Logic, Automata, and Computational Complexity

The Works of Stephen A. Cook

Bruce M. Kapron, Editor

ASSOCIATION FOR COMPUTING MACHINERY

Logic, Automata, and Computational Complexity

ACM Books

ACM Books is a new series of high-quality books for the computer science community, published by ACM. ACM Books publications are widely distributed in both print and digital formats through booksellers and to libraries (and library consortia) and individual ACM members via the ACM Digital Library platform.

Linking the World's Information: Essays on Tim Berners-Lee's Invention of the World Wide Web
Oshani Seneviratne, *Rensselaer Polytechnic Institute*
James Hendler, *Rensselaer Polytechnic Institute*
2023

Effective Theories in Programming Practice
Jayadev Misra, *The University of Texas at Austin, TX, US*
2023

Prophets of Computing: Visions of Society Transformed by Computing
Editor: Dick van Lente, *Erasmus University Rotterdam*
2022

On Monotonicity Testing and the 2-to-2 Games Conjecture
Dor Minzer, *Tel Aviv University*
2022

The Handbook on Socially Interactive Agents: 20 years of Research on Embodied Conversational Agents, Intelligent Virtual Agents, and Social Robotics Volume 2: Interactivity, Platforms, Application
Editors: Birgit Lugrin, *Julius-Maximilians-Universität of Würzburg*
Catherine Pelachaud, *CNRS-ISIR, Sorbonne Université*
David Traum, *University of Southern California*
2022

Spatial Gems, Volume 1
Editors: John Krumm, *Microsoft Research, Microsoft Corporation, Redmond, WA, USA*
Andreas Züfle, *Geography and Geoinformation Science Department, George Mason University, Fairfax, VA, USA*
Cyrus Shahabi, *Computer Science Department, University of Southern California, Los Angeles, CA, USA*
2022

Edsger Wybe Dijkstra: His Life, Work, and Legacy
Editors: Krzysztof R. Apt, *CWI, Amsterdam and University of Warsaw*
Tony Hoare, *University of Cambridge and Microsoft Research Ltd*
2022

Weaving Fire into Form: Aspirations for Tangible and Embodied Interaction
Brygg Ullmer, *Clemson University*
Orit Shaer, *Wellesley College*
Ali Mazalek, *Toronto Metropolitan University*
Caroline Hummels, *Eindhoven University of Technology*
2022

Democratizing Cryptography: The Work of Whitfield Diffie and Martin Hellman
Editor: Rebecca Slayton, *Cornell University*
2022

Applied Affective Computing
Leimin Tian, *Monash University*
Sharon Oviatt, *Monash University*
Michal Muszynski, *Carnegie Mellon University and University of Geneva*
Brent C. Chamberlain, *Utah State University*
Jennifer Healey, *Adobe Research, San Jose*
Akane Sano, *Rice University*
2022

Circuits, Packets, and Protocols: Entrepreneurs and Computer Communications, 1968–1988
James L. Pelkey
Andrew L. Russell, *SUNY Polytechnic Institute, New York*
Loring G. Robbins
2022

Theories of Programming: The Life and Works of Tony Hoare
Editors: Cliff B. Jones, *Newcastle University, UK*
Jayadev Misra, *The University of Texas at Austin, US*
2021

Software: A Technical History
Kim W. Tracy, *Rose-Hulman Institute of Technology, IN, USA*
2021

The Handbook on Socially Interactive Agents: 20 years of Research on Embodied Conversational Agents, Intelligent Virtual Agents, and Social Robotics
Volume 1: Methods, Behavior, Cognition
Editors: Birgit Lugrin, *Julius-Maximilians-Universität of Würzburg*
Catherine Pelachaud, *CNRS-ISIR, Sorbonne Université*
David Traum, *University of Southern California*
2021

Probabilistic and Causal Inference: The Works of Judea Pearl
Editors: Hector Geffner, *ICREA and Universitat Pompeu Fabra*
Rina Dechter, *University of California, Irvine*
Joseph Y. Halpern, *Cornell University*
2022

Event Mining for Explanatory Modeling
Laleh Jalali, *University of California, Irvine (UCI), Hitachi America Ltd.*
Ramesh Jain, *University of California, Irvine (UCI)*
2021

Intelligent Computing for Interactive System Design: Statistics, Digital Signal Processing, and Machine Learning in Practice
Editors: Parisa Eslambolchilar, *Cardiff University, Wales, UK*
Andreas Komninos, *University of Patras, Greece*
Mark Dunlop, *Strathclyde University, Scotland, UK*
2021

Semantic Web for the Working Ontologist: Effective Modeling for Linked Data, RDFS, and OWL, Third Edition
Dean Allemang, *Working Ontologist LLC*
Jim Hendler, *Rensselaer Polytechnic Institute*
Fabien Gandon, *INRIA*
2020

Code Nation: Personal Computing and the Learn to Program Movement in America
Michael J. Halvorson, *Pacific Lutheran University*
2020

Computing and the National Science Foundation, 1950–2016: Building a Foundation for Modern Computing
Peter A. Freeman, *Georgia Institute of Technology*
W. Richards Adrion, *University of Massachusetts Amherst*
William Aspray, *University of Colorado Boulder*
2019

Providing Sound Foundations for Cryptography: On the work of Shafi Goldwasser and Silvio Micali
Oded Goldreich, *Weizmann Institute of Science*
2019

Concurrency: The Works of Leslie Lamport
Dahlia Malkhi, *VMware Research* and *Calibra*
2019

The Essentials of Modern Software Engineering: Free the Practices from the Method Prisons!
Ivar Jacobson, *Ivar Jacobson International*

Harold “Bud” Lawson, *Lawson Konsult AB (deceased)*
Pan-Wei Ng, *DBS Singapore*
Paul E. McMahon, *PEM Systems*
Michael Goedicke, *Universität Duisburg–Essen*
2019

Data Cleaning
Ihab F. Ilyas, *University of Waterloo*
Xu Chu, *Georgia Institute of Technology*
2019

Conversational UX Design: A Practitioner’s Guide to the Natural Conversation Framework
Robert J. Moore, *IBM Research–Almaden*
Raphael Arar, *IBM Research–Almaden*
2019

Heterogeneous Computing: Hardware and Software Perspectives
Mohamed Zahran, *New York University*
2019

Hardness of Approximation Between P and NP
Aviad Rubinstein, *Stanford University*
2019

The Handbook of Multimodal-Multisensor Interfaces, Volume 3: Language Processing, Software, Commercialization, and Emerging Directions
Editors: Sharon Oviatt, *Monash University*
Björn Schuller, *Imperial College London and University of Augsburg*
Philip R. Cohen, *Monash University*
Daniel Sonntag, *German Research Center for Artificial Intelligence (DFKI)*
Gerasimos Potamianos, *University of Thessaly*
Antonio Kruğer, *Saarland University and German Research Center for Artificial Intelligence (DFKI)*
2019

Making Databases Work: The Pragmatic Wisdom of Michael Stonebraker
Editor: Michael L. Brodie, *Massachusetts Institute of Technology*
2018

The Handbook of Multimodal-Multisensor Interfaces, Volume 2: Signal Processing, Architectures, and Detection of Emotion and Cognition
Editors: Sharon Oviatt, *Monash University*
Björn Schuller, *University of Augsburg and Imperial College London*
Philip R. Cohen, *Monash University*

