acc_N(x)$  76, 77, 79–81, 121, 186, 187, 236–238, 251–253, 256–258, 278, 286, 290, 291, 297, 298
- $\#gap_N(x)$  186, 236, 238, 239, 253, 254, 256, 258, 259, *see* function, GapP, *see* function, gap, *see* GapP
- $\#rej_N(x)$  186, 187, 236–238, 258, *see* path, rejecting computation
- 2-ary function *see* function, ...2-ary...
- 2-disjunctively self-reducible set *see* set, 2-disjunctively self-reducible
- 3-CNF formula *see* formula, 3-CNF
- Abadi, M. 64, 164
- AC,  $AC^k$  88, 193, 194, 261, 279–281, 293, 301, 308, *see* class, circuit
- ACC 193, 194, *see* class, circuit
- acceptance
  - cardinality *see* cardinality, acceptance
  - categorical 283
  - mechanism *see* mechanism, acceptance
  - probability *see* probability, acceptance
  - type *see* reduction, of the cardinality of acceptance types, *see* type, acceptance, *see* type, finite-cardinality acceptance
- access
  - $k$ -round parallel 249
  - parallel, to NP 18, 64, 89, 301
  - sublinear-parallel, to NP 64
- $ACC_p$  88, *see* class, circuit
- Adachi, A. 264, 265
- addition
  - closure of  $\#P$  under *see* closure, of  $\#P$  under addition
- adjacency matrix *see* matrix, adjacency
- Aleman, L. 87, 277, 289, 290
- adversary problem *see* problem, adversary
- advice 45, 48, 84, 86, 87, 277, *see*  $C/poly$ , *see*  $coNP/linear$ , *see*  $coNP/poly$ , *see*  $E/linear$ , *see*  $EXP/linear$ , *see* interpreter, *see*  $NL/poly$ , *see*  $NP/linear$ , *see*  $NP/poly$ , *see*  $(NP \cap coNP)/poly$ , *see*  $P/linear$ , *see*  $P/poly$ , *see*  $P/quadratic$ , *see*  $\oplus L/poly$ , *see*  $\oplus P/poly$ , *see*  $PP/linear$ , *see* set, small advice, *see*  $UL/poly$
- for the semi-feasible sets 47–56
- hard to compute 277
- linear-sized 49, 51, 52
- optimal 48, 52

- quadratic length-bounded 48
- subquadratic length-bounded 49
- advice interpreter *see* interpreter, advice
- Agrawal, M. 296, 298
- agreement
  - secret-key 36, 43
- Aho, A. 266
- Ajtai, M. 232, 280
- algorithm
  - census-based 18
  - deterministic exponential-time 24
  - enumeration 119
  - greedy 272
  - high-degree polynomial-time 264
  - interval-pruning vii
  - low-degree polynomial-time 266
  - membership 294
  - nondeterministic 1, 76, *see* NP, *see*  $\Sigma_k^P$
  - NP 62, *see* NP, *see*  $\Sigma_k^P$
  - polynomial-time 3, 5, 6, 22, 23, 25, 41, 80, 139, 146, 165, 266
  - query simulation vii
  - randomized 67
  - sampling 131, 132, 135, 136, 145–147, 149
  - self-reducibility 1
  - semi-membership 294, 295
  - $\Sigma_2^P$  20, 21, *see*  $\Sigma_k^P$
  - that assumes a theorem's hypothesis 1, 2
  - $\Theta_2^P$  19, 20, *see*  $\Theta_k^P$
  - tree-pruning vii, 8
  - UL 85, 86, *see* UL
- algorithms
  - design and analysis of 265, 266
  - their central role in complexity theory vii, 1
- Allender, E. viiii, 28, 29, 43, 88, 89, 231, 260, 261, 265, 270, 278, 279, 281, 283–285, 293, 308
- almost polynomial-time set *see* set, almost polynomial-time
- alphabet 7
- alternating quantifiers *see* quantifiers, alternating
- alternating Turing machine *see* machine, alternating
- alternation
  - symmetric 27
- AM 300
- Ambainis, A. ix, 194
- Amir, A. 63, 163, 286, 296, 297
- analog
  - exponential-time, of  $\oplus P$  28
  - exponential-time, of the polynomial hierarchy 28, 29
  - exponential-time, of UP, FewP,  $\oplus P$ , ZPP, RP, or BPP 28
- AND circuits *see* circuits, AND
- AND gate *see* gate, AND
- AND-OR circuits *see* circuits, AND-OR
- Angluin, D. 87
- Antioch
  - Holy Hand Grenade of 67
- aperiodic monoid *see* monoid, aperiodic
- approach
  - census 18
  - self-reducibility-based tree-pruning 2
  - theorems via algorithms under hypotheses 1
- approximation *see* factor, of approximation, *see* nonapproximability
  - enumerative 119, 163
  - NP-hardness of 166
  - of perm 119, *see* permanent, of a matrix
  - of maximum clique 165, 166, *see* clique
  - of #P functions 286
  - of the proper subtraction of two #P formulas 108
  - of the sign function 235, 242
  - of threshold circuits by parity circuits 260
- argument
  - self-reducibility-based 4
- arithmetic characterization *see* characterization, arithmetic
- arithmetic expression *see* expression, arithmetic
- arithmetic formula *see* formula, arithmetic
- arithmetization 111, 112, 126, 140, 163
- Arora, S. 165, 166, 267, 269
- Arthur-Merlin *see* AM
- Arvind, V. ix, 27, 277, 289, 296, 298
- Aspnes, J. 289
- assignment
  - exactly one satisfying 69
  - lexicographically largest satisfying 163

- lexicographically smallest satisfying 57
- partial 199
- random, partial input 197
- satisfying 4, 7, 56–59, 139, 163, 268, 269, 283, 294
- unique satisfying 45, 56, 57, 284
- assignments
  - even number of satisfying 298
  - number of satisfying 286
- associative operation *see* operation, associative
- associativity *see* function, ...associative..., *see* function, associativity of
  - of concatenation 38
  - of multiplication 38
  - weak 44
- attribute-value description *see* description, attribute-value
- Ausiello, G. 166, 267
- automata *see* machine
  - $k$ -state 302
  - nondeterministic auxiliary pushdown 281
  - nonuniform deterministic finite 193, *see* NUDFA
  - pushdown 281
- automorphism *see* graph, automorphism counting in
  - counting in graph 289
- average-case cryptography *see* cryptography, average-case
  
- Babai, L. 28, 163–165, 232, 273, 300
- Baker, T. 231, 268–270, 272, 273
- Balcázar, J. 27, 43, 64, 87, 273, 286, 287, 293
- Barrington, D. 192–194, 281, 302
- Beals, R. 261, 278, 279, 289
- Beame, P. 233
- Beaver, D. 163
- behavior
  - normal 28, 29
- Beigel, R. ix, 29, 43, 63, 89, 106–108, 163, 194, 260, 269, 284, 286, 293, 296–298, 308
- Bellare, M. 166
- Ben-Or, M. 164, 194, 300
- Bennett, C. 87, 232, 268, 269
- Bent, R. viii
- Berg, C. 232
- Berman, L. 26, 27, 43, 269, 275–277, 282
- Berman, P. 26, 27
- Berman–Hartmanis Isomorphism Conjecture *see* Conjecture, Berman–Hartmanis Isomorphism
- Bertoni, A. 286
- Beygelzimer, A. viii, 44
- Bibliographic Notes *see* Notes, Bibliographic
- bidirectional connectivity *see* connectivity, bidirectional
- bijection 168
  - between natural numbers and strings 91, 92
  - between strings and pairs of strings 33
- binary tree
  - full *see* tree, full binary
- bits
  - a probabilistic circuit's random 219
  - generation of random 289
  - quadratic number of random 88
  - quasilinear number of random 88
  - random, of a probabilistic circuit 220, 221
- blackboard
  - polynomial-sized 274
- Blackburn, P. 274
- Blass, A. 283
- Blaylock, N. ix
- Blum, M. 164
- Book, R. 27, 64, 65, 87, 273–275, 286, 287, 293, 294
- boolean formula *see* formula, boolean
- boolean formula minimization problem *see* problem, boolean formula minimization
- boolean function *see* function, ...boolean...
- boolean hierarchy *see* hierarchy, boolean
- boolean operation *see* operation, boolean
- Boppana, R. 269
- Borchert, B. 285
- Borodin, A. 193, 269, 279, 280
- bottleneck machine *see* machine, ...bottleneck...
- bound
  - lower 43, 44, 194, 197, 207, 223, 232, 233, 248, 264
  - quadratic 264
  - superpolynomial-size lower 264
- bounded



- depth of recursion 17
- polynomial-time 145, 182, 187, 192
- bounded-depth circuits *see* circuits, bounded-depth
- bounded-fan-in circuits *see* circuits, bounded-fan-in
- bounded-width branching program *see* program, bounded-width branching
- boundedness
  - logarithmic-space 85, 86, 254, 255, 258, 259, *see* L, *see* NL, *see* UL
  - polynomial-space 125, *see* PSPACE
  - polynomial-time 7, 12, 80, 81, 115, 118, 119, 121, 122, 135, 187, 238, 239, 246, 250, 255, 259, *see* machine, nondeterministic polynomial-time, *see* NP, *see* P
- Boutell, M. viii
- Bovet, D. ix, 268, 285
- BP operator *see* operator, BP
- BPP 28, 72–75, 77, 78, 81, 87, 273, 276, 277, 288–290, 293, 297, 298, 307, 308, *see* circuits, small for BPP
  - exponential-time analog of *see* analog, exponential-time, of UP, FewP,  $\oplus$ P, ZPP, RP, or BPP
- BPP hierarchy *see* hierarchy, BPP
- branching program *see* program, branching
- branching time logic *see* logic, branching time
- Brassard, J. 43
- Brauer, W. ix
- Braunmühl, B. von *see* von Braunmühl, B.
- Bruschi, D. 232, 297
- brute-force search *see* search, brute-force
- Bshouty, N. 27
- Buhrman, H. 28, 64, 273, 284, 296, 308
- Buhrman–Hemaspaandra–Longpré Encoding *see* Encoding, Buhrman–Hemaspaandra–Longpré
- Buntrock, G. 87, 277, 278
- Burtschick, H. 64, 296
- $C/poly$  75, *see* advice
- $C=L$  261, 278, 279, *see* set,  $C=L$ -complete, *see* set, complete for the  $\leq_L^u$ -reducibility closure of  $C=L$
- $C=L$  hierarchy *see* hierarchy,  $C=L$
- $C=L$  oracle hierarchy *see* hierarchy, oracle, of  $C=L$
- $C=P$  87, 88, 102, 103, 186, 187, 191, 192, 235, 240, 241, 260, 261, 287, 290–293, 298, 308, *see* class, counting, *see* set,  $C=P$ -complete
- closure properties *see* closure, of  $C=P...$
- Cai, J. ix, 26, 27, 29, 64, 106, 107, 163, 164, 192, 194, 232, 265, 273–275, 286, 297, 298, 303
- call
  - recursive 14–18, 124, 125
- Canetti, R. 27
- cardinality
  - acceptance 102, 292
  - rejection 102
- cardinality reduction *see* reduction, of numbers of solutions, *see* reduction, of numbers of witnesses, *see* reduction, of the cardinality of a set of vectors, *see* reduction, of the cardinality of acceptance types, *see* reduction, of the number of accepting paths, *see* reduction, of the number of solutions, *see* reduction, witness
- Carroll, L. 272
- Cass, D. ix
- cat and mouse game *see* game, cat and mouse
- categorical acceptance *see* acceptance, categorical
- categorical machine *see* machine, categorical
- cauldrons
  - of recursion theory vii
- Caussinus, H. 194, 260
- census *see* approach, census
- census function *see* function, census
- census value *see* value, census
- census-based algorithm *see* algorithm, census-based
- certificate
  - deterministically verifiable 109
  - of acceptance or membership 8, 9, 34, 62, 109
  - of primality 131, 132, 142
  - polynomial-time verifiable 131
  - succinct 109, 110
- Chakaravarthy, V. 64
- Chandra, A. 193, 272, 279, 281, 308
- Chang, R. 231, 260, 270, 289, 300, 308
- characteristic function *see* function, characteristic
- characterization

- arithmetic 126
- robust, of low-degree polynomials 111, 112, *see* polynomial, low-degree
- robust, of NC 281
- Chari, S. 89, 231, 270
- Chebyshev's Inequality *see* Inequality, Chebyshev's
- Chebyshev's Theorem *see* Theorem, Chebyshev's
- Chebyshev, P. 135, 163
- Chen, S. viii
- Cheng, Y. viii
- Chinese Remainder Theorem *see* Theorem, Chinese Remainder
- Chor, B. 300
- Church A. 264
- circuit *see* graph, representing
  - boolean circuit, *see* graph, representing unbounded-fan-in circuit, *see* subcircuit
- depth reduction *see* reduction, of the depth of a circuit
- circuit family *see* family, circuit
- circuits 175, 197, 200, 276, 279, *see* class, circuit, *see* disjointness, of circuits
  - AND 203, 225
  - AND-OR 88, 201, 206–208, 211, 224, 227, 232
  - bounded-depth 211
  - bounded-fan-in 167, 169–171, 174, 177
  - computing the exclusive-or of three bits via 220
  - constant-depth 207, 212, 213, 218, 223–225, 228, 232
  - constant-depth  $2^{\log^{\mathcal{O}(1)} n}$ -size 218
  - constant-depth polynomial-size 88, 89, 201, 202, 207, 223, 231, 302
  - constant-depth polynomial-size unbounded-fan-in 280
  - constant-depth subexponential-size 213
  - constant-depth superpolynomial-size 223, 224, 232, 280
  - depth-0 170
  - depth-1 199, 204
  - depth-2 202, 232
  - depth-2  $2^{\log^{\mathcal{O}(1)} n}$ -size 89
  - depth-2  $2^{\log^{\mathcal{O}(1)} n}$ -size polylogarithmic bottom-fan-in 88, 89
  - depth-2  $2^{\log^{\mathcal{O}(1)} n}$ -size probabilistic 88
  - depth-3  $2^{\log^{\mathcal{O}(1)} n}$ -size polylogarithmic bottom-fan-in 88
  - depth-4  $2^{\log^{\mathcal{O}(1)} n}$ -size polylogarithmic bottom-fan-in 88
  - depth- $k$  197
  - depth- $k$  bounded-fan-in 174
  - depth- $k$  linear-size 232
  - depth- $\mathcal{O}(k \log k)$  bounded-fan-in 170
  - deterministic 89, 220, 221, 231
  - family of 201
  - for SAT, size of 87
  - for sets in DSPACE[n], size of 87
  - in the shape of a tree 227
  - logarithmic-depth polynomial-size bounded-fan-in 171
  - logspace-uniform 279, 280
  - NC<sup>1</sup> 176
  - OR 205, 210, 225
  - OR-AND 201, 204, 206, 207, 211, 212, 224, 227
  - P-uniform 279
  - parity 260
  - polylog-depth constant-size 308
  - polylog-depth polynomial-size 280, 308
  - polynomial-depth 176
  - polynomial-size 233
  - probabilistic 219, 220, 231
  - small 276, 277
  - small, for BPP 277
  - small, for NP 277
  - small, for the semi-feasible sets 45, 47
  - stratified 198
  - superpolynomial-size 222
  - threshold 260
  - transforming tree into 169
  - unbounded fan-in 198
- class
  - advice *see* advice
  - Arthur–Merlin 300
  - being “almost” closed under an operation 108
  - bounded-ambiguity 43, *see* FewP, *see* UL, *see* UP
  - circuit 88, 279, *see* AC, AC<sup>k</sup>, *see* ACC<sub>p</sub>, *see* ACC, *see* circuits, *see* NC, NC<sup>k</sup>, *see* nonuniform-NC<sup>1</sup>, *see* SAC, SAC<sup>k</sup>, *see* SC