Daniel Sonntag, *German Research Center for Artificial Intelligence (DFKI)*
Gerasimos Potamianos, *University of Thessaly*
Antonio Krüger, *Saarland University and German Research Center for Artificial Intelligence (DFKI)*
2018

Declarative Logic Programming: Theory, Systems, and Applications

Editors: Michael Kifer, *Stony Brook University*
Yanhong Annie Liu, *Stony Brook University*
2018

The Sparse Fourier Transform: Theory and Practice

Haitham Hassanieh, *University of Illinois at Urbana-Champaign*
2018

The Continuing Arms Race: Code-Reuse Attacks and Defenses

Editors: Per Larsen, *Immunant, Inc.*
Ahmad-Reza Sadeghi, *Technische Universität Darmstadt*
2018

Frontiers of Multimedia Research

Editor: Shih-Fu Chang, *Columbia University*
2018

Shared-Memory Parallelism Can Be Simple, Fast, and Scalable

Julian Shun, *University of California, Berkeley*
2017

Computational Prediction of Protein Complexes from Protein Interaction Networks

Sriganesh Srihari, *The University of Queensland Institute for Molecular Bioscience*
Chern Han Yong, *Duke-National University of Singapore Medical School*
Limsoon Wong, *National University of Singapore*
2017

The Handbook of Multimodal-Multisensor Interfaces, Volume 1: Foundations, User Modeling, and Common Modality Combinations

Editors: Sharon Oviatt, *Incaa Designs*
Björn Schuller, *University of Passau and Imperial College London*
Philip R. Cohen, *Voicebox Technologies*
Daniel Sonntag, *German Research Center for Artificial Intelligence (DFKI)*
Gerasimos Potamianos, *University of Thessaly*
Antonio Krüger, *Saarland University and German Research Center for Artificial Intelligence (DFKI)*
2017

Communities of Computing: Computer Science and Society in the ACM
Thomas J. Misa, Editor, *University of Minnesota*
2017

Text Data Management and Analysis: A Practical Introduction to Information Retrieval and Text Mining
ChengXiang Zhai, *University of Illinois at Urbana–Champaign*
Sean Massung, *University of Illinois at Urbana–Champaign*
2016

An Architecture for Fast and General Data Processing on Large Clusters
Matei Zaharia, *Stanford University*
2016

Reactive Internet Programming: State Chart XML in Action
Franck Barbier, *University of Pau, France*
2016

Verified Functional Programming in Agda
Aaron Stump, *The University of Iowa*
2016

The VR Book: Human-Centered Design for Virtual Reality
Jason Jerald, *NextGen Interactions*
2016

Ada's Legacy: Cultures of Computing from the Victorian to the Digital Age
Robin Hammerman, *Stevens Institute of Technology*
Andrew L. Russell, *Stevens Institute of Technology*
2016

Edmund Berkeley and the Social Responsibility of Computer Professionals
Bernadette Longo, *New Jersey Institute of Technology*
2015

Candidate Multilinear Maps
Sanjam Garg, *University of California, Berkeley*
2015

Smarter Than Their Machines: Oral Histories of Pioneers in Interactive Computing
John Cullinane, *Northeastern University; Mossavar-Rahmani Center for Business and Government, John F. Kennedy School of Government, Harvard University*
2015

A Framework for Scientific Discovery through Video Games
Seth Cooper, *University of Washington*
2014

Trust Extension as a Mechanism for Secure Code Execution on Commodity Computers

Bryan Jeffrey Parno, *Microsoft Research*

2014

Embracing Interference in Wireless Systems

Shyamnath Gollakota, *University of Washington*

2014

Logic, Automata, and Computational Complexity

The Works of Stephen A. Cook

Bruce M. Kapron, editor
University of Victoria

ACM Books #43

Logic, Automata, and Computational Complexity: The Works of Stephen A. Cook
Bruce M. Kapron, Editor

books.acm.org
http://books.acm.org

ISBN: 979-8-4007-0779-7 hardcover
ISBN: 979-8-4007-0777-3 paperback
ISBN: 979-8-4007-0778-0 EPUB
ISBN: 979-8-4007-0780-3 eBook

Series ISSN: 2374-6769 print 2374-6777 electronic

DOIs:

10.1145/3588287 Book
10.1145/3588287.3588288 Introduction
10.1145/3588287.3588289 Chapter 1
10.1145/3588287.3588290 Chapter 2
10.1145/3588287.3588291 Chapter 3
10.1145/3588287.3588292 Chapter 4
10.1145/3588287.3588293 Chapter 5
10.1145/3588287.3588294 Chapter 6
10.1145/3588287.3588295 Chapter 7
10.1145/3588287.3588296 Chapter 8
10.1145/3588287.3588297 Chapter 9
10.1145/3588287.3588298 Chapter 10
10.1145/3588287.3588299 Chapter 11
10.1145/3588287.3588300 Chapter 12
10.1145/3588287.3588301 Chapter 13
10.1145/3588287.3588302 Chapter 14
10.1145/3588287.3588303 Chapter 15
10.1145/3588287.3588304 Chapter 16
10.1145/3588287.3588305 Chapter 17
10.1145/3588287.3588306 Chapter 18
10.1145/3588287.3588307 Bibliography
10.1145/3588287.3588308 Bios/Index

A publication in the ACM Books series, #43

This book was typeset in Arnhem Pro 10/14 and Flama using pdfTEX.
The photo of Stephen A. Cook on page xiii is provided courtesy of the BBVA Foundation.

First Edition

10 9 8 7 6 5 4 3 2 1

Stephen A. Cook, O.C., O.Ont., F.R.S.C., F.R.S

Contents

Introduction

Bruce M. Kapron

It would be difficult to overestimate the impact that Steve Cook has had on the field of Theoretical Computer Science. In posing the question of the power of non-deterministic polynomial time computation in 1967 and formulating the theory of NP-completeness in 1971, he created a new focus that dominated research in the theory of computation during the latter half of the 20th century and continues to animate it to this day. Furthermore, since its introduction the theory of NP-completeness has provided an organizing principle for understanding when and how problems are resistant to efficient computational solutions, with an impact that extends well beyond computer science. While this is the work that has made Steve Cook famous, it is part of a larger program of research that has had a significant impact in a diverse range of fields from the foundations of mathematics to the architecture of parallel computers.

Cook was awarded the ACM Turing Award in 1982. The citation for the award recognized his achievement as follows: "For his advancement of our understanding of the complexity of computation in a significant and profound way. His seminal paper, 'The Complexity of Theorem Proving Procedures,' presented at the 1971 ACM SIGACT Symposium on the Theory of Computing, laid the foundations for the theory of NP-Completeness. The ensuing exploration of the boundaries and nature of NP-complete class of problems has been one of the most active and important research activities in computer science for the last decade."

This volume presents a selection of works by Cook, focusing on some of his most significant contributions without attempting to be comprehensive. Notwithstanding its title, there are many works and many areas that are beyond the scope of this volume, which primarily considers contributions made before his 1982 Turing Award. Thematically, the included works are centered around the complexity of computation and its connection to logical systems and models of computation.

Even given this restriction, there are significant areas, such as the complexity of multiplication, that have not been included; some of these are described below. The current volume contains a selection of papers and chapters focusing on Cook's contributions to P, NP, and the theory of NP-completeness, the complexity of propositional proof systems, logical systems for bounded arithmetic, space-bounded computation, and models for efficient parallel computation. His work in each of these areas is fundamental, and in a number of cases, such as the theory of NP-completeness, proof complexity, and bounded arithmetic, Cook is among the handful of researchers who could be considered founders of these fields. In other areas, such as space complexity, he has had a sustained impact through his results and techniques.

Cook's work on complexity, automata, and logic not only made important contributions to each of these individual subareas but it also brought them together in a way that provided a cornerstone in the foundation of computer science as it is understood today. The papers included in this volume and the chapters addressing their nature and significance not only survey a collection of contributions but they also give a picture of how Cook was able to create a unified theory of complexity.

The chapter by Christos Papadimitriou focuses on Cook's most famous and significant contribution, namely the 1971 STOC paper, "The complexity of theorem proving procedures" that introduced NP-completeness. While much has been written on the history and significance of NP-completeness, the current chapter provides a historically focused look at Cook's paper, in particular locating it in the computer science research milieu of the late 1960s and early 1970s, including its influences and immediate impact. The chapter also points out the important link between Cook's contemporary work on automata theory and his breakthrough in the theory of NP-completeness, with a particular focus on another 1971 contribution, "Characterizations of pushdown machines in terms of time-bounded computers."

Chapters by Sam Buss and Jan Krajíček address Cook's work in the areas of bounded arithmetic and proof complexity. At the time of Cook's papers, "Feasibly constructive proofs and the propositional calculus" and "On the lengths of proofs in the propositional calculus," co-authored with Robert A. Reckhow, the idea of bringing complexity measures into logical systems had been barely explored. The research areas for which these papers were foundational are flourishing and play an important role in complexity theory and other areas of theoretical computer science. As the chapters make clear, these areas are complementary while also providing another approach to understanding basic questions about computational complexity. They also provide a bridge between theoretical computer science

and traditional areas of interest in mathematical logic, including proof theory and model theory.

In their chapter, Paul Beame and Pierre McKenzie present a comprehensive account of the development of the theory of parallel computation while documenting the crucial contributions made by Cook and his collaborators, including chapter authors Beame, McKenzie, and Nicholas Pippenger, which helped advance the field. Cook's paper "Towards a complexity theory of synchronous parallel computation" gives a comprehensive overview of his view of the field as it stood at the start of the 1980s, and itself serves as a good introduction to the field.

The theory of space-bounded computation and the power of polynomial time versus logarithmic space is considered in the chapter by Nicholas Pippenger. This is another area whose direction was impacted by Cook's contributions to the definition of related complexity classes, development of techniques, and exploration of computational models. Two of Cook's related papers, spanning his work in this area, are included in the volume. The first, "A time–space tradeoff for sorting on a general sequential model of computation," co-authored with Alan Borodin, is an early contribution to the theory of branching programs. Many of Cook's ideas in this area come together in the second paper "Pebbles and branching programs for tree evaluation," co-authored by Cook with Pierre McKenzie, Dustin Wehr, Mark Braverman, and Rahul Santhanam. This paper is unique in the volume in that it was written much later than the others. The motivation behind its inclusion was to demonstrate the long-term nature of Cook's work on basic problems, and how it may take many years to synthesize previous work in obtaining new results.

The remaining contribution included in the volume is the oldest, and one that has not been previously published. This is "A survey of classes of primitive recursive functions," which summarizes material presented in a course Cook taught at UC Berkeley in 1967. This is an important document that demonstrates the influence of logic and automata theory on the development of complexity theory, and in particular provide a foundation for his future work on P and NP.

As previously noted, the extent and significance of Cook's work goes well beyond the contributions surveyed in this volume. The following selection of his works is still not comprehensive but will give some additional indication of the scope, variety, and impact of his research.

Cook's Ph.D. thesis, titled "On the Minimum Computation Time of Functions" [Coo66a] addresses the computational complexity of multiplication. This includes a presentation of Andrei Toom's recursive multiplication algorithm [Too63], in a form now known as Toom–Cook multiplication. This algorithm is asymptotically subquadratic and is used in practice for the multiplication of large integers as well

as for polynomial multiplication over finite fields. In this latter form it has recently seen application in lattice-based post-quantum cryptography, in particular in the implementation of a NTRU-based key-encapsulation mechanism [NG21].

In the area of program verification, in particular Hoare logics, Cook has made two notable contributions. The first is his work with Derek Oppen [CO75, OC75], which is an early contribution to the problem of verifying programs that manipulate data structures. A more significant work is Cook [Coo75a, Coo78b], which introduces the notion of *relative completeness* for Hoare logic. Hoare logic [Hoa69] is a formal system for proving *partial correctness* assertions about programs, over some programming formalism and formal language of assertions. In the general case of while-programs and assertions in the language of Peano arithmetic, Hoare logic is not complete—there are valid assertions that are not provable. Cook's result essentially shows that this is due to the incompleteness of Peano arithmetic itself. If an oracle for the validity of arithmetic assertion statements were available, then Hoare logic is complete. This notion of relative completeness has become a central notion in the metatheory of Hoare logics and systems for program verification.

Works on subrecursive characterizations of complexity classes, including characterizations by Robert W. Ritchie of linear space via bounded primitive recursion [Rit63] and by Alan Cobham of polynomial time via bounded recursion on notation [Cob65], were an early influence on Cook's approach to computational complexity. With Stephen Bellantoni, he made his own fundamental contribution to this approach [BC92a, BC92b], giving a characterization of polynomial time via a form of *predicative* recursion on notation. In contrast to earlier characterizations, the Bellantoni–Cook scheme does not rely on bounding the size of the result of recursions via functions from an already-defined class, instead using a syntactic restriction to ensure that a value defined by recursion cannot control a nested subrecursion. Along with a related characterization by Daniel Leivant [Lei93], this work was the starting point for the field of *implicit computational complexity*, which has provided a deeper understanding of the connections between computational complexity and subrecursion, as well as having applications to programming languages, proof theory, and the foundations of mathematics.

Starting with the work of Alan Turing [Tur36], there has been a direction in the theory of computability that considers computation over nonfinitary domains, such as real numbers and function spaces, and even hierarchies of higher-order functions. Work on computational complexity over such domains is much less developed. With respect to higher-order functions, Sam Buss [Bus86b] proposed a notion of polynomial time higher-order functions, primarily as a technical device for interpreting systems of intuitionistic bounded arithmetic. This was followed by Cook's work with Alasdair Urquhart [CU89, CU93] on the finite-type system

PV^{ω}, again with the goal of providing functional interpretations of constructive systems of bounded arithmetic. The problem of relating these higher-order formalisms to computational models was taken up by Cook and Bruce M. Kapron in Cook and Kapron [CK89, CK90] and Kapron and Cook [KC91, KC96]. The latter of these works provided a characterization of Kurt Mehlhorn's class [Meh74, Meh76] of type-two polynomial time functions via a natural generalization of polynomial-time oracle Turing machines to function oracles. In collaboration with Paul Beame, Jeff Edmonds, Russell Impagliazzo, and Toniann Pitassi [BCE+95, BCE+98] and with Impagliazzo and Tomoyuki Yamakami [CIY97], he investigated the relationship between type-two polynomial time, generic oracles, and NP search problems using a complexity model originally proposed by Mike Townsend [Tow90]. Work on polynomial time for real-number computation was initiated by Ker-I Ko and Harvey Friedman [KF82] and has grown into a substantial area of study (see, e.g., [Ko91].) Using ideas from Kapron and Cook [KC96], Akitoshi Kawamura and Cook were able to extend models for real-number computation to model the complexity of real operators [KC10, KC12].