- counting 64, 82, 290–293, 297, *see*  $C=L$ , *see*  $C=P$ , *see* PL, *see* PP, *see*  $\#L$ , *see*  $\#P$ , *see* SPP
- “exactly-half”-based 192, *see*  $C=L$ , *see*  $C=P$
- exponential-time 23, 28, 29, 274, 275, *see* EXP, *see* E, *see* NEXP, *see* NE
- GapP-based characterization of 240
- interactive proof 299, *see* IP, *see* MIP, *see* system, interactive proof
- logspace 82, 277–279, *see*  $C=L$ , *see* L, *see* NL, *see*  $\oplus L$ , *see* PL
- low-error probabilistic, as intuitively feasible 289
- majority-based 192, *see* PL, *see* PP
- modulo-based 297, *see*  $\text{coMod}_k P$ , *see*  $\text{FTM}_k P$ , *see*  $\text{Mod}_k P$ , *see*  $\text{ModZ}_k P$ , *see*  $\oplus E$ , *see*  $\oplus L$ , *see*  $\oplus P$
- modulo-based, logspace analogs 277, *see*  $\oplus L$
- one-way function 31, *see* UP
- optimization-based 297, *see* OptP
- output-cardinality-based 297, *see* SpanP
- polynomial-time 28, 274, *see* P
- probabilistic 277, 290, *see* BPP, *see* PL, *see* PP, *see* ZPP
- “promise”-like 293, *see* BPP, *see* coUP, *see* FewP, *see* RL, *see* RP, *see* UL, *see* UP, *see* ZPP
- query-order-based 303
- uniform complexity 168, *see* advice
- whose name sears the tongues of mere mortals vii
- whose very name contains hundreds of characters vii
- clause 112, 139
- Cleve, R. 27, 194
- clique 165, 166
- closure
  - of a class under a reducibility, potential 235
  - of a class under an operation, potential 235
  - of  $C=P$  downward under positive truth-table reductions 260
  - of  $C=P$  downward under  $\leq_{ctt}^p$  187, 240, 260
  - of  $C=P$  downward under  $\leq_{dt}^p$  240
  - of  $C=P$  downward under  $\leq_{pos}^p$  260, 293
  - of  $C=P$  downward under coNP-many-one reductions 241
  - of  $\text{coC}=P$  downward under  $\leq_{pos}^p$  260, 293
  - of coNP downward under coNP-many-one reductions 241
  - of coNP downward under  $\leq_{ctt}^p$  240
  - of coNP downward under  $\leq_{dt}^p$  240
  - of  $\mathcal{F}$  (a class) under  $\sigma$  (an operation) 92
  - of  $\mathcal{F}$  (a class) under integer division 98
  - of LOGCFL under complementation 280
  - of  $\text{Mod}_k P$  under union 298
  - of natural polynomials under multiplication 247
  - of NP downward under  $\leq_{pos}^p$  268, 307
  - of NP under intersection 268
  - of NP under union 268
  - of OptP under proper subtraction, potential lack of 104, 105
  - of P downward under  $\leq_{btt}^p$  96
  - of P under union 36
  - of  $\oplus P$  downward under  $\leq_T^p$  182
  - of PL downward under P-uniform  $\text{NC}^1$  reductions 260
  - of PP downward under constant-round truth-table reductions 249, 251
  - of PP downward under  $\leq_{btt}^p$  245
  - of PP downward under  $\leq_m^p$  239, 245
  - of PP downward under  $\leq_T^p$ , potential lack of 252, 259
  - of PP downward under  $\leq_{tt}^p$  235, 245, 293
  - of PP downward under P-uniform  $\text{NC}^1$  reductions 260
  - of PP downward under polynomial-time parity reductions 260
  - of PP under complementation 97, 239, 245, 260
  - of PP under disjoint union 245
  - of PP under intersection 235, 242, 244, 245
  - of PP under union 245
  - of probabilistic- $\text{NC}^1$  under intersection 260
  - of P-sel downward under  $\leq_{pos}^p$  296
  - of P-sel under almost-completely degenerate boolean functions 296

- of P-sel under complementation 52, 296
- of P-sel under completely degenerate boolean functions 296
- of P-sel under intersection, lack of 296
- of P-sel under union, lack of 296
- of  $\text{SAC}^k$  under complementation 280
- of  $\text{SF}_5$  downward under  $\leq_m^p$  177
- of  $\#P$  under addition 76, 87, 92, 100, 101, 106, 287
- of  $\#P$  under finite sums of multiples of binomial coefficients whose upper element is the input and whose lower element is a constant 107
- of  $\#P$  under integer division by 2, potential lack of 100, 101, 287
- of  $\#P$  under integer division, potential lack of 98, 99, 101, 287
- of  $\#P$  under maximum, potential lack of 100–103, 287
- of  $\#P$  under minimum, potential lack of 100–103, 287
- of  $\#P$  under multiplication 92, 93, 101, 106, 287
- of  $\#P$  under operators in relativized worlds 107
- of  $\#P$  under proper decrement, potential lack of 100, 105, 106, 287
- of  $\#P$  under proper subtraction, potential lack of 91, 93, 95, 96, 98, 99, 104, 107, 287
- of  $\text{SpanP}$  under proper subtraction, potential lack of 105
- of the context free languages downward under  $\leq_m^p$  279
- of  $\Theta_2^p$  under complementation 20
- of  $\text{UP}$  under intersection 284
- Cobham, A. 264, 265
- $\text{coC=P}$  260, 293
  - closure properties *see* closure, of  $\text{coC=P}$ ...
- coefficients
  - binomial 93
  - multinomial 93
- cofinite set *see* set, cofinite
- coin
  - fair 74, 131
  - unbiased 288
- collision
  - in image of one-way function 31
- potential relationship between intensity of and collapses 31
- combinatorial game *see* game, combinatorial
- commutativity *see* function, ...commutative...
  - of concatenation, lack of 38
  - of min 41
  - of multiplication 38
  - of subtraction, lack of 38
- commutator 168, 172, 174, 175
- $\text{coMod}_kP$  194, 298, 301
- companion
  - best  $v$
- comparison
  - lexicographic *see* order, lexicographic
- complementation
  - closure of LOGCFL under *see* closure, of LOGCFL under complementation
  - closure of PP under *see* closure, of PP under complementation
  - closure of P-sel under *see* closure, of P-sel under complementation
  - closure of  $\text{SAC}^k$  under *see* closure, of  $\text{SAC}^k$  under complementation
  - closure of  $\Theta_2^p$  under *see* closure, of  $\Theta_2^p$  under complementation
- complete equality *see* equality, complete
- completeness
  - of a protocol 110, 115, 116, 131, 132, 134, 135, 138, 145, 147, 149
  - of a set for a class  $C$  *see* the entry for that class  $C$
  - of a set for some class  $C$  *see* set, “[that class  $C$ ]”-complete
- complexity
  - circuit, of SAT 89
  - Kolmogorov 269
  - nonuniform 45, 51, *see* advice
- complexity theory *see* theory, complexity
- computation
  - ambiguity-bounded 281, 285, *see* FewP, *see* UP
  - bottleneck 181, 300, *see* machine, bottleneck, *see* machine, bounded-width bottleneck, *see* ProbabilisticSSF $_k$ , *see* SF $_k$ , *see* SSF $_k$
  - deterministic logspace 278, *see* L

- deterministic polynomial-time 22, *see* P
- error-bounded probabilistic 288, 289, *see* BPP, *see* coRP, *see* RP
- exponential-time deterministic 275, *see* EXP, *see* E
- exponential-time nondeterministic 275, *see* NEXP, *see* NE
- majority-based probabilistic symmetric bottleneck 192
- modulo-based 106, 297, 298, *see*  $\text{coMod}_k\text{P}$ , *see*  $\text{FTM}_k\text{P}$ , *see*  $\text{Mod}_k\text{P}$ , *see*  $\text{ModZ}_k\text{P}$ , *see*  $\oplus\text{E}$ , *see*  $\oplus\text{L}$ , *see*  $\oplus\text{P}$
- nondeterministic logspace 278, *see* NL
- nondeterministic polynomial-time 22, *see* NP
- nondeterministic space-bounded 125, *see* NL
- polynomial-ambiguity 283, *see* FewP
- polynomial-space 176, *see* PSPACE
- PSPACE oracle 213
- quantum 194, 195
- relativized logspace 279
- semi-feasible 294–296, *see* P-sel
- single-valued nondeterministic function 57, *see* NPSV
- unambiguous 281, 283, 285, *see* UL, *see* UP
- unambiguous logspace 84, 85, *see* UL
- zero-error probabilistic 290, *see* ZPP
- concatenation *see* assignments, of concatenation
- condition
  - truth-table 10, 11, 13, 14, *see* refinement, of a truth-table condition
- Condon, A. 164
- coNE 275
- confidence
  - high 109
- configuration
  - unique accepting 126, 129
  - unique middle-point 112
- configuration space *see* problem, robotics configuration space
- conflicting oracle results *see* results, conflicting oracle
- conflicting relativizations *see* results, conflicting oracle
- Conjecture
  - Berman–Hartmanis Isomorphism 26, 282
  - One-Way 282, 285
- coNL 82, 277, 278
- connectivity
  - bidirectional 177
- coNP 5, 6, 28, 52, 59–62, 64, 95–97, 100, 104–106, 194, 240, 241, 269, 271–273, 276, 277, 287, 296, 306, 307, *see* set, coNP-complete, *see* set, potential lack of sparse  $\leq_m^p$ -complete, for coNP, *see* set, potential lack of sparse  $\leq_m^p$ -hard, for coNP
- closure properties *see* closure, of coNP...
- coNP/linear 52, 296
- coNP/poly 61, 64, 276, 296
- constant function *see* function, constant
- constant-depth circuits *see* circuits, constant-depth, *see* circuits
- constant-to-one function *see* function, constant-to-one
- construction
  - oracle 197, 223
  - widely-spaced 53, 223
- context-free grammar *see* grammar, context-free
- context-free languages
  - closure properties *see* closure, of context-free languages...
- context-free set *see* set, context-free
- Cook's Theorem *see* Theorem, Cook's
- Cook, S. 58, 59, 64, 91, 163, 266–268, 275, 279–281, 305, 308
- Cook–Karp–Levin Theorem *see* Theorem, Cook's
- Cook–Levin Theorem *see* Theorem, Cook's
- Cormen, T. 266
- coRP 288–290
- coSSF<sub>k</sub> 185
- Count 84–86
- counter
  - as auxiliary input 302
- counting
  - enumerative *see* approximation, enumerative
- counting hierarchy *see* hierarchy, counting
- coUP 106, 107, 282, 284
- course, use of this book in *see* textbook, use of this book as

- coUS 284
- Crescenzi, P. 166, 267, 269, 285
- cryptocomplexity
  - average-case *see* cryptography, average-case
  - worst-case *see* cryptography, worst-case
- cryptographic protocol *see* protocol, cryptographic
- cryptography
  - and UP 282, 283
  - average-case 31, 44
  - central role of one-way functions in 31
  - not helped by non-honest functions 32
  - worst-case 31, 33, 44, 282
- Culbertson, E. 270
- culling method *see* method, culling
- cyclic ring *see* ring, cyclic
  
- DAAD ix
- Damm, C. 87, 277
- DARPA ix
- Davis, M. 264
- Deaett, L. viii
- decision graph *see* graph, decision
- decrementation
  - proper 101
- defeat
  - another element in a tournament *see* tournament, defeat in
  - another element with respect to a P-selector *see* tournament, defeat in
- degree
  - reducibility 240
  - total, of a polynomial 111, 112, 126, 140–144, 147, 148, 150–153, 155, 156, 160, 162
- Demers, A. 269
- DeMillo, R. 163
- Denny-Brown, D. 295
- dense set *see* set, dense
- density 218, 219
- depth
  - of a recursion tree 14
- derandomization 87
- description
  - attribute-value 274
  - instantaneous *see* ID
- determinant
  - of a matrix 148, 149
- deterministic circuits *see* circuits, deterministic
- deterministic machine *see* machine, ...deterministic...
- diagonal elements *see* elements, diagonal
- diagonalization 55, 197, 285
- Díaz, J. 29, 43
- difference
  - symmetric 68, 235, 260
- digital signature protocol *see* protocol, digital signature
- dimension
  - of a matrix 114, 115, 117, 122, 149
- direct product *see* product, direct
- disjoint sets *see* sets, disjoint
- disjoint union *see* union, disjoint
  - closure of PP under *see* closure, of PP under disjoint union
- disjointness 16, 17, 205
  - of intervals 12–14
  - of restrictions 200, 209
  - of subcircuits 205
- disjunctive self-reducibility *see* self-reducibility, disjunctive, of SAT
- disjunctive self-reducible set *see* set, disjunctive self-reducible
- distribution 224, 225
  - probability 219, 299, 302
  - probability, of restrictions 198, 200, 202, 204, 207, 208, 210, 224, 225, 232
  - uniform 116, 131, 162, *see* selection, under uniform distribution
- divide and conquer
  - as a motto 45
- Dolev, D. 193
- domain 32, 33, 37, 38, 44, 121, 122, 141, 142, 152, 247
  - having size at least two 44
  - relation of, between a function and a refinement of that function 58
  - size 84
- double-exponential time *see* time, deterministic double-exponential
- downward closure *see* closure, *see*  $R_a^b(C)$ ,  $R_a^b(C)$
- downward path *see* path, downward
- downward separation *see* separation, downward
- DSPACE 87, 89, 271
- DTIME 53, 56, 165, 264, 265, 274, 275, 296
- Du, D. 282

- Durand, A. 108  
 Dymond, P. 279, 280
- E 23–25, 28, 64, 265, 268, 273–275,  
 307, *see* set, E-complete  
 E/linear 64  
 $E_a^b(C)$ ,  $E_a^b(C)$  276, 296, 307  
 edge 45, 46, 50–52, 62, 83–86, 177–179,  
 198, 199  
 Edmonds, J. 264, 265  
 element  
 – minimum-weight 71  
 – range 41, 42  
 elements  
 – diagonal 82, 159  
 elimination  
 – Gaussian 146  
 Emde Boas, P. van *see* van Emde  
 Boas, P.  
 Emerson, E. 274  
 Encoding  
 – Buhrman–Hemaspaandra–Longpré  
 28  
 – Hartmanis–Immerman–Sewelson  
 22, 24, 25, 27, 28  
 enumeration  
 – of all polynomial-time computable  
 functions 216  
 – of machines as a central tool in  
 proving the existence of complete  
 sets 285  
 – of NPTMs 58, 268  
 – of NPTMs that are unambiguous on  
 all inputs, seeming lack of 285  
 – of oracle NPTMs clocked in a way  
 that holds over all oracles 60  
 – of relativized PH machines 213  
 – of relativized predicates 221  
 – of relativized predicates specifying  
 alternating quantifications 221  
 enumeration algorithm *see* algorithm,  
 enumeration  
 enumerative approximation *see*  
 approximation, enumerative  
 enumerator 119, 163  
 – polynomial-time computable 119,  
 163, 287, 289  
 $E_{k-T}^P(P\text{-sel})$  296  
 $E_{k-nt}^P(P\text{-sel})$  296  
 $E_T^P(P\text{-sel})$  296  
 $E_T^P(\text{SPARSE})$  276  
 equality  
 – complete 43  
 – weak 43  
 Erdős, P. 28  
 error  
 – one-sided 87, 165, 290  
 – two-sided 166  
 evaluation  
 – query *see*  $\leq_{tt}^P$ , *see* reduction,  
 polynomial-time truth-table  
 evaluator  
 – query 247, 248  
 evidence  
 – relativized 18, 27  
 exhaustive search *see* search,  
 exhaustive  
 EXP 55, 163, 164, 274, 275, 284, 296,  
 298, 307, *see* set, EXP-complete  
 EXP/linear 296  
 expansion  
 – of a tree 6  
 expectation 135, 137, 203  
 exponential hierarchy  
 – strong *see* hierarchy, strong  
 exponential  
 exponential time *see* time, determin-  
 istic exponential  
 expression  
 – arithmetic 138, 140
- factor  
 – of approximation 165, 166, 286  
 False 1, 3–7, 14, 21  
 family  
 – circuit 219, 279  
 family of circuits *see* circuits, family  
 of  
 Feige, U. 164–166  
 Feigenbaum, J. 64, 163, 164  
 Feller, W. 163  
 Fellows, M. 284  
 Fenner, S. ix, 108, 260, 269, 282, 284,  
 290, 293, 294, 298  
 FewP 28, 43, 281, 283–285, 287, 293,  
 296, 298  
 – exponential-time analog of *see*  
 analog, exponential-time, of UP,  
 FewP,  $\oplus P$ , ZPP, RP, or BPP  
 Fich, F. 193  
 filter 88  
 finite monoid *see* monoid, finite  
 finite set *see* set, finite  
 first 37, 39, 42  
 Fischer, D. 289  
 Fischer, M. 264, 274, 275, 289  
 Fischer, P. 29

- Fischer, S. 287
- flexibility
- of query generation in  $\leq_T^P$  18
- Formula *see* boolean formula
- Lagrange Interpolation 163
  - Newman's 260
- formula
- 3-CNF 112, 139
  - arithmetic 110
  - boolean 3, 29, 58, 69, 265, 267, 268, 272, 281, 286
  - fully quantified boolean 177
  - quantified boolean 274
  - quantifier-free boolean 274, *see* QBF
  - satisfiable boolean 1, 3–7, 56, 58, 112, 267, 294, *see* SAT
  - unsatisfiable boolean 8, 29
  - variable-free 3, 7
  - well-formed 21
- Forster, J. ix
- Fortnow, L. ix, 28, 108, 163–165, 231, 260, 269, 270, 273, 282, 284, 290, 293, 298
- Fortune, S. 26, 43, 193
- FP 10, 11, 57–59, 64, 78, 80, 81, 89, 119, 236–239, 268, 291, 293, 294, 305–307, *see* function, polynomial-time computable
- Fraenkel, A. 272
- Frankl, P. 28
- Frege proof *see* proof, Frege
- Frost, R. 266
- $f_{SAT}$  61, 294
- $FTM_kP$  106, 107, 287
- Fu, B. 260
- function
- ...two-argument... *see* function, ...2-ary...
  - 2-ary 38, 44
  - 2-ary one-way 31, 32, 36, 37, 39, 43
  - 2-ary, definitions of additional properties for 37
  - 2-ary, lower bound on the degree of many-to-one-ness of 44
  - advice 48–50, 52, 67, 82, 84
  - almost-completely degenerate boolean 296
  - almost-completely degenerate boolean, closure of P-sel under *see* closure, of P-sel under almost-completely degenerate boolean functions
  - associative 42, 43, *see* function, ...associative...
  - associative 2-ary 38
  - associative, 2-ary one-way 44
  - associativity of 38
  - boolean 10, 11, 14, 174, 208, 245
  - bounded-ambiguity 35
  - bounded-ambiguity one-way 35, 284
  - census 49
  - characteristic 105, 242
  - characterizing a  $\leq_{btt}^P$ -reduction 10
  - commutative 41, 44
  - commutative 2-ary 38
  - commutative, associative, 2-ary 43
  - commutativity of 38
  - completely degenerate boolean 296
  - completely degenerate boolean, closure of P-sel under *see* closure, of P-sel under completely degenerate boolean functions
  - constant 96, 181, 200, 208, 211, 239
  - constant-to-one 35
  - constant-to-one one-way 31, 35
  - determinant 279
  - deterministic 291, 294
  - dishonest 37
  - edge-weight 83
  - gap 235, 236, 258, 260, *see* function, GapP, *see* GapP, *see* GapNC, *see* GapP, *see* #gap<sub>N</sub>(x)
  - GapP *see* function, gap, *see* GapP, *see* #gap<sub>N</sub>(x)
  - GapP 102, 236, 237, 239–242, 244, 247, 248, 251, 252
  - honest 32–34, 37, 41–43, 269, 282
  - honest 2-ary 37, *see* function, honest
  - honest, strongly noninvertible 43
  - inverse of 32, 34, 35, 37, 42
  - invertible 33
  - k-to-one 35
  - length-decreasing 32
  - low-ambiguity, commutative, associative one-way 42
  - magic 223
  - many-one one-way 268
  - mod 106, 119, 120, 122, 127, 129, 132, 133, 141, 146, 155, 158, 160–162, 193
  - multivalued 56–59, 61, 89, 292, 294



- nondeterministic 56, 291, 294, *see* function, NPMV, *see* NPMV, *see* NPSV
- nondeterministic polynomial-time 56, *see* function, NPMV, *see* NPMV, *see* NPSV
- nondeterministic selector 57
- nondeterministic total 64
- noninvertible 32–34
- NPMV 58, 59, 61, 63–65, 108, 291–294, *see* NPMV
- NPSV 57, *see* NPSV
- NPSV-selector 61–63, 297, *see* NPSV-sel, *see* selectivity
- of a matrix 114
- one-argument one-way 32, 36
- one-to-one 33, 34, 42, 43
- one-to-one one-way 31, 34, 35, 43
- one-way 31–33, 35, 37–39, 42, 43
- optimization 297, *see* function, OptP, *see* OptP
- OptP *see* OptP
- OptP 103–105, 182, 297
- oracle 147, 149, 155
- P-selector 47–49, 51, 54, 297, *see* P-sel, *see* selectivity
- pairing 39, 54, *see* function, standard pairing
- parity 167, 193, 197, 200–202, 204, 206, 207, 213, 216, 218, 219, 222, 224, 231, 232, 280, 297
- partial 38, 43, 57, 59, 61, 62, 64, 282, 291, 294
- partial selector 61
- permanent 110, 112–116, 119, 122
- polynomial-time computability and invertibility of pairing 33
- polynomial-time computable 5, 32, 33, 37, 38, 40, 41, 47, 48, 57, 79, 99, 106, 113, 176, 180, 181, 185, 194, 213–216, 228, 237, 238, 241, 246, 265, 268, 269, 282, 284, 290–292, *see* FP, *see* function, polynomial-time computable...
- polynomial-time computable 2-ary 294, 295
- polynomial-time computable one-to-one 265
- polynomial-time computable total 241
- polynomial-time invertible 32, 33, 41, 246
- polynomial-time invertible 2-ary 37
- polynomial-time noninvertible 37
- polynomial-time selector 295
- polynomial-to-one one-way 43
- polynomially bounded 256
- probabilistic 187
- projection 37
- ranking 286
- recursive 56, 123
- recursive selector 295
- s-honest 37, 38, 41
- s-honest 2-ary 38, *see* function, s-honest
- selector 53, 57, 59, 61, 62, 294, 297, *see* selectivity
- selector, oblivious to the order of its arguments 297
- selector, symmetric 297
- #P 80, 88, 92, 94–97, 99–103, 106, 121, 286, *see* #P
- sign 235, 242
- single-valued 57, 294
- single-valued NPMV *see* NPSV
- special pairing 70
- standard pairing 33, 39–41
- strong, total, commutative, associate one-way 36
- strong, total, commutative, associative 2-ary one-way 36, 37
- strong, total, commutative, associative, 2-ary one-way 40, 42
- strongly noninvertible 2-ary 38
- strongly noninvertible, total, commutative, associative, 2-ary,  $\mathcal{O}(n)$ -one one-way 44
- strongly noninvertible, total, commutative, associative, 2-ary, many-one one-way 31, 38, 39, 268
- symmetric 231
- total 236–238
- total single-valued 282
- total, associative, 2-ary one-way 44
- total, associative, 2-ary one-way, upper bounds on the degree of many-to-one-ness of 43
- total, weakly-associative, 2-ary one-way, potential lack of 44
- two-argument... *see* function, ...2-ary...
- unambiguous 35
- unambiguous inversion of 35
- unambiguous one-way 35, 284

- weight 68, 69, 71, 72, 83, 84
- zero, modulo a prime 144
- Füredi, Z. 28
- Fürer, M. 289
- Furst, M. 192, 194, 232, 264, 303
  
- Gabarró, J. 43
- Gál, A. 87, 89
- gallery
  - rogues' 263, 305
- Gambosi, G. 166, 267
- game *see* problem, game
  - cat and mouse 264
  - combinatorial 265
  - in a tournament 46
  - of pursuit and evasion 265
- GAP 82, 83, *see* Problem, Graph Accessibility
- $\widehat{\text{GAP}}$  82–86
- GapL 235, 252–254, 257, 258, *see* function, gap
- GapNC 194, *see* function, gap
- GapP 102, 107, 108, 191, 192, 235–242, 244, 245, 247–249, 251–253, 258–260, *see* function, gap, *see* function, GapP, *see* function,  $\#gap_N(x)$
- Garey, M. 87, 267, 272
- Gasarch, W. ix, 63, 163, 286, 289, 296, 297
- gate
  - AND 88, 89, 172, 177, 178, 198, 201, 217, 229, 279, 308
  - input 172, 174, 198, 217, 229, 230
  - MAJORITY 88, 89
  - MODULO 88, 302
  - OR 88, 89, 172, 173, 176–178, 198, 201, 217, 220, 229, 279, 308
  - oracle 308
  - output 172, 174, 177, 198, 201, 217, 220, 229, 230
  - PARITY 88
  - symmetric 89
  - threshold 232, 260
  - top 198
- Gaussian elimination *see* elimination, Gaussian
- Gavaldà, R. 27, 64, 276
- Geffert, V. ix, 27
- GEM 2, 31, 32, 35, 45, 62, 68, 93, 110, 168, 197, 236
- Gemmell, P. 164
- generator
  - polynomial-time pseudorandom 265
  - pseudorandom 31, 44
  - query 246–248, 251, 257
  - query, length-increasing 246–248, 257
- generators
  - permutation group membership from 264
- Gengler, R. 27
- Gill, J. 87, 106, 231, 232, 260, 268–270, 272, 273, 278, 279, 288–290, 293, 297, 298
- Glaßer, C. 27
- Gödel, K. 264
- Goldberg, A. 265, 286
- Goldreich, O. ix, 269, 289, 300
- Goldschlager, L. 297, 298
- Goldsmith, J. 29, 265, 287, 293, 296
- Goldstein, G. viii
- Goldwasser, S. 163–166, 269, 277, 300
- Goldwurm, M. 286
- Gottlob, G. 269
- Grail
  - Holy 67
- grammar
  - context-free 265, 272
- graph 45
  - automorphism counting in 289
  - decision 193
  - directed, reachability sets in 50
  - representing boolean circuit 279
  - representing unbounded fan-in circuit 198
  - short paths problem in 233
  - topologically sorted directed 278
  - tournament 45, 47, 50, 52, *see* tournament
- Graph Accessibility Problem *see* Problem, Graph Accessibility
- graph isomorphism *see* Problem, Graph Isomorphism
- Graph Isomorphism Problem *see* Problem, Graph Isomorphism
- graphs
  - isomorphic 289
- greedy algorithm *see* algorithm, greedy
- Green, F. viii, 88, 232
- Greenlaw, R. 265
- Grollmann, J. 43, 282–284
- group
  - multiplicative 112