Steve Cook has been recognized as an innovator in computer science and mathematics and a leader in his field, not only with the Turing Award but also the CRM–Fields–PIMS Prize (1999), awarded by Canada's three mathematics institutes for research achievements in the mathematical sciences, the Association for Symbolic Logic's Gödel Lecture (1999), the Royal Society of Canada's John L. Synge Award (2006) for outstanding research in the mathematical sciences, the Czech Academy of Sciences Bernard Bolzano Medal for Merit in the Mathematical Sciences (2008), the Gerhard Herzberg Canada Gold Medal for Science and Engineering (2013), and also as a Fellow of the Royal Society of Canada, the Royal Society of London, and Association for Computing Machinery, and as a Member of the National Academy of Sciences (US), the American Academy of Arts and Sciences, and the Göttingen Academy of Sciences. Michelle Waitzman's biography "Stephen Cook: Complexity's humble hero," gives us a picture of the man behind these achievements, one who is respected and valued by those who know him not only as a researcher but also as a collaborator, mentor, teacher, and friend.

Note on formatting and typesetting As noted above, this volume is not intended to be a compendium or critical anthology of Cook's works. Rather, the selected articles and supporting chapters are meant to provide an introduction to some of his fundamental contributions, especially those for which he received the Turing Award. The papers reproduced here (some of which were originally prepared on a typewriter) are typeset in a consistent style reflecting modern conventions. In particular, the citation style is consistent with that used in the rest of the volume,

and citations in those articles are linked to the volume bibliography, although for completeness the papers' reference sections have been retained.

Acknowledgments

To begin, I would like to thank my wife, Valerie King, for her continuing support and enthusiasm for this project. The idea of the volume was initially proposed to me by Tamer Öszu, in his role as the founding editor-in-chief of ACM Books. I thank Tamer for his vision and assistance in getting things off the ground and also for compiling the bibliography of Steve Cook's works that appears in this volume. Tamer was capably succeeded in his job as the academic editor for the volume by Sanjiva Prasad, whom I thank for his ongoing guidance and attention to detail which were essential in getting the volume to its completion. I am very grateful to all of the contributors and thank them for their diligence and patience, and I thank the staff at ACM Books, including Scott Delman, Sean Pidgeon, Bernadette Shade, Barbara Ryan, and Julia Stevenson, for their help in keeping what has become a multi-year project organized and on track. I also thank Karen Grace and the team at Compuscript for their care and skill in the final production of the volume. Finally, I want to express my deep gratitude to Steve Cook for his academic supervision and mentoring and the intellectual curiosity and enthusiasm for research that he imparted to me.

PART I

BIOGRAPHICAL BACKGROUND

Stephen Cook: Complexity's Humble Hero

Michelle Waitzman

Anyone who has known Stephen Cook, more commonly known as Steve, during his long and influential career has similar things to say about him. The words "smart," "modest," and "kind" are used in equal measure. Although he is most often associated with his groundbreaking work on NP-completeness—which has become so fundamental to the study of math and computer science that practically all students encounter it in their undergraduate textbooks—he is far from a one-dimensional figure. He derives as much satisfaction from racing sailboats as he does from examining computability problems. He enjoys music as well as logic. He has been a dedicated father to two boys, now men, who in very different ways have followed in his footsteps.

Steve has accomplished a great deal, and yet he seems to be utterly without ego. He is equally happy to give his time to undergraduate students with an interest in complexity theory and fellow Turing Award winners; to him, every person and every idea deserves a fair hearing. Perhaps this egalitarian approach to life, and his lack of focus on his own achievements, explains why so few people outside Steve's areas of research are familiar with the true breadth of his accomplishments.

Steve's small, cramped office at the University of Toronto paints a portrait of his long career. The crowded bookshelves are bowed under the weight of the dissertations his graduate students have produced and the proceedings of conferences where he has presented his work. A few of his many awards lie on top of piles of papers, not deemed important enough to hang on the wall. The furnishings have not changed much since he first occupied the space in the early 1980s, although the computer monitor has grown larger to accommodate aging eyes. From across

the hall, long-time colleagues with similar, cramped offices poke their heads in to see about plans for lunch.

Steve's retirement from lecturing and the supervision of graduate students in 2018 provided an opportunity for his colleagues and friends to put him in the spotlight for a moment—an awkward place for Steve. A symposium in his honor at the Fields Institute for Research in Mathematical Sciences in Toronto in 2019 drew speakers from around the world, and so many attendees that even the overflow room needed an overflow room. In Steve's own words, he was "flabbergasted" by the lineup of speakers and the scope of the event. However, attendees and speakers alike were less surprised since it was clear that an event of that scale was needed in order to celebrate a career of such significance.

Steve Cook is a role model, not only as a dedicated researcher but also as an example of mentorship, compassion, and humility.

1.1 Growing Up: Buffalo and Cows

Steve's parents met at the University of Michigan, where his father earned a doctorate in chemistry and his mother earned a master's degree in history. She would later add a master's degree in English. From the start, his was an academically inclined family.

"It was assumed that we were all going to university, and they were keeping track and making sure we were doing OK. I was a little slow—I think my mother was a bit worried when I was in kindergarten, learning to read," Steve recalls.

Steve was born in December 1939, the second of four boys in the family. Their early days were spent in Buffalo, New York, near the Canadian border. His father worked for Linde and Union Carbide and also taught at the University of Buffalo. He would sometimes go to Kleinhans Music Hall to indulge in his love of classical music. Steve's mother, in addition to raising four sons, was heavily involved in the League of Women Voters in the area and taught English at a local community college.

Although the family enjoyed everything that Buffalo had to offer (Steve's favorite childhood event was the circus that came through every year), they decided to move out of town and live on a farm. Steve's mother had grown up on a farm in Michigan, and his father liked the idea of a rural lifestyle. When Steve was 10 years old, the family moved to Clarence, New York, about 20 miles from Buffalo. They purchased a farm and leased most of it to a local farmer who grew crops and raised heifers. Steve says, "We had 68 acres, and at the back there was about 8 acres of woods. So it was really quite nice. In fact, I spent time making a trail in the woods that led to a special tree, a blossom tree."

The family kept one Guernsey cow on their homestead, named Millie, and Steve was in charge of the afternoon milking. His brother Mike, the second youngest, was more of a morning person and took the 5 am milking shift. Steve recalls, "We bought a pasteurizer. She gave a couple of gallons of milk a day; it was way too much for us to use so we sold it to the neighbors."

The farm was a fun place for a bunch of young boys, and they could often be found playing in the barn. They would make tunnels through the bales of hay, and they even set up a little basketball court in the middle of the barn. As they grew a bit older, their interests evolved. At 14 years old, Steve was excited to have the opportunity to drive the farmer's tractor down the road a couple of miles.

Clarence was also home to someone who would have a profound influence on Steve—Wilson Greatbach, an electrical engineer. Steve told ACM during an interview, "Transistors were a very new thing then, and he designed a transistor circuit that went 'bip, bip, bip,' and eventually turned it into an artificial pacemaker for hearts, which was implantable. That had never been done before. Well, of course you couldn't do it with vacuum tubes, obviously. So he eventually got ushered into the Inventors Hall of Fame in the United States for inventing this thing" [ACM16].