- nilpotent 193
- nonsolvable 167, 174
- nonsolvable permutation 167, 168
- order of *see* order, of a group
- permutation 167, 168
- growth
  - slower, of polynomials 264
- Gruska, J. 194
- guess
  - of an oracle answer 186
- Gundermann, T. 26, 27, 106, 107, 260, 274, 275
- Gupta, S. 88, 106–108, 260
- Gurevich, Y. 269, 283
- Guruswami, V. 166
  
- Halpern, J. 274
- Han, Y. 27, 265, 277, 289, 295
- hardness *see* NP-hard
  - for UP 283
  - NP 166
  - $\text{NP-}\leq_{\text{btt}}^p$  8, 26
  - $\text{NP-}\leq_{\text{ctt}}^p$  26
  - $\text{NP-}\leq_{\text{dtt}}^p$  26
  - $\text{NP-}\leq_m^p$  18
  - $\text{NP-}\leq_T^p$  18, 20
  - of sets, classifying via reductions 305
  - relative, of sets 305
- Hardy, G. 163
- Hartmanis, J. ix, 22, 24, 26–29, 106, 107, 163, 231, 264, 269, 270, 272, 274–277, 282–285, 300
- Hartmanis–Immerman–Sewelson Encoding *see* Encoding, Hartmanis–Immerman–Sewelson
- Hartmanis–Immerman–Sewelson Theorem *see* Encoding, Hartmanis–Immerman–Sewelson
- Håstad, J. 44, 166, 232, 269, 300
- head
  - input-tape 255
  - work-tape 255
- Heberle, E. ix
- Heller, H. 87, 289
- Hemachandra, L. *see* Hemaspaandra, L.
- Hemaspaandra, E. ix, 28, 63, 272–274, 296, 303, 308
- Hemaspaandra, L. iv, ix, 1, 26–29, 43, 44, 63–65, 106–108, 163, 192–194, 231, 260, 265, 268, 269, 272–277, 282–290, 293, 295–298, 303, 308
- Hempel, H. ix, 28, 273, 303
- Hermann, M. 108
- Hertrampf, U. viii, 87, 88, 107, 194, 277, 297, 298, 301, 303
- hierarchy
  - alternation-based, small-space 27
  - arithmetical 270, 271, 277
  - boolean 27
  - bounded to  $\Sigma_k^p$  28
  - BPP 73, 75
  - $\text{C=L}$  261
  - counting 252
  - counting, logspace analog of 279
  - Kleene 271, 277, *see* hierarchy, arithmetical
  - limited-nondeterminism 29
  - NL 82
  - oracle, of  $\text{C=L}$  279
  - oracle, of PL 279
  - PL 252, 254, 256, 260
  - polynomial 1, 18, 22, 25, 28, 29, 43, 50, 56, 58, 63–65, 67, 73, 78, 81, 82, 87, 115, 186, 197, 213, 222, 223, 228, 231, 232, 268–275, 277, 296, 297, *see*  $\Delta_k^p$ , *see* hierarchy, polynomial, *see* PH, *see*  $\Pi_k^p$ , *see*  $\Sigma_k^p$ , *see*  $\Theta_k^p$
  - polynomial, exponential-time analogs of 22, 274
  - polynomial-time *see* enumeration, of relativized PH machines, *see* hierarchy, polynomial
  - probabilistic logspace *see* PLH
  - query, to NP 25, 28
  - strong exponential 22, 27, 275
- Hoang, T. 279
- Hoene, A. 64, 193, 269, 296, 297, 303
- Hoffmann, C. 264
- Hofmann, A. ix
- Holy Grail *see* Grail, Holy
- Holzwarth, F. ix
- Homan, C. viii, 44
- Homer, S. 26, 27, 265, 275, 294
- honest function *see* function, honest
- honesty *see* function, honest
  - why a natural condition 32
- Hoover, H. 265
- Hopcroft, J. 27, 43, 264, 266
- Huang, M. 289, 290
- Hunt, H. 274, 275
- Huynh, D. 272, 286
  
- ID 255, 256
- Ilardi, P. ix

- Immerman, N. 22, 24, 27–29, 269, 274, 275, 277, 278, 281  
 immunity 28, 232, 284  
 Impagliazzo, R. 29, 44, 64, 233, 289  
 In Polynomial Time We Trust 45  
 incantations vii  
 Inequality  
   – Chebyshev's 135, 137, 163, 203, 204  
 input gate *see* gate, input  
 input tape *see* tape, input  
 instantaneous description *see* ID  
 instantiation  
   – random, of variables 110  
 integer subtraction *see* subtraction, integer  
 interaction 109, 110, 115, 116, 123, 124, 132, 136, 137, 299, *see* system, interactive proof  
 interpolation  
   – polynomial *see* Technique, Polynomial Interpolation  
 interpreter  
   – advice 48, 49, 52, 62, *see* advice  
   – nondeterministic 51, *see* advice  
   – probabilistic *see* advice  
   – probabilistic, of advice 49  
 intersection  
   – closure of NP under *see* closure, of NP under intersection  
   – closure of PP under *see* closure, of PP under intersection  
   – closure of probabilistic-NC<sup>1</sup> under *see* closure, of probabilistic-NC<sup>1</sup> under intersection  
   – closure of UP under *see* closure, of UP under intersection  
 interval 12–18, *see* disjointness, of intervals, *see* procedure, interval-pruning, *see* refinement, of a set of intervals, *see* splitting, of intervals  
 invariant 4, 5  
 inverse  
   – matrix 159  
   – multiplicative 120, 160  
 IP 111, 114, 115, 122, 123, 125, 131, 163, 164, 273, 288, 299, 300, *see* system, interactive proof  
 Isolation Lemma *see* Lemma, Isolation  
 isomorphism  
   – graph *see* Problem, Graph Isomorphism  
 Istrate, G. viii  
 Iwata, S. 264, 265, 272, 274  
 Jain, S. 268, 283, 285, 288, 308  
 Jenner, B. 278, 289  
 Jha, S. 28, 29  
 Jiang, Z. 63, 296, 297  
 Jockusch, C. 295  
 Johnson, D. 87, 267, 272  
 Jones, N. 265  
 Joseph, D. 282, 283, 296, 297  
 JSPS ix  
 Jung, H. 260, 278, 279  
  
*k*-locally self-reducible set *see* set, *k*-locally self-reducible  
*k*-round parallel access *see* access, *k*-round parallel  
 Kadin, J. 27  
 Kämper, J. 64  
 Kann, V. 166, 267  
 Kannan, R. 88  
 Kannan, S. 27, 164  
 Kao, M. 289  
 Karloff, H. 163, 270  
 Karp, R. 20, 27, 60, 64, 91, 266, 267, 276, 277  
 Karp–Lipton Theorem *see* Theorem, Karp–Lipton  
 Kasai, T. 264, 265, 272, 274  
 Kilian, J. 64, 164, 277, 300  
 King Arthur *see* Pendragon, A.  
 Kintala, C. 29  
 Kleene hierarchy *see* hierarchy, arithmetical  
 Kleene, S. 43  
 Ko, K. 43, 63, 87, 232, 273, 282, 289, 295, 296, 308  
 Kobayashi, K. ix  
 Köbler, J. 27, 64, 88, 276, 277, 284, 289, 293, 298, 299  
 Koblitz, N. 284  
 Kolaitis, P. 108  
 Kolmogorov complexity *see* complexity, Kolmogorov  
 Kolmogorov-easy string *see* string, Kolmogorov-easy  
 Komlós, J. 280  
 Kosub, S. ix, 65, 108  
 Kozen, D. 266, 272  
 Krentel, M. 297  
 Ku, J. viii  
 Kumar, A. 289  
 Kummer, M. 296  
 Kunen, K. 265

- Kurtz, S. 108, 260, 269, 275, 282–284, 290, 293, 298
- L 82, 265, 277, 278
- Ladner, R. 265, 268, 274, 279, 308
- Lagakos, D. viii
- Lagrange Interpolation Formula *see* Formula, Lagrange Interpolation
- Landau, H. 64
- Lang, S. 163
- Lange, K. 278
- language *see* set
- canonical complete *see* set, canonical complete
  - leaf 268
- Lapidot, D. 164
- Lasser, W. 265
- Lautemann, C. 194
- leaf 170, 177–180, 198, 199, 217, 227, 230
- leaf language *see* language, leaf
- Learn, A. viii
- Lebesgue measure *see* measure, Lebesgue
- Lee, C. 193, 302
- Leiserson, C. 266
- Lemma
- Isolation 68, 70, 83, 87, 89, *see* Technique, Isolation
  - Switching 207, 232, 233
- length-decreasing function *see* function, length-decreasing
- Levin, L. 44, 64, 91, 164, 165, 267, 268
- Lewin, D. 166
- Lewis, P. 27
- lexicographic comparison *see* order, lexicographic
- lexicographic order *see* order, lexicographic
- lexmin 40, 41
- Li, L. 284
- Li, T. viii
- Lichtenstein, D. 274
- limited-nondeterminism hierarchy *see* hierarchy, limited-nondeterminism
- Lindner, W. 64, 296
- linear-sized advice *see* advice, linear-sized
- Lipton, R. ix, 20, 27, 60, 64, 163, 164, 193, 276, 277
- Liśkiewicz, M. 27
- literals
- conflicting 199
- logarithmic-space boundedness *see* boundedness, logarithmic-space
- LOGCFL 279–281
- closure properties *see* closure, of LOGCFL...
- logic
- branching time 274
  - propositional dynamic 274, 275
- logspace *see*  $C=L$ , *see* L, *see* NL, *see* PL, *see* space, logarithmic, *see* UL
- randomized 279
- logspace machine *see* machine, ...logspace...
- logspace reduction *see* reduction, ...logspace...
- logspace-uniform circuits *see* circuits, logspace-uniform
- Long, T. 64, 65, 232, 273, 282, 294, 308
- Longpré, L. 26–28, 308
- Lovász, L. 164–166
- low-degree polynomial *see* polynomial, low-degree
- lower bound *see* bound, lower
- lowness
- of sparse sets 273
- Lozano, A. 27, 277, 298
- Luby, M. 44, 164
- Luks, E. 264
- Lund, C. 163–166, 269, 270
- Lupanov, O. 232
- Lusena, C. 293
- Lynch, N. 279, 308
- MA 164, 275
- Macarie, I. viii
- machine
- alternating 271, 272
  - alternating logarithmic space 280, 281
  - bottleneck 194, 301, 303, *see* computation, bottleneck, *see* ProbabilisticSSF<sub>k</sub>, *see* SF<sub>k</sub>, *see* SSF<sub>k</sub>
  - bounded-width bottleneck 176, 181, 185, 301, 302, *see* computation, bottleneck, *see* ProbabilisticSSF<sub>k</sub>, *see* SF<sub>k</sub>, *see* SSF<sub>k</sub>
  - bounded-width probabilistic symmetric bottleneck 301, *see* ProbabilisticSSF<sub>k</sub>
  - bounded-width symmetric bottleneck 185, 186, 301, 302, *see* SSF<sub>k</sub>
  - categorical 282, 284, *see* UP
  - deterministic 214

- deterministic logarithmic space 277, 279, *see* L
  - deterministic polynomial-space 125, 176, *see* PSPACE
  - deterministic polynomial-time 1, 3, 6, 24, 112, 117, 264, 280, 285, 294, 302, 306, *see* P
  - deterministic polynomial-time oracle 19, 20, 57, 194, 214, 246, 269
  - expected-polynomial-time probabilistic 22, *see* ZPP
  - generic 275
  - multi-tape 255
  - nondeterministic 94, 121, 138, 239, 258, 266, 297, *see* NEXT, *see* NE, *see* NP, *see* tree, computation, of a nondeterministic polynomial-time Turing machine
  - nondeterministic exponential-time 23, 24, 112, *see* NEXT, *see* NE
  - nondeterministic logarithmic space 82, 277, 278, *see* NL
  - nondeterministic logarithmic space oracle 279
  - nondeterministic polynomial-time 10, 54, 57–59, 61, 62, 69, 76, 79, 80, 92–106, 113, 139, 182, 183, 186, 238, 239, 251, 260, 266, 281, 283, 285, 286, 290, 291, 294, *see* enumeration, of NPTMs, *see* enumeration, of relativized PH machines, *see* NP
  - nondeterministic polynomial-time oracle 19, 20, 76, 80, 186, 213, 269
  - nondeterministic polynomial-time, logarithmic space 252, 253
  - nondeterministic polynomial-time, logarithmic space oracle 254
  - nondeterministic space-bounded oracle 254
  - oblivious oracle NL 254, 256
  - oblivious RSTNL 256, 257
  - polynomial-space 112, 274, *see* PSPACE
  - polynomial-time oracle 115
  - probabilistic 71, *see* BPP, *see* coRP, *see* PP, *see* RP, *see* ZPP
  - probabilistic bounded-width symmetric bottleneck 302, *see* ProbabilisticSSF<sub>k</sub>
  - probabilistic logarithmic space 278, *see* PL
  - probabilistic polynomial-time 49, 70, 75, 138, 288, 290, 291, *see* BPP, *see* PP, *see* RP, *see* ZPP
  - probabilistic polynomial-time oracle 72, 75, 110, 134, 138, 142, 299
  - probabilistic polynomial-time, logarithmic space 260, *see* PL
  - probabilistic symmetric bottleneck 186, 192, 302, 303, *see* ProbabilisticSSF<sub>k</sub>
  - probabilistic Turing 74, *see* BPP, *see* PP, *see* RP, *see* ZPP
  - RSTNL 254, 255
  - symmetric bottleneck 185, 194, 302, 303, *see* SSF<sub>k</sub>
  - symmetric bounded-width bottleneck 185, *see* SSF<sub>k</sub>
  - unambiguous NP 34, *see* UP
  - universal 55
- Maciel, A. ix, 298
- Mahaney's Theorem *see* Theorem, Mahaney's
- Mahaney, S. 2, 8, 26, 27, 269, 282, 283
- MAJORITY gate *see* gate, MAJORITY
- MajSat 293
- Marchetti-Spaccamela, A. 166, 267
- Masek, W. 193
- matrix
- adjacency 82
  - determinant *see* determinant, of a matrix
  - dimension *see* dimension, of a matrix
  - lower triangular 159
  - minor *see* minor, of a matrix
  - nonsingular 149, 278
  - permanent *see* permanent, of a matrix
  - Vandermonde 148, 149, 153
- matrix inverse *see* inverse, matrix
- matrix multiplication *see* multiplication, matrix
- maximally disjoint circuit *see* circuit, maximally disjoint
- maximization 102
- maximum 99–103, 181, 183, 287
- lexicographic 9, 16
- Mayer, I. ix
- Maynard
- Brother 67
- McKenzie, P. 194, 260, 289
- measure