Steve went to Greatbach's home workshop and would solder circuits together based on instructions and drawings from Greatbach. This got Steve interested in electronics, and he started to think about pursuing a career in that area.

"Later, I got a summer job at the company he was working with," Steve says, "so I was actually attached to designing circuits to some extent. The one I designed was to make a computer that divided large numbers. This was all in the 50s. It all sounds trivial now, and I don't think it was ever used, but Wilson Greatbach was impressed, anyway, with what I was doing."

At school, Steve's talent for math was already becoming clear. He did well in his math and science courses generally, but New York state has a set of standardized exams called the Regents Exams. Steve recalls that his score on the math exam was 100 percent. Despite his undeniable strength in that subject, Steve decided that electronics was more interesting, and he enrolled in engineering for his undergraduate studies. He never considered going anywhere other than his "family university," the University of Michigan. Not only had his parents met there, Steve's older brother had already followed in their footsteps and enrolled at the university when he finished high school.

1.2 The Lure of Mathematics

Steve's path to a career in engineering started off according to plan. He moved to Michigan in 1957 and began working toward his engineering degree. But during his freshman year, he also had his first real exposure to computers. The university had

an IBM 650, the first commercially mass-produced computer. Coincidentally, it was created and manufactured in upstate New York, not too far from where Steve had grown up. The 650 was marketed as a "magnetic drum data processing machine," and it was known as the workhorse of early computing.

Steve took a computing course taught by Bernard Galler for one hour per week during his second semester. He learned to program the machine using the university's own programming language—Michigan Algorithmic Decoder, or MAD. Even at that early stage, he was using his access to computing power to test out mathematical proofs. "I wrote a program to test Goldbach's conjecture, which was that every even integer greater than two is the sum of two primes. So I tested it up to some large number and it turned out to be verified," Steve says. This idea that computers could be used to test proofs would stay with Steve and influence his graduate work.

In addition to his early computing experience, Steve immediately started to develop his talent for mathematics. His first calculus professor, Nicholas Kazarinoff, took notice of Steve's impressive level of understanding and started giving him extra problems to work on that were more advanced than the usual course work. Professor Kazarinoff also encouraged him to take more challenging math courses and work at an accelerated pace. In the second semester of his first year, Steve was enrolled in a third-year algebra course. While it may have seemed obvious to Professor Kazarinoff where Steve's real talents lay, Steve was still thinking of a career in engineering, not math. In an interview with the Babbage Institute he said, "I was good in mathematics in high school, but I didn't know any mathematicians. I didn't really know what mathematicians did" [CBI02]. This unclear path to a career in mathematics steered Steve toward engineering, but fortunately the emerging field of computer science would provide a way for his interests to be combined into a rewarding career.

During his summers as an undergraduate student, Steve found jobs that used his computer and engineering-related experience. One summer, he worked for Cornell Aeronautical Laboratory (now called Calspan), which was near Buffalo. Steve used a Bendix G-15 computer there, he says, "which had 400 vacuum tubes and was about the size of a refrigerator. I did a little bit of programming, and every once in a while a tube would burn out and the computer would stop, so I learned how to take an oscilloscope and find out where the bad tube was and replace it." The laboratory was working at the time on a computer guidance system to help fighter jets land on aircraft carriers. It was a project with little room for error. During the testing, Steve felt that the pilots were very hesitant to trust a computer over their own training and instincts. Computer navigation was so new at the time that it was hard to blame them for their reluctance.

That job wasn't the only one where Steve worked on military projects. He also held a summer job with Autonetics, a company that did avionics work for the US military. Steve worked on a program to test their guidance system for intercontinental ballistic missiles. For that, he required confidential government clearance.

For a while, a career in military engineering was emerging as a possibility for Steve. But despite his interest and skill on the engineering side of computing, Steve's fascination with math continued to nudge him in a more academic direction. After two-and-a-half years as an engineering student, he switched to a mathematics major and ended up graduating with a math degree.

Steve's next challenge was making a decision about graduate studies. It was a given that Steve would pursue a graduate degree in math. He says, "My parents had graduate degrees, so there was no question about that. I think I was already thinking about an academic position." The only decision to be made was where to apply. Staying at the University of Michigan was an option, but Steve was attracted to some of the better-known math departments at top universities like Berkeley, MIT, and Harvard. In 1961, Steve moved to Massachusetts and began working toward his master's degree at Harvard.

Harvard introduced Steve to new influences and new possibilities—not all of them academic. It was at Harvard that Steve first tried his hand at sailing, which would become one of his great loves. But sailing on the Charles River wasn't quite what Steve was looking for, so after a couple of outings with the university's sailing club he put his nautical activities on hold and concentrated on his studies.

Harvard was not just an Ivy League school with a strong mathematics department, it was also one of the first American universities to be involved in computer science, beginning as early as the 1940s. Harvard launched a master's program in computer science in 1950, which was the first in the country. In 1962, just as Steve was completing his master's degree, the Harvard Computation Center was launched. So although he'd chosen to study mathematics, Steve found himself in exactly the right place at the right time to be at the forefront of the growth of computer science in American academia.

Steve's master's degree in math consisted of coursework, and he took some courses from an applied physics professor, Hao Wang. "I took a course from him because it involved computation," Steve says, "and that's how I became hooked." Professor Wang was a logician and had a strong interest in computers. He'd worked for the IBM Watson Research Laboratory before coming to Harvard, and he had an interest in using computers to prove theorems. Steve found that idea intriguing too. "It was really clear that logic and computation were very interesting and intriguing subjects," Steve says. It was these shared interests with his professor

that drew Steve toward computer science as a possible field of research. In 1962, Hao Wang became Steve's advisor for his doctorate.

Another Harvard alumnus had a hand in steering Steve toward his eventual area of interest. Alan Cobham had been a doctoral student at Harvard before Steve arrived, although he'd left to work for IBM after completing his thesis without actually receiving his Ph.D. In a paper, Cobham posed the question: in what way is multiplication harder than addition? The question had a profound influence on Steve. In the early 1960s, computational complexity was just beginning to emerge as an area of study. "I got in there early!" Steve says. "In my thesis I proved theorems, so in that sense it was mathematics even though nobody in the math department did that." In fact, Steve's thesis focused on the computational complexity of multiplication, so it was directly inspired by Cobham's work.

Steve was drawn to complexity theory because it was just beginning to evolve. Compared with other areas of mathematics, which had been studied for centuries, it was a new world with much to be discovered. Steve told the authors of *Out of Their Minds*, "Before real computers existed, you couldn't execute algorithms except by hand. The process was so tedious that the question of complexity was less interesting. Now that we had these powerful machines to help us and they seemed like an enormously powerful tool—thousands of operations per second—it was very natural to ask, just what sorts of problems could you really solve? ... Obviously there are problems that are solvable in principle by algorithms but not in practice, because the sun burns out before you solve them. So, it's just a very natural question to ask about the inherent difficulty of problems" [SL95].

In addition to completing his thesis, Steve had to jump over other hurdles to earn his Ph.D. at Harvard. "The math department had these exams that the students had to take. In order to continue towards a Ph.D. after your second year, you had to pass this exam—and a fair number of people failed," Steve recalls. "I was nervous because I knew people who had failed. So I remember about a week before, I just intensely studied the material for 12 hours a day and then took a sleeping pill at night. I was waiting for the answer to appear in my mailbox. I thought I did OK, but I wasn't sure. I did pass—that was a relief." Was there any real chance that Steve, who would become an internationally respected mathematician, was at risk of failing the exam? It seems unlikely, but his reluctance to assume that he would easily pass is typical of Steve's humble nature.