- Lebesgue 218
- mechanism
  - acceptance 2
- Meinel, C. 87, 277
- Melkebeek, D. van *see* van Melkebeek, D.
- membership algorithm *see* algorithm, membership
- Merkle, W. ix
- Merlin 300
- method
  - counting 218
  - culling 13, 17
  - isolation 89
  - tableau 112, 127, 139, 163, *see* Theorem, Cook's
- Meyer, A. 27, 29, 264, 270–272, 275, 277, 296
- Micali, S. 269, 300
- MIN 201, 208–210
- MINIMAL-FORMULAS 272
- minimization 102
- minimum 100–103, 287
  - lexicographic 15
- minimum weight *see* weight, minimum
- minimum-weight element *see* element, minimum-weight
- minimum-weight path *see* path, minimum-weight
- minimum-weight set *see* set, minimum-weight
- minor
  - of a matrix 112, 113, 116, 117, 119, 120
- Minsky, M. 232
- minterm 200, 201, 208, 209
- MinWeight* 68, 69, 84–86
- MinWeightSet* 68, 69
- MIP 111, 133, 134, 137, 138, 164, 165, 275, 299, 300, *see* system, interactive proof
- $\text{Mod}_k\text{P}$  87, 194, 297, 298, 301, *see* closure, of  $\text{Mod}_k\text{P}$ ..., *see*  $\oplus\text{P}$ , *see* set,  $\text{Mod}_k\text{P}$ -complete, *see* set, potential lack of sparse  $\leq_{\text{btt}}^p$ -hard, for  $\text{Mod}_k\text{P}$
- MODULO gate *see* gate, MODULO
- $\text{ModZ}_k\text{P}$  106
- monoid 168, 193, 301
  - aperiodic 193, 301
  - finite 174, 193
  - solvable 193
- monoid operation *see* operation, monoid
- morsel
  - bite-sized 24
- motto
  - of computer science 45
- Motwani, R. 165, 166, 269
- moves
  - randomized, directed by coin tosses 74
- Mukherji, P. viii
- Mulmuley, K. 87, 89
- multilinear polynomial *see* polynomial, multilinear
- multilinear testing *see* testing, multilinear
- multiparty protocol *see* protocol, multiparty
- multiple
  - of a power of 2 79
- multiplication *see* assignments, of multiplication
  - closure of  $\#P$  under *see* closure, of  $\#P$  under multiplication
- multiplication group *see* group, multiplication
- multiprover protocol *see* protocol, multiprover
- multiset 160
- multivalued function *see* function, multivalued
- multivariate polynomial *see* polynomial, multivariate
- Mundhenk, M. 27, 277, 293
- Naik, A. 29, 63–65, 88, 265, 276, 284, 295–297
- Nasipak, C. 63
- Nasser, N. 260
- natural polynomial *see* closure, of natural polynomials under multiplication, *see* polynomial, natural
- $\text{NC}$ ,  $\text{NC}^k$  167, 168, 171, 174, 176, 193, 195, 260, 261, 269, 279–281, 293, 301, 302, 308, *see* characterization, robust, of  $\text{NC}$ , *see* class, circuit
- $\text{NC}^1$  circuits *see* circuits,  $\text{NC}^1$
- NE 23–25, 28, 265, 268, 274, 275, *see* set, NE-complete
- near-testable set *see* set, near-testable
- nearly near-testable set *see* set, nearly near-testable

Neumann, J. von *see* von Neumann, J.  
 Newman's Formula *see* Formula, Newman's  
 Newman, D. 260  
 NEXP 110, 112, 133, 134, 137, 138, 145, 163–165, 274, 275, 300  
 NIA ix  
 Nickelsen, A. 297  
 nilpotent group *see* group, nilpotent  
 Nisan, N. 28, 163, 164, 270, 278, 279  
 NL 67, 68, 82–84, 87, 89, 277, 278, 280, 281, *see* set, NL-complete, with respect to 1-L reductions, *see* set, NL-complete  
 NL hierarchy *see* hierarchy, NL  
 NL/poly 68, 82, 84, 278  
 nonapproximability *see* approximation  
 – of NP optimization problems 166, 267  
 nonconstructive proof *see* proof, nonconstructive  
 nondeterminism 20, 267, *see* FewP, *see* NEXP, *see* NE, *see* NL, *see* NP, *see* UL, *see* UP  
 – ambiguous 87  
 – linear 52  
 – unambiguous 87  
 nondeterministic algorithm *see* algorithm, nondeterministic  
 nondeterministic auxiliary pushdown automata *see* automata, nondeterministic auxiliary pushdown  
 nondeterministic machine *see* machine, ...nondeterministic...  
 noninvertibility  
 – of non-honest functions 32  
 – strong 37–39, 41, 43  
 noninvertible function *see* function, ...noninvertible...  
 nonleaf *see* leaf  
 nonsingular matrix *see* matrix, nonsingular  
 nonsolvable group *see* group, nonsolvable  
 nonuniform complexity *see* complexity, nonuniform  
 nonuniform deterministic finite automata *see* NUDFA  
 nonuniform-NC<sup>1</sup> 167, 168, 171, 174  
 normalization  
 – of coin tosses 74

## Notes

– Bibliographic ix, 26, 38, 43, 63, 64, 87, 106, 163, 192, 231, 260, 308  
 NP 1–3, 5, 6, 8–11, 18–29, 31, 33–35, 39, 41, 43–45, 49–65, 67–70, 72, 73, 87, 89, 91, 93–100, 102, 104–106, 109, 131, 163–166, 185, 186, 192, 194, 197, 231, 232, 263–278, 282–287, 289, 292–294, 296–301, 305–308, *see* circuits, small for NP, *see* machine, nondeterministic polynomial-time, *see* SAT, *see* set, NP-complete ones that are non-isomorphic, *see* set, NP-complete, ones that are P-isomorphic, *see* set, NP-complete, relativizably so, *see* set, NP-complete, *see* set, possibility of NP having sparse Turing-complete, *see* set, possibility of NP having sparse Turing-hard, *see* set, potential lack of sparse  $\leq_{btt}^P$ -hard, for NP, *see* set, potential lack of sparse  $\leq_{att}^P$ -hard, for NP, *see* set, potential lack of sparse  $\leq_m^P$ -complete, for NP, *see* set, potential lack of sparse  $\leq_r^P$ -complete, for NP, *see* set, potential lack of tally  $\leq_m^P$ -complete, for NP, *see* set, potential lack of tally  $\leq_m^P$ -hard, for NP, *see* set, sparse, in NP, *see* set, sparse, in P, *see* tree, computation, of a nondeterministic polynomial-time Turing machine  
 – closure properties *see* closure, of NP...  
 NP-hard 3, 6, 8, 9, 11, 19–22, 26, 27, 29, 60, 268, 296, *see* hardness, NP...  
 NP-hardness *see* hardness, NP..., *see* NP-hard  
 NP-hardness of approximation *see* approximation, NP-hardness of  
 NP-printable set *see* set, NP-printable  
 NP-selective set *see* set, NP-selective  
 NP/linear 49, 51, 52, 63, 296  
 NP/poly 61, 64, 276, 296  
 NPFewV 65  
 (NP  $\cap$  coNP)/poly 59–62, 64, 276, 296  
 NPkV 65  
 NPMV 57–59, 61, 63–65, 89, 108, 291–294, 306, *see* function, NPMV  
 NPSV 57–59, 61–65, 291–295, 297, *see* refinement, NPSV



- NPSV-sel 59, 61, 62, 296, *see* function, NPSV-selector  
 NPTM *see* machine, nondeterministic polynomial-time  
 NSF ix  
 NTIME 266, 267, 274, 275  
 NUDFA 193, 301  
 number  
   – of primes 122  
  
 oblivious machine *see* machine, oblivious...  
 obliviousness  
   – of selector functions 297  
 Ogihara, Ellen ix  
 Ogihara, Emi ix  
 Ogihara, Erica ix  
 Ogihara, M. iv, ix, 1, 26, 27, 29, 63–65, 87, 106–108, 193, 194, 260, 261, 265, 269, 276–279, 287, 290, 293–298, 301, 303, 308  
 Ogihara–Watanabe Theorem *see* Theorem, Ogihara–Watanabe  
 Ogiwara, M. *see* Ogihara, M.  
 one-sided error *see* error, one-sided  
 One-Way Conjecture *see* Conjecture, One-Way  
 one-way function *see* function, ...one-way...  
 operation 92, 99, 107  
   – 1-ary 100  
   – 2-ary 100  
   – associative 168, 169  
   – boolean 280  
   – monoid 169  
   – multi-argument 107  
   – one-argument 107  
   – polynomial-time computable 92, 93, 95–99, 101, 104, 105, 108, 287  
 operator  
   – BP 87, 88, 301  
   – R 87, 88, 297  
 optimal advice *see* advice, optimal  
 optimization function *see* function, optimization  
 optimization problem *see* problem, NP optimization  
 OptP 103–105, 107, 108, 181–183, 297, 299, *see* function, OptP  
   – closure properties *see* closure, of OptP...  
 OR circuits *see* circuits, OR  
 OR gate *see* gate, OR  
  
 OR-AND circuits *see* circuits, OR-AND  
 oracle 19–21, 28, 29, 57, 64, 72, 75, 81, 87, 107, 115, 121, 123, 132, 134–138, 141, 142, 144, 146, 147, 149, 151, 152, 155, 163–165, 187, 192, 197, 207, 213, 214, 216, 218, 222, 223, 228, 231, 246, 247, 249–252, 254–258, 268–270, 282, 284, 285, 293, 306, 308, *see* separation, by an oracle, *see* world, relativized  
   –  $C=P$  192  
   –  $\oplus P$  76  
   – PSPACE-complete 197  
   –  $\#P$  80  
   – NP 1, 19, 89  
   – perm 115–117  
   – PH 194  
   – PL 256  
   – random 64, 89, 219, 232, 268, 269, 273, 282, 300  
   – sparse 21  
 oracle construction *see* construction, oracle  
 oracle function *see* function, oracle  
 oracle gate *see* gate, oracle  
 oracle machine *see* machine, ...oracle...  
 order  
   – among the variables 199, 200  
   – lexicographic 5, 7, 9, 12, 14–16, 36, 40, 42, 55–57, 80, 91, 100, 101, 163, 194, 228, 238, 265  
   – of a group 112  
   – of all possible moves 138  
   – of instructions 181, 185  
   – of the nodes in a graph 82  
   – of the non-appendix chapters viii  
 Othello 272  
 overhead  
   – quadratic 55  
  
 P 1, 2, 5, 8–12, 18, 19, 22–29, 33–36, 39–41, 43, 44, 47–49, 57, 60, 63, 68, 70, 72–78, 80–82, 87, 91, 93, 95, 97–99, 106, 107, 110, 112, 114, 115, 119, 139, 163–165, 183, 187, 194, 197, 231, 232, 236, 240, 252, 259, 263–277, 282–287, 289, 293–298, 300, 305, 306, 308, *see* machine, deterministic polynomial-time, *see* set, potential lack of sparse  $\leq_{btt}^P$ -hard, for P  
 P-capturable set *see* set, P-capturable  
 P-close set *see* set, P-close

- P-immune *see* immunity
- P-immunity *see* immunity
- P-isomorphism 26, 31, 269, 278, 282
- P-sel 47–49, 51, 52, 56, 63, 64, 276, 294–296, 298, *see* circuits, small, for the semi-feasible sets, *see* function, P-selector
- closure properties *see* closure, of P-sel...
- P-selector function *see* function, P-selector
- P-uniform circuits *see* circuits, P-uniform
- P-uniformAC<sup>1</sup>(C=P) 293
- P-uniformAC<sup>k</sup>(C=P) 308
- P-uniformNC<sup>1</sup>(C=P) 293
- P-uniformNC<sup>k</sup>(C=P) 308
- P/linear 48, 63
- P/poly 22, 27, 47, 48, 60, 63, 75, 76, 78, 87, 164, 263, 275–277, 289, 296
- P/quadratic 48, 49, 63, 276, 296
- padding *see* set, paddable, *see* set, padded version of, *see* translation, via padding
- of strings 49
- pair
- matrix-integer 115, 117
- pairing function *see* function, pairing, *see* function, standard pairing
- Papadimitriou, C. ix, 87, 271, 272, 274, 275, 297, 298
- Papathanasiou, T. ix
- Papert, S. 232
- parallel access to NP *see* access, parallel, to NP
- parallel queries *see* queries, parallel
- parameterized strategy *see* strategy, parameterized
- Parberry, I. 297, 298
- parity
- of #acc 76, 77
- of a number 239
- parity circuits *see* approximation, of threshold circuits by parity circuits, *see* circuits, parity
- parity exponential time *see* time, parity exponential
- parity function *see* function, parity
- PARITY gate *see* gate, PARITY
- Parkins, K. 63
- partial function *see* function, partial
- partition
- of potential queries 216, 222, 229
- of variables 224
- Pasanen, K. 43
- Paterson, M. 296
- path *see* reduction, of the number of accepting paths
- downward 174, 177, 179, 198
- minimum-weight 83, 84, 86
- nondeterministic 62, *see* machine, nondeterministic polynomial
- rejecting computation 59, 99, 101, 105, 235, 236, 238, 294, *see* #rej<sub>N</sub>(x)
- unique accepting computation 281, 285
- unique minimum-weight 84
- unique successful computation 86
- paths
- shortness of, in a tournament 51
- Paturi, R. 260
- Paul, W. 193
- PBP 167, 168, 171, 174, 193, 300–302
- width of *see* width, of PBP
- P<sup>C=P</sup> *see* set, P<sup>C=P</sup>-complete
- PCP Theorem *see* Theorem, PCP
- PCP(*f*(*n*), *g*(*n*)) 165, 269
- pebble 167, 232
- pebbling game *see* game, pebbling
- Pendragon, A. 67, 300
- perceptron 232
- perm 113–122, *see* permanent
- permanent *see* perm, *see* protocol, for permanent
- of a matrix 112, 113, 115, 118, 119
- permanent function *see* function, permanent
- permutation 113, 114, 121, 172, 185, 188–190, 192, 193, 302
- permutation group *see* group, permutation
- permutation group membership *see* generators, permutation group membership from
- PH 19, 20, 22, 27, 43, 58, 60, 61, 64, 67, 68, 73, 78, 81, 82, 87, 88, 97, 101, 105, 115, 119, 164, 194, 197, 207, 213, 218, 219, 221–223, 228, 231, 232, 259, 271–273, 276, 286, 287, 289, 293, 294, 297, 298, 301, *see* hierarchy, polynomial, *see* set, sparse, in PH
- exponential-time analog of *see* analog, exponential-time, of the polynomial hierarchy
- Pinheiro, E. ix
- Pippenger, N. 279–281

- Pitassi, T. 232, 233
- PL 82, 252, 254, 256, 260, 278–280, *see*  
 set, canonical complete for PL, *see*  
 set, PL-complete
- closure properties *see* closure, of  
 PL...
- PL hierarchy *see* hierarchy, PL
- PL oracle hierarchy *see* hierarchy,  
 oracle, of PL
- PLH 254
- $P^{NP|O(\log n)|}$  1, 19, *see*  $\Theta_k^p$
- polylog-depth circuits *see* circuits,  
 polylog-depth constant-size, *see*  
 circuits, polylog-depth polynomial-  
 size
- polynomial *see* specification, unique,  
 of a polynomial by coefficients, *see*  
 specification, unique, of a polynomial  
 by points, *see* specification, unique,  
 of a polynomial
- increasing 75
- low-degree 111, 112, 242, *see* char-  
 acterization, robust, of low-degree  
 polynomials
- multilinear 141, 164
- multivariate 111, 112, 141, 156, 235
- natural 247, 248, 251, 255–257
- nonzero 111, 112, 150, 151
- root of *see* root, of a polynomial
- strictly increasing 11, 75, 184, 247
- two-variable 242
- univariate 111
- polynomial hierarchy *see* hierarchy,  
 polynomial
- polynomial interpolation *see* Tech-  
 nique, Polynomial Interpolation
- polynomial machine *see* machine,  
 ...polynomial...
- polynomial-depth circuits *see* circuits,  
 polynomial-depth
- polynomial-size circuits *see* circuits,  
 polynomial-size
- polynomial-space boundedness *see*  
 boundedness, polynomial-space
- polynomial-time algorithm *see*  
 algorithm, polynomial-time
- polynomial-time boundedness *see*  
 boundedness, polynomial-time
- polynomial-time computable enumer-  
 ator *see* enumerator, polynomial-  
 time computable
- polynomial-time computable function  
*see* function, polynomial-time  
 computable
- polynomial-time computable operator  
*see* operator, polynomial-time  
 computable
- polynomial-time predicate *see*  
 predicate, polynomial-time
- polynomial-time reduction *see*  
 reduction, ...polynomial-time...
- polynomial-time relation *see* relation,  
 polynomial-time
- Pomerance, C. 277
- position
- input-tape head 255
- work-tape head 255
- Post, E. 264
- power
- relative, of different complexity  
 classes 22
- PP 49, 50, 67, 68, 78, 80–82, 87, 94–99,  
 101, 105–107, 112, 119, 186, 232,  
 235–237, 239–242, 244, 245, 247–252,  
 259, 260, 263, 273, 279, 286, 287,  
 290–293, 296–298, 300, 301, *see* class,  
 counting, *see* set, PP-complete
- closure properties *see* closure, of  
 PP...
- PP/linear 49, 50
- $P^{PP}$  *see* set, canonical complete for  
 $P^{PP}$
- Pr 135, 187–189, 191, 208–210, 278,  
 288, 291, 302
- Pratt, V. 163, 274
- predecessor 180, 194
- predicate
- polynomial-time 95, 102, 103, 267,  
 271, 283, 288, 292, 298
- preimage 34, 42
- primality *see* certificate, of primality,  
*see* number, of primes, *see* set, of all  
 prime numbers, *see* set, of primes
- complexity of 99, 267, 277, 284, 289,  
 290
- complexity of certificate checking  
 131, 132, 142
- Prime Number Theorem *see*  
 Theorem, Prime Number
- Principle
- Pigeonhole 222, 233
- probabilistic circuits *see* circuits,  
 probabilistic

- probabilistic function *see* function, probabilistic
- probabilistic logspace hierarchy *see* PLH
- probabilistic machine *see* machine, ...probabilistic...
- probabilistic oracle protocol *see* protocol, probabilistic oracle
  - NC<sup>1</sup>
- closure properties *see* closure, of probabilistic-NC<sup>1</sup>...
- probabilistic-NC<sup>1</sup> 260
- Probabilistic-PSPACE 273
- probabilistically checkable proof *see* proof, probabilistically checkable
- ProbabilisticSSF<sub>k</sub> 186, 188, 192, 301, 302
- probability
  - acceptance 116, 117, 123, 124, 134–136, 289
  - maximum acceptance 116, 124
- probability distribution *see* distribution, probability
- Problem
  - Graph Accessibility 82, 89, *see* GAP
  - Graph Isomorphism 267, 269, 289
  - Unique Optimal Traveling Salesperson 272
- problem *see* set
  - NP optimization 166, 266, 267
  - adversary 274
  - boolean formula minimization 272
  - canonical complete *see* set, canonical complete
  - evaluation, of polynomials over integer matrices 278
  - game 274
  - game-based 274
  - pebbling 264
  - PSPACE 124
  - robotics configuration space 264
  - word 193
- procedure
  - enumeration 12, 13
  - interval-pruning 1, 12–18
  - nondeterministic logarithmic space 84, *see* NL
  - polynomial-time 12, 13, *see* FP, *see* P
  - polynomial-time search 12
  - self-reducibility-based tree-pruning 2
    - tree-pruning 1–3, 6
- product
  - direct 193
  - of all prime divisors 298
  - partial 175
- program
  - bounded-width branching 167, 168, 193, 194, 300–302
  - bounded-width-branching *see* PBP
  - branching 172, 193, *see* PBP
  - branching, generalizing the notion 174
  - over a monoid 193
  - straight-line 164, 194
- programs *see* machine, *see* PBP, *see* procedure
  - bounded-width permutation-only branching 194
  - bounded-width polynomial-size branching 168, 169, 172, 173, 301
  - polynomial-size, permutation-only branching 193
- projection function *see* function, projection
- promise
  - in the definition of SPP 290
  - in the definition of UP 285
- proof
  - census-based 20
  - deterministically verifiable 109
  - Frege 233
  - nonconstructive 198
  - probabilistically checkable 267, *see* PCP( $f(n)$ ,  $g(n)$ )
  - relativizable 60
- proper decrement *see* decrement, proper
- proper subtraction *see* subtraction, proper
- property *see* closure
  - $\leq_T^P$ -hardness, of PP for PH 50, 67
  - closure, hardest 93
  - closure, of C=L 261
  - closure, of #P 94, 98, 99
  - closure, of #P involving binomial and multinomial coefficients 93
  - intermediate closure 99
  - NP-completeness, of SAT under  $\leq_m^P$ -reduction 61
  - one-argument, of GapP 108
  - one-to-one, of one-way functions 34, 44

- polynomial-time computable closure 107
- polynomial-time computable closure, of OptP 104
- propositional dynamic logic *see* logic, propositional dynamic
- Protasi, M. 166, 267
- protocol 110, 112, 115–119, 123, 125, 126, 129, 131, 136, 147–149, 153, 165, *see* completeness, of a protocol, *see* interactive protocol, *see* soundness, of a protocol, *see* system, interactive proof
  - building block for 31, 36, 43
  - cryptographic 31, 36
  - digital signature 43
  - for an interactive proof system 109
  - for permanent 110, 115, 116
  - for PSPACE 132
  - for reachability 126
  - fully parallelized multiprover interactive proof 164
  - multiparty 36
  - multiprover 164
  - one-round perfect-zero-knowledge 164
  - one-sided-error PCP 166
  - probabilistic oracle 134, 135, 138, 142, 164, 165
  - probabilistic polynomial-time protocol 269
  - two-prover 135
  - two-sided-error PCP 166
- prover 109–111, 115, 116, 118, 119, 123–125, 131–136, 163, 165, 299, 300, *see* system, interactive proof
  - deterministic 123, 134, 136
  - power of 109
- pruning
  - tree 5, 6
- pseudorandom generator *see* generator
- PSPACE 22, 23, 87, 110, 112, 122, 124–126, 133, 163, 164, 176, 177, 181, 194, 197, 207, 213, 218, 219, 222, 231, 232, 263, 270–275, 285, 287, 293, 300, 301, 303, *see* protocol, for PSPACE, *see* QBF, *see* set, PSPACE-complete, *see* tree, computation, of a deterministic polynomial-space Turing machine
- pushdown automata *see* automata, pushdown
- Python, M. 67
- QBF 176, 177, 181, 274, *see* PSPACE
- quadratic bound *see* bound, quadratic
- quantified boolean formula *see* formula, quantified boolean
- quantifier
  - block, existential or universal 272
- quantifier switching *see* switching, quantifier
- quantifiers
  - alternating 221, 271, 272, 274
- quaternary tree
  - full *see* tree, full quaternary
- qubit 194, 195
- queries
  - parallel 250
- query generator *see* generator, query
- query round *see* round, query
- query simulation algorithm *see* algorithm, query simulation
- query state *see* state, query
- query tape *see* tape, query
- Rabi, M. 36, 43, 44
- Rabin, M. 264, 277
- Rackoff, C. 87, 279, 284, 300
- Ramachandran, A. 29, 231, 269
- random bits *see* bits
- random oracle *see* oracle, random
- random restriction *see* restriction, random
- random self-reducibility *see* self-reducibility, random
- random variable *see* variable, random
- randomized algorithm *see* algorithm, randomized
- randomized moves *see* moves, randomized, directed by coin tosses
- randomness 67, 109, 110, 186, 299
- range 32, 34, 35, 37, 38, 282, 284, 286
- Ranjan, D. 231, 270, 300
- rank
  - of a string 80, 114, 169, 181–185, 187, 188, 191, 236, 238
- rankable set *see* set, rankable
- ranking function *see* function, ranking
- Rao, R. 28
- $R_a^b(C)$ ,  $R_a^b(C)$  18, 19, 22, 27, 60, 64, 269, 271, 273, 293, 296, 298, 307, 308
- Reach 84–86
- reachability 50, 51, 82, 84, 112, 129, *see* protocol, for reachability
- ReachTest 85, 86

- recurrence relation *see* relation, recurrence
- recursion
  - bounded depth of *see* bounded, depth of recursion
- recursive call *see* call, recursive
- recursive function *see* function, recursive
- recursive set *see* set, recursive
- reducibility closure *see*  $R_a^b(C)$ ,  $R_a^b(C)$
- reduction
  - $\leq_m^P$ , of languages in  $SSF_k$  301
  - 1-L 278
  - as a means for studying relative hardness of sets 305
  - between functions 121
  - coNP-many-one 241, 306, 308, *see*  $\leq_m^{conp}$
  - coNP-many-one, closure of  $coC=P$  downward under *see* closure, of  $coC=P$  downward under coNP-many-one reductions
  - coNP-many-one, closure of coNP downward under *see* closure, of coNP downward under coNP-many-one reductions
  - constant-round truth-table, closure of PP downward under *see* closure, of PP under constant-round truth-table reductions
  - Cook's 305, 308
  - logspace bounded-truth-table 265
  - logspace many-one 306, *see*  $\leq_m^L$
  - logspace-uniform  $AC^k$  308
  - logspace-uniform  $NC^k$  308
  - $NC^1$  many-one 308
  - of error-probability in BPP computation 73
  - of numbers of solutions 108
  - of numbers of witnesses 108
  - of the cardinality of a set of vectors 88
  - of the cardinality of acceptance types 108
  - of the depth of a circuit 206
  - of the number of accepting paths 94
  - of the number of outputs 292
  - of the number of solutions 65, 67
  - one-way logspace 278
  - P-uniform  $AC^k$  308
  - P-uniform  $NC^1$ , closure of PL downward under *see* closure, of PL downward under P-uniform  $NC^1$  reductions
  - P-uniform  $NC^k$  308
  - polynomial-time bounded-truth-table 2, 9, 18, 26, 96, 245, 306, *see*  $\leq_{btt}^P$
  - polynomial-time conjunctive truth-table 240, 241, 305, *see*  $\leq_{ctt}^P$
  - polynomial-time conjunctive Turing 306, *see*  $\leq_c^P$
  - polynomial-time conjunctive-truth-table 26
  - polynomial-time constant-round truth-table 240, 249–251, *see*  $\leq_{tt[k]}^P$
  - polynomial-time disjunctive truth-table 240, 305, *see*  $\leq_{dtt}^P$
  - polynomial-time disjunctive Turing 306, *see*  $\leq_d^P$
  - polynomial-time disjunctive-truth-table 26
  - polynomial-time locally positive Turing 306, *see*  $\leq_{locpos}^P$
  - polynomial-time many-one 3, 6, 8, 9, 61, 71, 305, *see*  $\leq_m^P$
  - polynomial-time parity, closure of PP downward under *see* closure, of PP downward under polynomial-time parity reductions
  - polynomial-time positive Turing 306, *see*  $\leq_{pos}^P$
  - polynomial-time randomized *see*  $\leq_{randomized}^P$
  - polynomial-time truth-table 246–248, 305, *see*  $\leq_{tt}^P$
  - polynomial-time Turing 18, 259, 305, *see*  $\leq_T^P$
  - polynomial-time two-round truth-table 250, 251, *see*  $\leq_{tt[k]}^P$
  - positive truth-table, closure of  $C=P$  downward under *see* closure, of  $C=P$  under positive truth-table reductions
  - randomized 67, 69, 70
  - randomized, of NP-language to USAT 70
  - strong nondeterministic *see*  $\leq_T^{sn}$
  - witness 93–95, 106, 108
- refinement
  - $FP_{tt}^{NP}$  64, 89
  - $FP_{tt}^{UP}$  293