Steve received his Ph.D. in 1966 and took a short break over the summer after graduation. He took the opportunity to have a European adventure and not do any work. He was to meet his youngest brother, Phil, in London to travel together. But Phil did not appear as planned. Instead, he sent the tent that the two had been planning to share, and Steve ended up traveling through England on his own for

the summer. He made his way through the country mainly by hitchhiking and staying at bed-and-breakfasts. It served as a relaxing interval between life as a graduate student and the beginning of Steve's academic career.

1.3 From Smooth Sailing to Rough Waters

The University of California, Berkeley was a lively place in the late 1960s, to say the least. It had become the epicenter of the student-led free speech movement, with its famous sit-ins, protests, and occasional confrontations with law enforcement. It was not unheard of for campus events at Berkeley to end with tear gas being thrown into the crowd. This was the environment in which a quiet, understated young assistant professor from upstate New York, Steve Cook, joined the math department and began his prestigious academic career. The university was just launching a new computer science department, but Steve was not a part of that. If he had been, his entire career might have been happily spent in sunny California rather than snowy Canada. But things don't always go to plan, and Steve would make the necessary adjustments to keep the wind in his sails.

Admittedly, academia was not the only thing on Steve's mind when he decided to join the faculty at Berkeley. "It had a reputation as a good place to meet a wife," Steve says. "Also, I knew San Francisco Bay was there, and although I had not sailed much, I liked the idea of sailing. I joined the sailing club in January of my first year—and this is Berkeley so it was actually warm in January, and not much wind, so it was a good time to learn to sail. It was a student sailing club, but they let faculty join. And they gave sailing lessons, so I learned to sail."

As it happened, Steve's two extracurricular motivations for moving to Berkeley came together rather serendipitously. Steve met a young woman named Linda, who was the secretary of the University of California Yacht Club. He'd spotted her at a club meeting and thought she was attractive, but he didn't get a chance to talk to her there. They met later, at a sailing club party on a boat, and spoke for the first time. She was an undergraduate student in philosophy and Spanish at Berkeley. Steve was immediately smitten and soon the two started dating. Steve had such a youthful appearance that at first she didn't believe he was not a student. The 28-year-old looked so young that he had trouble buying beer. He eventually grew a beard to help him look his age, and kept it for many years.

Steve's interest in sailing strengthened as he participated in club races in San Francisco Bay, and he began to recruit his own students from the university to crew for his boat. It was a practice he would continue after he moved to Toronto.

Steve also had the opportunity to go on some longer trips, including sailing to Hawaii over Christmas break one year. "One of the grad students at the club lived in Hawaii and he was an expert sailor. He got a contract to deliver a brand new

41-foot sailboat from Los Angeles to Hawaii and he needed a crew. He tried various people out, and we went out and it happened to be a windy day and I actually got seasick. But he decided to take me, and at first he chose Linda, too, but the owner said 'no women and no booze.' So Linda didn't get to go, but we did have some booze." They sailed for 17 days in open ocean, the longest trip Steve has ever undertaken. It gave him some good training in navigation. "This was before all of the electronic navigation—you used a sextant and the sun. And when we knew we were approaching the islands of Hawaii, it was pitch dark and the wind was behind us and we were tearing along. So that was a bit nerve wracking; we hadn't seen anything or anybody, we were just trusting that our sextant didn't lie to us. And then the dawn came and it was foggy, but when the fog cleared, five miles off to the left was the island." They arrived just in time for New Year's Eve, and one of the crew used a ham radio to let everyone back in California, including an anxious Linda, know that they had arrived safely.

Steve and Linda turned out to be a good match, despite their very different areas of study and personalities. Drawn together by their shared interest in sailing, they ended up marrying two years after they met and have been together since. The couple have two sons, Gordon, born in 1978, and James, born in 1985. James says, "Their personalities really complement each other." He remembers his mother playing practical jokes on Steve, like pretending that their car had been stolen. "It's the kind of thing my dad would never do, play some joke like that on my mom. He's serious and doesn't joke that much, and my mom is a little more fun." The two are a good balance for one another. The academic pressure that marked Steve's childhood was not a big part of their sons' upbringing. Steve provided the boys with challenges in both academic and athletic areas, and they rose to meet those challenges in their own ways. But parenthood was years away and the last thing on Steve's mind as he settled into his new life as an assistant professor in 1966.

Steve's position at Berkeley was unusual. He spent half his time as part of the math department, and half with the "computing center." It was an on-campus computing facility, but it was separate from the university's new computer science department. The computing center was part of the College of Letters and Science, while the computer science department was part of the College of Engineering. It was an odd situation brought about by a piecemeal approach to accommodating the new and emerging field of computers into existing faculties. This unusual state of affairs would end up being problematic for Steve.

Steve was responsible for supervising graduate students almost immediately. His first student turned out to be an exceptional one: Walter Savitch. Walter's work for his dissertation at Berkeley led to his discovery of "Savitch's theorem," which would eventually be included in most textbooks on complexity theory. It was at

Berkeley that Steve's open and respectful style of supervision began to take shape. At the time, he was not much older than the students he was supervising (and probably looked younger than many of them), so perhaps it seemed natural to him to treat them like peers rather than students. This approach continued throughout his career, making him a sought-after supervisor for students with an interest in complexity theory and logic.

Steve was not a prolific publisher during his time at Berkeley. He has been known throughout his career for publishing fewer papers than most researchers, but the papers that he did publish ended up helping to define and influence the field of complexity theory from that point forward. Although his ideas were not yet fully formed at Berkeley, Steve's research there laid the groundwork for his impressive career.

In 1967, just a year after receiving his PhD, Steve prepared some course notes at Berkeley that he later distributed to some colleagues. Bruce M. Kapron, who was a doctoral student of Steve's in the 1980s, was asked to interview Steve on behalf of the ACM in 2016. When they met for that interview, Steve showed him "some notes from a course that he taught at UC Berkeley in the January term of 1967. I was quite surprised to see that in these notes there was a fully worked out formulation of the classes that we now call P and NP. He also directly posed the question of whether P is equal to NP, with the comment that it would probably be a difficult problem to solve. People typically think of Steve's contribution to the P versus NP problem as starting with his 1971 NP-completeness paper, but from these notes it is clear that he had formulated the problem long before that. My feeling is that this was the first complete statement of the problem as we understand it today." When Bruce realized the significance of the notes, he transcribed them for digital upload and made them publicly available for the first time. Otherwise, this historical information may have been completely lost, and the origins of the P versus NP problem left to speculation.

Steve's early papers caught the attention of faculty members in the new computer science department at Berkeley, including future Turing Award winners Dick Karp and William "Velvel" Kahan. Steve also began to present his research at conferences, which introduced more people in the theoretical computer science field to his ideas.

Steve also attended some of the seminars and lectures by professors in the computer science department because he was interested in the work they were doing. Dick Karp remembers Steve attending his seminars. "I remember a couple of times when I gave a talk and he asked a sharp, clarifying question—very helpful. He was always very gracious, gentlemanly, low-key—but obviously very smart."

Unfortunately, the budding field of complexity theory was not well understood by Steve's peers in the math faculty and, more importantly, by his employers. According to Dick Karp, "The problem was that computational complexity looked pretty foreign to the people in the math department, so he didn't have advocates—or enough advocates." After four years at Berkeley, he was considered by the tenure committee, and they decided not to offer Steve a tenured position. Steve told the Babbage Institute, "My natural colleagues tended to be in computer science departments and I think that made a big difference. My field may have been a little too new to be accepted in mathematics" [CBI02].

Velvel Kahan, who had joined the computer science faculty at Berkeley in 1969 after leaving the University of Toronto, recalled the situation during an interview at the Heidelberg Laureate Forum. "My office was in the same building as the math department, and I would pass Steve's office and I would hear how he was treating students. I came to the conclusion that I should visit the math chairman at that time, Addison, and told him, 'You know, you have a gem here, but he doesn't blow his own horn. But I've heard what he says and how he says it, and we really should keep this guy.' And Addison said, 'Well, it's too bad you weren't here last September, because that's when we decided not to give him tenure.' Steve was working in what you could call discrete math, on some rather important problems. But other members of the math department didn't appreciate the importance of those problems. But more than that, they didn't appreciate the guy's talent—particularly his talent for dealing with students" [HLF18].

Behind the scenes, several members of the computer science faculty were lobbying their dean to offer a position in their department to Steve. Dick Karp, Mike Harrison, Elwyn Berlekamp, and Velvel Kahan all tried to explain what a valuable addition Steve would be to the department. Velvel Kahan says, "We tried to go to the dean and say we need a position for this guy or we're going to lose him. And the dean said the budget committee is not going to grant this guy a tenured position if so prestigious a department as mathematics has denied him tenure. We could protest until we were blue in the face, but it didn't do any good" [HLF18]. In an article on the Berkeley website recounting the history of the computer science department on the occasion of its 30th anniversary, Dick Karp said about Steve, "It is to our everlasting shame that we were unable to persuade the math department to give him tenure" [Kar19].

There was a chance that the computer science department would reconsider and offer him a position if Steve stayed on in his untenured position for another year. However, Steve thought that it was too risky to spend another year in an uncertain position and instead decided that it would be best to look for a job elsewhere and secure his future.

Leaving Berkeley was a great disappointment to Steve. Even during discussions about it in interviews years later, it is clear that the rejection was hurtful at that early stage of his career, and it left him a bit cynical about the bureaucratic side of academia. He had also settled into the California lifestyle and was enjoying sailing with his wife, who'd lived in Berkeley since she was a toddler. To Linda, the idea of leaving California was almost unthinkable. But, Steve says, "to be fair, when I proposed to her, I told her I might not get tenure. So she was forewarned." Steve took it all in stride. Dick Karp says, "I remember being very impressed by the fact that he didn't cry out about the injustice of it all or make a fuss. He just found a place where he wanted to be."

What Steve didn't know at the time was that his computer science colleagues were also busy trying to find him a good place to be. In fact, it was a phone call from Velvel Kahan that steered the direction of Steve's career. Velvel says, "I phoned the chairman of the computer science department at the University of Toronto, which I had just left. He was an old friend, Tom Hull. And I said, 'Tom, we're about to make a terrible mistake from which you can profit.'" He encouraged Tom to recruit Steve for the University of Toronto's growing computer science department, which already had a strong reputation. The department was founded by three senior members, physicist Kelly Gottlieb, who'd been involved in computer science since the end of the Second World War, Tom Hull, and Pat Hume. They would be the first three chairs of the department as well. In addition, the department included graph theorist Derek Corneil and complexity theorist Allan Borodin, both of whom would remain at U of T throughout Steve's career, and the three of them would be cornerstones of the theory group's enduring reputation as one of the best in the world.

Toronto was not somewhere that Steve had thought of moving, though. He was busy interviewing at other universities, including Princeton, Yale, and the University of Washington in Seattle, when Tom Hull called his home in California. Tom had to persuade Linda to tell him where he could contact Steve. His persistence paid off.

Allan Borodin recalls Tom's initial inquiries in 1970, following up on Velvel Kahan's phone call. "Tom was out aggressively recruiting people. When I was here, in that first year, he asked me if I knew Steve and I said I knew his work. I studied his PhD thesis in the first summer of my graduate work, and that got me very interested in complexity theory and fundamental questions about computations. I was very influenced by Steve's early work."

Steve says, "Tom was a very good recruiter. He was very polished and showed me around Toronto and pointed out that there is a lake there [for sailing] and that part was a feature! The department was up and coming."

After his initial interview, Steve was invited back for a second visit along with Linda. Allan says, "at that time it wasn't heard of that you brought back a candidate and his spouse to see the city, to convince them to come. It was rather unique, I think. So he had Steve and Linda back, and found out what their interests were and showed them everything. They showed him what sailing could be like here. You're really selling to a couple, you're not just selling to one person. You're selling the department, the university, the city, the country—you're selling everything."

Convincing Steve to come to Toronto involved more than selling the city or university. Allan Borodin believes that Tom Hull was a successful recruiter because he made people feel like they were valued. "People often are very impressed when you genuinely feel people really want you. I'm sure he was getting other offers, but I think this offer probably came across as being very genuine." Steve did get other offers, including one from Yale, but he decided to take a chance and move to Toronto.

1.4 Growing Roots, Making Waves

Steve's initial appointment at the University of Toronto was split between the math and computer science departments, similar to his position at Berkeley, and he taught courses on both subjects. But after a year, Steve made the switch to working full time in the computer science department. This move eliminated the possibility of repeating the disappointment he'd gone through at Berkeley, and he was offered tenure very early on. Finally, Steve had the security he'd been hoping for. The computer science department had just expanded to include an undergraduate program around the time that Steve arrived in 1970.

Allan Borodin, who arrived a year before Steve and eventually became the department's fourth chair, recalls the atmosphere that they all worked to build. "We tried to make very careful appointments, and once a person gets here we want them to really enjoy being here—we want them to want to stay and we want them to succeed. It was a remarkably supportive environment. In the early days, Tom Hull had us over for dinner once a month. It wasn't just that he did a great job recruiting, he did a great job of making you want to stay. What I always say when I'm trying to recruit someone is 'your success is my success,' and that is an attitude I inherited from Tom and all of the senior people here."

That unique environment and approach continued for many years. Toniann Pitassi was a doctoral student supervised by Steve in the late 1980s, and she eventually returned to Toronto to take up a faculty position with the theory group. She says that the family-like atmosphere is truly special there, and that's why she returned to join the faculty. "They just created this amazing environment of cooperation.

Everybody was interested in each other's research and working together on problems and thinking and sharing. I've been to many places, and I came to realize that the situation in Toronto in the theory group was really exceptional and unique and I might not find it anywhere else."

Silvio Micali, who would later be the co-winner of a Turing Award for his work in cryptography, spent a year at the University of Toronto as a postdoc in 1982. He was immediately struck by the level of interaction among the faculty, and the students, in the department. It was an approach that had been developing since the late 1960s when the department was formed. Silvio did his undergraduate degree in Italy, where students generally don't live on campus, but he strongly preferred the North American system because it creates an atmosphere of collaboration. He says, "In Europe you go to classes and then you go home. In the American system it really works because there's true immersion. In Toronto, the immersion also extended to faculty. Yes, everybody went back home, but we had communal activities and research ideas were tossed around. It was as close to full immersion as it could be for faculty members. It was an unusually friendly group. It really set a standard for me, how a group of colleagues should be interacting. We had personal dinners, communal lunches, and activities together."