- NPFewV 65
- NP $\leq$ V 65
- NPSV 58, 59, 61, 63, 65, 292–294
- of a multivalued function 58, 64, 292, 294
- of a set of intervals 12–17
- of a truth-table condition 14
- Regan, K. viii, 87, 88, 106, 285
- Reingold, N. 89, 260
- Reinhardt, K. 89
- Reischuk, R. 27
- Reith, S. 277
- rejecting computation path *see* path, rejecting computation
- rejection cardinality *see* cardinality, rejection
- relation
  - polynomial-time 266, 287
  - recurrence 47
  - relativizable 213, 260
- relativizable result *see* result, relativizable
- relativization 60, 73, 78, 231
  - by sparse oracles 273
  - RST *see* relativization, Ruzzo–Simon–Tomp
  - Ruzzo–Simon–Tomp 82, 254, 260, 268–270, 279
- relativizations, conflicting *see* results, conflicting oracle
- relativized world *see* world, relativized
- replacement
  - of provers with an oracle 134
  - of the oracle 75
- research
  - esoteric vii
- restriction 198–200, 202, 204, 206–212, 224, 225, 227, 228, *see* disjointness, of restrictions, *see* distribution, probability, of restrictions
- disjoint pair 200
- empty 200
- product 200, 224
- random 202, 204, 224, 225
- size of 200
- result
  - partial 100, 107
  - relativizable 60, 72, 73, 75
- results
  - conflicting oracle 87, 197, 270
- Rettinger, R. ix, 27
- reversal
  - of the accepting and rejecting path behavior 100
- Rice's Theorem *see* Theorem, Rice's
- Rice, H. 285
- ring
  - cyclic 193
  - zero-divisor of *see* zero-divisor, of a ring
- Rivest, R. 36, 43, 266
- RL 279
- Roche, J. ix
- Rochester ix
  - University of viii
- Rogers, H., Jr. 271
- Rogers, J. 65, 282, 284
- Rohatgi, P. 89, 231, 270
- Rompel, J. 163
- root
  - of a polynomial 111–113, 150, 151, 207, 208, 211
  - of a tree 5, 7, 92, 180
- Rossmanith, P. 278
- Rothe, J. ix, 28, 43, 44, 63, 272, 284, 285, 287, 297
- round
  - of the integer sampling algorithm 131
  - parallel query 251
  - query 249–252
  - simulation 136, 137
- Royer, J. 65, 269, 275, 282–284
- Rozenberg, G. ix
- RP 28, 29, 72, 87, 268, 288–290, 296
  - exponential-time analog of *see* analog, exponential-time, of UP, FewP,  $\oplus$ P, ZPP, RP, or BPP
- RP operator *see* operator, RP
- $R_{tt}^p(\text{NP})$  307
- $R_{bt}^p(\text{P-sel})$  296, 298
- $R_{ct}^p(\text{SPARSE})$  307, 308
- $R_{ctt}^p(\text{TALLY})$  307, 308
- $R_{k-T}^p(\text{NP})$  269
- $R_{k-T}^p(\text{P-sel})$  296
- $R_{k-T}^p(\Sigma_i^p)$  273
- $R_{k-tt}^p(\text{NP})$  269
- $R_{k-tt}^p(\text{P-sel})$  296
- $R_{\mathcal{O}(\log n)-T}^p(\text{NP})$  271, 273
- $R_{\mathcal{O}(\log n)-T}^p(\Sigma_k^p)$  271
- $R_{\mathcal{O}(n)-T}^p(\text{P-sel})$  64
- $R_{pos}^p(\text{NP})$  307
- $R_{tt}^p(\text{SPARSE})$  22, 27
- $R_{tt}^p(\text{C=P})$  260, 293
- $R_{tt}^p(\text{NP})$  18, 19

- Rubinfeld, R. 163, 164, 284  
 Rubinstein, R. 43, 283–285  
 Rudich, S. 265, 286  
 Rumely, R. 277  
 runtime 19, 20, 24, 34, 76, 80, 123, 124, 138, 183, 186–188, 214, 236, 256  
 Russell, A. 27, 164  
 Russo, D. 260  
 Ruzzo, W. 82, 254, 260, 265, 279–281  
  
 s-honest function *see* function, s-honest  
 Sabadini, N. 286  
 SAC,  $SAC^k$  279–281, *see* class, circuit  
 – closure properties *see* closure, of  $SAC^k$ ...  
 safe storage *see* storage, safe  
 Safra, S. 164–166  
 Saks, M. 260  
 Salomaa, A. ix  
 Samorodnitsky, A. ix, 166  
 sampling algorithm *see* algorithm, sampling  
 Santha, M. 279  
 SAT 1, 3–9, 19–22, 26, 57–61, 87, 89, 91, 93, 95, 98, 267–270, 273, 276, 282, 284, 287, 289, 296, 305, *see* circuits, for SAT, size of, *see* NP  
 satisfiable formula *see* formula, satisfiable boolean  
 satisfying assignment *see* assignment, satisfying, *see* assignments  
 Savage, J. 276  
 Savitch's Theorem *see* Theorem, Savitch's  
 Savitch, W. 89, 125, 163, 273  
 Saxe, J. 232  
 SC 279  
 Schaefer, M. ix, 272  
 Schaefer, T. 272  
 Schear, M. ix  
 scheme  
 – oracle construction 213, 215  
 – pruning 5  
 Scherer, B. viii  
 Schöning, U. ix, 27, 87, 269, 273, 276, 277, 284, 286, 287, 289, 293, 296, 298, 299  
 Schulman, L. 194  
 Schwartz, J. 163, 264  
 Schwentick, T. 88, 194  
 Scully, V. 263  
 search  
 – brute-force 53, 54  
 – exhaustive 264  
 search procedure *see* procedure, polynomial-time search  
 second 37, 39, 42  
 secret  
 – of complexity theory vii  
 – real, of complexity theory vii  
 secret-key agreement *see* agreement, secret-key  
 Seiferas, J. viii, ix, 275, 279  
 selection  
 – random, of an oracle 218  
 – under uniform distribution 71, 80, 81, 116, 118, 131–133, 146, 154, 160–162, 187, *see* distribution, uniform  
 selectivity 62, 64, *see* function, NPSV-selector, *see* function, P-selector, *see* function, P-sel, *see* function, selector, oblivious to the order of its argument, *see* function, selector, symmetric, *see* function, selector, *see* NPSV-sel, *see* P-sel, *see* set, semi-feasible-sel  
 – counting-class-based 64  
 – importance of 295  
 – nondeterministic and other analogs of P-selectivity 64  
 selector function *see* function, selector  
 self-correction 164  
 self-reducibility 4, 5, 60, 119  
 – disjunctive *see* tree, disjunctive self-reducibility  
 – disjunctive, of SAT 1  
 – downward 164  
 – many-one 194  
 – of SAT 2  
 – random 164  
 – tree *see* tree, self-reducibility  
 self-reducibility algorithm *see* algorithm, self-reducibility  
 self-reducibility-based argument *see* argument, self-reducibility-based  
 self-reducibility-based tree-pruning approach *see* approach, self-reducibility-based true-pruning  
 self-testing 164  
 Selman, A. viii, ix, 43, 44, 63–65, 268, 273, 276, 282–284, 294–297, 308  
 semi-feasible set *see* set, semi-feasible  
 semi-membership algorithm *see* algorithm, semi-membership



semi-recursive set *see* set, semi-recursive

Sengupta, S. ix, 27, 164

separation

- by an oracle 231, 232, 259, 272, 273, 284, 293, 297, 298, *see* oracle
- downward 273
- probability one 284, 300

set

- 2-disjunctively self-reducible 60
- almost polynomial-time 296
- $C=L$ -complete 278
- $C=P$ -complete 293
- canonical complete for PL 278
- canonical complete for  $P^{PP}$  293
- cofinite 219
- complete for the  $\leq_t^L$ -reducibility closure of  $C=L$  279
- complete, for levels of PH 272
- coNP-complete 2, 5
- context-free 279, 280
- $\Delta_2^P$ -complete 272
- dense 26, 265
- disjunctively self-reducible 60
- E-complete 275
- EXP-complete 274, 275
- $\mathcal{F}$ -selective 59
- finite 68, 219
- $FP_{total}$ -selective 59
- hardness for classes *see* hardness, of sets, classifying via reductions
- $k$ -locally self-reducible 194
- minimum-weight 68, 69
- $Mod_k P$ -complete 298
- NE-complete 275
- near-testable 296
- nearly near-testable 296
- NL-complete 82, 83, 89
- NL-complete, with respect to 1-L reductions 278
- NP-bounded-truth-table-complete *see* set, NP-complete
- NP-complete 1–3, 8, 9, 18–20, 22, 23, 26, 27, 31, 60, 91, 93, 95, 98, 99, 266–269, 282, 286, 287, 305
- NP-complete ones that are non-isomorphic 284, 285
- NP-complete, ones that are P-isomorphic 282
- NP-complete, relativizably so 60
- NP-conjunctive-truth-table-complete *see* set, NP-complete
- NP-hard 1, 3, 8, 9, 18, *see* NP-hard

- NP-hard, sparse *see* NP-hard

- NP-many-one-complete *see* set, NP-complete

- NP-printable 275

- NP-Turing-complete *see* set, NP-complete

-  $NP \cap coNP$ -complete 269

- NPSV-selective 295–297

- of all prime numbers 131

- of primes 277

- P-capturable 6, 26

- P-close 296

- P-complete 265

- P-printable 28, 284, 285

- P-selective 47, 59, 64, 294–297, *see* P-sel

- paddable 269, 282

- padded version of 251

-  $\oplus P$ -complete 298

-  $P^{C=P}$ -complete 293

- PL-complete 278

- polynomial-time 264, *see* P

- possibility of NP having sparse Turing-complete 27

- possibility of NP having sparse Turing-hard 27

- potential existence of sparse, in NP–P 23, 275

- potential existence of tally, in NP–P 23, 24, 275

- potential lack of sparse  $\leq_T^P$ -hard, for UP 284

-  $Mod_k P$  298

- potential lack of sparse  $\leq_{btt}^P$ -hard, for NP 9, 268

- potential lack of sparse  $\leq_{btt}^P$ -hard, for P 265

- potential lack of sparse  $\leq_{dtt}^P$ -hard, for NP 268

- potential lack of sparse  $\leq_m^P$ -complete, for coNP 5

- potential lack of sparse  $\leq_m^P$ -complete, for NP 8

- potential lack of sparse  $\leq_m^P$ -hard, for coNP 5

- potential lack of sparse  $\leq_T^P$ -complete, for NP 1, 18, 19, 268

- potential lack of sparse  $\leq_T^P$ -hard, for NP 1, 20, 22, 60, 268

- potential lack of tally  $\leq_m^P$ -complete, for NP 2

- potential lack of tally  $\leq_m^P$ -hard, for NP 2

- PP-complete 293
- PSPACE-complete 87, 176, 194, 197, 272, 274
- rankable 265
- recursive 264
- self-reducible 60
- semi-feasible 45, 47–49, 51–54, 57, 59, 63, 64, *see* selectivity
- semi-feasible, nondeterministic analog of 57
- semi-recursive 295
- $\#L$ -complete 279
- $\#P$ -complete 115, 119, 286, 287
- $\Sigma_2^P$ -complete 272
- small advice 47
- sparse 1–29, 269, 276, 277, 307, 308, *see* relativization, by sparse oracles, *see* SPARSE
- sparse, in NP 268, 274, 275
- sparse, in P 284, 285
- sparse, in PH 275
- supersparse 28
- tally 2, 3, 5, 6, 23, 24, 26, 274, 287, *see* TALLY
- $\Theta_2^P$ -complete 272
- Turing self-reducible 296
- UL-complete 278
- UP-hard 283, 284
- US-complete 69
- weakly P-selective 63
- ZPP-hard 288
- set-f 57–59, 61, 62, 65, 291, 292, 294, 295, 297, 306
- sets
  - disjoint 185, 246
  - P-isomorphic *see* P-isomorphism
  - sparse, lowness of *see* lowness, of sparse sets
- Sewelson, V. 22, 24, 26–29, 106, 107, 274, 275
- $SF_k$  176, 177, 181, 183, 185, 194, 300–303, *see* computation, bottleneck, *see* machine, bottleneck
- closure properties *see* closure, of  $SF_5$ ...
- Shamir, A. 163, 164, 270
- Sharir, M. 264
- Sherman, A. 36, 43, 44
- Sheu, M. 232
- Siefkes, D. 269
- sign function *see* function, sign
- Silvestri, R. 27, 269, 277, 285
- Simon, I. 279, 293
- Simon, J. 82, 254, 260, 273, 279, 290, 293
- simulation
  - nondeterministic, of an oracle 76
  - probabilistic, of an oracle 75
- simulation round *see* round, simulation
- single-valuedness
  - of FP 291
- Sipser, M. 163, 232, 264, 265, 269, 274, 285, 286, 288, 300
- Sivakumar, D. viii, 29, 88, 265
- Skyum, S. 302
- small circuits *see* circuits, small
- Smith, C. ix
- Solovay, R. 231, 268–270, 272, 273
- solution *see* reduction, of numbers
  - of solutions, *see* reduction, of the number of solutions
  - type *see* type, solution
  - unique 67, 68
- solvable monoid *see* monoid, solvable
- sorcerer
  - pointy-hatted vii
  - pointy-headed vii
- soundness
  - of a protocol 110, 115, 116, 118, 123, 132–138, 145, 149, 155, 299
- $S_2^P$  27, 64, 164, 268
- $(S_2^P)^{NP} \cap coNP$  64
- space
  - exponential 264
  - logarithmic *see*  $C=L$ , *see* L, *see* NL, *see* PL, *see* UL
  - polynomial *see* PSPACE
- space hierarchy theorem *see* theorem, space hierarchy
- SpanP 104, 105, 107, 108, 297, 299
  - closure properties *see* closure, of SpanP...
- SPARSE 273, 276, 307, 308
- sparse P superset *see* superset, sparse P
- sparse set *see* set, sparse
- specification
  - unique, of a polynomial 111
  - unique, of a polynomial by coefficients 148
  - unique, of a polynomial by points 157
- Spielman, D. 89, 260
- splitting
  - of intervals 13

- SPP 100–103, 105, 106, 284, 287,  
290–293, 298, *see* class, counting  
– promise *see* promise, in the  
definition of SPP
- Srinivasan, A. 89
- SSF<sub>k</sub> 185, 194, 300–302, *see* machine,  
symmetric bottleneck
- state  
– query 110  
– random starting 195  
– unique accept 127
- Stearns, R. 27, 264, 275
- Stein, C. 266
- Stephan, F. 285, 296
- Stockmeyer, L. 29, 264, 270–274, 279,  
281, 286, 308
- Stoness, S. ix
- storage  
– safe 302  
– value of 302
- straight-line program *see* program,  
straight-line
- strategy  
– parameterized 136
- stratified circuits *see* circuits,  
stratified
- Straubing, H. viii, 193, 194, 281
- string  
– Kolmogorov-easy 265  
– tally 5
- strong exponential hierarchy *see*  
hierarchy, strong exponential
- strongness  
– of length-based honesty 37
- subcircuit 198, 203–206, 208, 211–213,  
217, 223–225, 227, 228, 230, *see*  
circuit
- subcircuits  
– maximally disjoint 206
- subgroup 174  
– commutator 167, 174, 175
- sublinear parallel access to NP *see*  
access, sublinear-parallel, to NP
- subroutine  
– polynomial-time 264  
– unit-cost 269
- subtraction  
– integer 38  
– proper 91–96, 98, 99, 104–107
- succinct certificate *see* certificate,  
succinct
- Sudan, M. ix, 163–166, 267, 269
- Sudborough, L. 280, 281
- Sundaram, R. 27, 164
- Sundell, S. ix
- superpolynomial-size circuits *see*  
circuits, superpolynomial-size
- superset  
– sparse P 26
- supersparse set *see* set, supersparse
- survey  
– amusing, of the research leading to  
IP=PSPACE 164
- Swier, R. viii
- switching  
– quantifier 77
- Switching Lemma *see* Lemma,  
Switching
- symmetric alternation *see* alternation,  
symmetric, *see* S<sub>2</sub><sup>P</sup>
- symmetric bottleneck machine *see*  
machine, ...symmetric bottleneck...
- symmetric difference *see* difference,  
symmetric
- symmetric function *see* function,  
symmetric
- symmetric gate *see* gate, symmetric
- system  
– bounded-round multiprover interac-  
tive proof 164  
– interactive proof 109–112, 114,  
115, 123, 133, 164, 274, 299, 300, *see*  
completeness, of a protocol, *see*  
interaction, *see* IP, *see* MIP, *see* pro-  
tocol, *see* prover, *see* replacement, of  
provers with an oracle, *see* soundness,  
of a protocol, *see* verifier  
– multi-prover interactive proof *see*  
MIP  
– multiprover interactive proof 110,  
111, 164, 299  
– of elections developed in 1876 by  
Lewis Carroll 272  
– one-prover interactive proof 111,  
299  
– two-prover interactive proof 133,  
135  
– two-prover one-round interactive  
proof 164
- Szegedy, M. 164–166, 269
- Szelepcsényi, R. 278
- Szemerédi, E. 280
- tableau method *see* method, tableau
- TALLY 307, 308
- tally set *see* set, tally
- tally string *see* string, tally

- Tamon, C. 27
- Tan, S. 279
- Tang, C. ix
- tape
  - input 250, 254, 255
  - query 110, 213, 249–251, 254–256
  - work 250, 254, 255
  - write-only 254
- Tardos, G. 29, 64
- Tarui, J. ix, 87–89, 297
- task
  - divided computation 185, 301
- Technique
  - Isolation 56, 64, 67–89, 108
  - Nonsolvable Group 167–195
  - One-Way Function 31–44
  - Polynomial 235–261
  - Polynomial Interpolation 109–166
  - Random Restriction 197–233
  - Self-Reducibility 1–29
  - Tournament Divide and Conquer 45–65
  - Witness Reduction 91–108
- technique
  - divide and conquer, in general 45, 169, 170
  - large gaps and brute force short strings 56
  - nonrelativizable proof 270
  - organization by vii
  - pruning 2
  - relativizable proof 87, 197, 259, 270
- testing
  - multilinearity 164
  - of a witness 40
- textbook
  - use of this book as viii
- Thakur, M. ix, 285
- Theorem
  - Chebyshev's 163
  - Chinese Remainder 120
  - Cook's 58, 59, 64, 268
  - Cook–Karp–Levin 64, *see* Theorem, Cook's
  - Cook–Levin 64, *see* Theorem, Cook's
  - Hartmanis–Immerman–Sewelson *see* Encoding, Hartmanis–Immerman–Sewelson
  - Karp–Lipton 20, 27, 60, 64
  - Mahaney's 2, 26
  - Ogihara–Watanabe 26
  - PCP 165, 166
  - Prime Number 131
  - Rice's 285
  - Savitch's 125, 163
  - Toda's 68, 72, 78, 87, 88, 115, 194, 259
- theorem
  - space hierarchy 22, 27
  - time hierarchy 22, 27, 264
- theorems via algorithms under hypotheses approach *see* approach, theorems via algorithms under hypotheses
- theory
  - circuit 88, *see* circuits, *see* class, circuit
  - complexity 1–308, *see* secret
  - recursive function 271, 295
  - tournament 45, 50, 51, *see* tournament
- Thérien, D. 193, 194, 260, 289
- Thierauf, T. ix, 27, 64, 108, 277, 279, 289, 297
- threshold
  - acceptance 50
  - rejection 50
- threshold circuits *see* approximation, of threshold circuits by parity circuits, *see* circuits, threshold
- threshold gate *see* gate, threshold
- time 138, 251
  - deterministic double-exponential 53
  - deterministic exponential 2, 23, *see* EXP, *see* E
  - deterministic polynomial 49, 264, *see* P
  - double exponential nondeterministic 264
  - nondeterministic exponential 2, 23, 133, *see* NEXP, *see* NE
  - nondeterministic polynomial 49, *see* NP
  - parity exponential 28
  - probabilistic polynomial 49, *see* PP
  - triple-exponential 32
  - unambiguous polynomial 283, 284, *see* UP
- time hierarchy theorem *see* theorem, time hierarchy
- Toda's Theorem *see* Theorem, Toda's
- Toda, S. ix, 64, 68, 78, 87–89, 108, 115, 194, 259, 279, 284, 286, 293, 296–298
- Tomer, J. ix
- Tompas, M. 82, 254, 260, 279, 280

- tongue
- seared vii
- top gate *see* gate, top
- Torán, J. ix, 29, 88, 106, 284, 289, 293, 297–299
- Torenvliet, L. ix, 63, 287, 295, 296
- total degree *see* degree, total, of a polynomial
- total function *see* function, ...total...
- tournament 45–47, 50–52, 57, 62, *see* graph, tournament
  - defeat in 45–50, 62
  - round-robin 45
- tournament graph *see* tournament
- tournament theory *see* theory, tournament
- translation
  - downward, of equality 23, 25, 28
  - upward 23, 28, 29
  - via padding 307
- traversal
  - in-order 178, 180
- tree
  - bushy 8, 56
  - computation, of a deterministic polynomial-space Turing machine 112
  - computation, of a nondeterministic Turing machine 92, 278, 281
  - disjunctive self-reducibility 5, 7
  - expanding of a 6
  - full binary 169, 177
  - full quaternary 177
  - pruning *see* procedure, self-reducibility-based tree-pruning, *see* procedure, tree-pruning
  - root of *see* root, of a tree
  - self-reducibility 5–7
- tree-pruning algorithm *see* algorithm, tree-pruning
- tree-pruning approach *see* approach, self-reducibility-based tree-pruning
- Trevisan, L. ix, 166
- triple-exponential time *see* time, triple-exponential
- True 2–7, 14, 21, 178–180
- truth-table 10, 14, *see*  $\leq_{tt}^P$ , *see* condition, truth-table
- Turing self-reducible set *see* set, Turing self-reducible
- Turing, A. 264
- two-sided error *see* error, two-sided type
- acceptance 108, *see* reduction, of the cardinality of acceptance types
- finite-cardinality acceptance 108
- solution 65
- Ukkonen, E. 26
- UL 67, 68, 82–87, 89, 278, *see* set, UL-complete
- UL/poly 68, 82–84, 278
- Ulfberg, S. 232
- Ullman, J. 264, 266
- Umans, C. ix, 272
- unambiguous function *see* function, unambiguous
- unambiguous nondeterminism *see* UL, *see* UP
- unambiguous NP machine *see* machine, unambiguous NP
- unbounded fan-in circuits *see* circuits, unbounded fan-in
- uniform distribution *see* distribution, uniform
- uniformity 281
  - logspace 281
  - $NC^1$  281
  - P 281
  - $U_{E^*}$  281
- union
  - closure of  $Mod_kP$  under *see* closure, of  $Mod_kP$  under union
  - closure of NP under *see* closure, of NP under union
  - closure of P under *see* closure, of P under union
  - closure of PP under *see* closure, of PP under union
  - disjoint 96, 185, 245
- unique accepting configuration *see* configuration, unique accepting
- universal machine *see* machine, universal
- University of Rochester *see* Rochester, University of
- UP 28, 33–36, 43, 44, 63, 67, 87, 94–102, 105–107, 263, 268, 281–285, 287, 291–293, 296, 298, *see* machine, categorical, *see* set, potential lack of spare  $\leq_T^P$ -hard, for UP, *see* set, UP-hard
  - closure properties *see* closure, of UP...
  - exponential-time analog of *see* analog, exponential-time, of UP, FewP,  $\oplus P$ , ZPP, RP, or BPP

- gap analog of 100
- promise *see* promise, in the definition of UP
- upward translation *see* translation, upward
- US 69, 71, 281, 283–285, 293, *see* set, US-complete
  
- Valiant, L. 88, 279, 282, 283, 286, 287, 302
- value
  - census 20, 49
  - maximum 104
- van der Waerden, B. 163
- van Emde Boas, P. ix, 296
- van Melkebeek, D. ix, 27, 29, 265
- Vandermonde matrix *see* matrix, Vandermonde
- variable
  - random 135, 203
- variance 135, 137, 203
- Vazirani, U. 87–89, 194
- Vazirani, V. 87, 89
- vector
  - nonzero 88
- vectors *see* reduction, of the cardinality of a set of vectors
- Venkateswaran, H. 88, 280
- Vereshchagin, N. 231, 269, 283, 285, 288
- verification
  - deterministic, of mathematical statements 109
  - mechanical, of mathematical statements 267
  - of a certificate 109
  - of accepting computation of a PSPACE machine 125
  - of an arithmetic expression 138
  - of computation of a verifier 165
  - of mathematical statements 109
  - of permanent 112
  - of primality 131, 142
  - of reachability 129
  - of satisfiability 112
  - via interactive proof systems 109
  - with  $\mathcal{O}(\log n)$  random bits 166
  - with polylogarithmic random bits 165
- verifier 109, 110, 115, 123, 132–135, 163, 165, 299, 300, *see* system, interactive proof
- polynomial-time 110
- power of 109
- Vinay, V. 88, 279
- Vishkin, U. 279, 281, 308
- Vollmer, H. ix, 107, 194, 260, 277
- Voltaire, François Marie Arouet 268
- von Braunmühl, B. 27
- von Neumann, J. 264
  
- Waerden, B. van der *see* van der Waerden, B.
- Wagner, K. ix, 26, 27, 106, 107, 194, 260, 271, 272, 274, 275, 290
- wands
  - of combinatorics vii
- Wang, J. 64, 297
- Watanabe, O. ix, 26–29, 43, 63, 64, 89, 108, 276, 277, 297, 308
- weak assignments *see* assignments, weak
- weak equality *see* equality, weak
- weakly P-selective set *see* set, weakly P-selective
- Wechsung, G. ix, 26, 27, 64, 65, 106–108, 260, 274, 275, 293, 303
- weight
  - minimum 68
- WeightSum* 84–86
- West, D. 64
- Whitman, W. 274
- width
  - of PBP 167
- Wigderson, A. 28, 87, 89, 164, 166, 269, 289, 300
- Wilson, C. 28, 308
- winner
  - in an election system 272
  - of a match 45
- wisdom
  - conventional vii
- witness *see* reduction, of numbers of witnesses
  - accepting computation viewed as 94
  - length of 39, 41
  - membership 9, 10, 39–41
  - unique 282
- witness reduction *see* reduction, witness
- witness testing *see* testing, of a witness
- Wössner, H. ix
- word problem *see* problem, word
- work tape *see* tape, work
- world

- relativized 19, 27–29, 65, 107, 108, 228, 231, 232, 268–270, 282, 283, 285, 288, *see* oracle
- worst-case cryptography *see* cryptography, worst-case
- Wrathall, C. 271, 272
- Wright, E. 163
- write-only tape *see* tape, write-only
  
- Yao, A. 88, 232
- Yap, C. ix, 26
- Yesha, Y. 26, 28, 272, 274, 275, 285
- Young, P. 27, 269, 277, 282, 283, 296, 297
  
- Zachos, S. ix, 87, 269, 271, 289, 290, 297, 298, 308
  
- Zaki, M. 297
- Zankó, V. 163
- zero-divisor
  - of a ring 112
- zero-knowledge protocol *see* protocol, one-round perfect-zero-knowledge
- zero-polynomial 152
- Zhong, J. ix
- Zhou, S. 89
- Zimand, M. ix, 29, 43, 231, 269, 297
- Zippel, R. 163
- ZPP 22, 27, 28, 58, 61, 64, 268, 276, 277, 288–290, 294, *see* set, ZPP-hard
  - exponential-time analog of *see* analog, exponential-time, of UP, FewP,  $\oplus P$ , ZPP, RP, or BPP
- Zuckerman, D. 289