It was a great place for a young professor like Steve, and Linda also became a fixture on the university's campus. She worked at the registrar's office until the late 1980s. It became clear that Toronto was going to be their long-term home as they settled into their new lives and became friends with their colleagues.

Steve started supervising graduate students once again and became a popular lecturer as well. Allan Borodin says, "Steve, early on, was getting rave reviews as an undergraduate lecturer. He's a careful lecturer, and very informative."

Toniann Pitassi remembers his lectures similarly. "He's very understated but extremely clear, and he says things very succinctly and is always accurate. He taught this one course, and I teach it now, a course in logic and computability. And he developed his own set of lecture notes for it, and he was famous for that course—both because it was only him that ever taught it and because it was his lecture notes. Everybody knew that it was one of the hardest courses, but also one of the best courses, the most enlightening."

In addition to his own lectures, Steve took an interest in what other professors were teaching and often attended seminars and lectures by his colleagues, like he had at Berkeley. When Silvio Micali was in Toronto for his postdoc, he was working on zero-knowledge proofs, which would become very important for modern cryptography applications. "I was very surprised that Steve came every single day and took notes and attended lectures," he says. "So I felt it was a testament to the man that it was not only a question of discussions in his office once or twice, but

actually he really wanted to understand things. But the best that I got from him was to have him in the audience. I really thought his questions were targeted, and always very enlightening."

In fact, Steve was so interested in Silvio's area of research that he worked on some of the assignments that Silvio gave his students. Silvio says, "I gave homework—essentially the topic was zero-knowledge, but at the time there was not a fully distilled definition. So I gave a very vague definition for the homework and later on Steve invited me to discuss it because, he said, 'I have no idea how to go about solving this exercise that you gave.' So if I think something is obvious, and Steve doesn't think it's obvious, maybe it's not obvious! All of a sudden I realized I really need to figure out (a) there was a new definition of a proof to be distilled, and (b) I have to be formal about it. Essentially, he made me realize how significant this advance was."

Silvio worked extensively on this topic with another professor in Toronto's theory group, Charlie Rackoff, but Steve's outside perspective provided valuable guidance on how to present the concept to the computer science community. "Steve helped me figure out that there's something big here, waking up the sleeping giant that I was trying to start a new direction. And, with Charlie, we really formalized and defined it properly." This type of collaboration and support was typical of the department and helped to propel their research, and careers, forward.

Steve was not the only faculty member who attended other professors' lectures. Many of the faculty members also attended Steve's courses and participated actively. Bruce M. Kapron says, "I remember attending one course that Steve taught with Russell Impagliazzo—it was a fairly advanced course on logic and its connections with complexity theory. This was a very advanced topics course, so it was right on the edge between teaching a course and just sort of having a seminar that would lead immediately into talking about research problems. All the grad students would be there, but often the other theory faculty would attend the courses as well. So you'd have Charlie [Rackoff] and Al [Borodin] and maybe Faith [Ellen] sitting up there in the front row, and they'd be carrying things on at a high level. In a way it was kind of intimidating when you first get there to be in these courses where the brightest 'students' are up at the front, and they're the other faculty members!"

As the study of theoretical computer science grew, sharing new ideas at conferences became an important aspect of academic life. Steve has been presenting papers in the ACM SIGACT (later called STOC, or Symposium on Theory of Computing) conferences since they began in 1969. He had no notion that the paper he submitted in 1971 for the third annual conference, "The complexity of theorem proving procedures," would earn him a place in computer science history and a number of prestigious awards.

His paper discussed the concept of NP-completeness, but Steve admits that it was actually not in the original draft he submitted to the conference committee, and history could have taken a very different turn if that draft had endured. "What I sent to the committee to decide whether they're going to let me present at the conference, that did not have NP-completeness. But by the time I published I had thought of this idea; so that got put in the actual paper. I was one of the people who invented the notion of NP-completeness, but I didn't have that when I submitted the paper."

Thankfully, the version Steve presented at the conference did include NP-completeness, but it was far from obvious that this was the key element in his paper. Allan Borodin admits that the significance of Steve's result was not immediately clear to him. "Steve told me that he was submitting this result to the STOC 1971 conference, his NP-completeness result. It seemed like a nice result, but he had other things in the paper, he was talking about proof complexity, and I wasn't much of a logician. I was on that [STOC] program committee, and a lot of people didn't quite understand the significance of that paper."

Dick Karp remembers that it was something he had to discover within the text. "It was a funny situation, because this gem of a result of the completeness of SAT was buried in a not very evident part of the paper. There was a lot of other stuff in there. It leaped out at me immediately because I was familiar with some of these other problems, and I was immediately convinced that there were many problems, if not most, that could stand in for SAT in terms of being complete."

In fact, had Dick Karp not been immediately interested in pursuing Steve's result and determining which other problems were NP-complete, the paper might have had much less impact on the field of complexity theory. Steve was not the sort of person to go around bragging about the impressive work he'd done, and it was fortunate that others brought attention to it. The next year, Dick published a paper showing that he'd found many more problems that were NP-complete. Steve says, "Dick Karp saw what I had done, the notion that certain problems are NP-complete, and what he did was find—I think I had 3 examples [in my paper], and I think he had 20 and showed that this was really important."

Allan Borodin remembers how the notion of NP-completeness soon became an important topic in the complexity theory world. "It caught on so quickly because of Dick Karp, who at this point in time was a full professor at Berkeley and much better known. He followed up on Steve's work and between those two papers it was just—it started to spread like wildfire. In that period of time, in the 1970s, every conference was featuring more and more NP-completeness results, along with people trying to prove that P does not equal NP. Not successfully of course."

The growing popularity of Steve's paper affected the atmosphere at the university and in the department, according to Allan Borodin. "The dynamics changed because all of a sudden we had a star in our department. People knew already how profound it was. There was a big textbook written by Garey and Johnson, I think in 1979, just about NP-completeness, a whole book on NP-complete problems." That textbook, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, contains more than 300 NP-complete problems and continues to interest students more than 40 years later.

The students at the university were also attracted by the idea that one of their professors had "discovered" the very thing they were learning about. According to Allan, "at some point in time, all the students, every undergraduate in math and computer science knows about NP-completeness. So when you're taking a course from Steve, you're taking a course from the expert. 'The guy.' People really appreciate that they're in a class with him."

Jim Hoover was an undergraduate student at the University of Alberta in the mid-1970s who would later be supervised by Steve at U of T. He remembers learning about Steve's work in one of his undergraduate courses and how it helped to build the reputation of the strong theory department at the University of Toronto. "I took a graduate course on computability theory. About one-third of the course was on complexity theory, NP-completeness, and so on. So Cook's theorem was a topic. Everyone I talked to said that if you want to study theory, Toronto was one of the best places in the world."

The question of whether P is equal to NP has long been associated with Steve's work and has challenged mathematicians and computer scientists for almost 50 years. It is such a difficult problem that the Clay Mathematics Institute at Cambridge made it one of their "Millennium Prize" problems, which is a list of the most important open problems in mathematics. Each of the seven problems on the list is attached to a $1 million prize for anyone who finds a solution.

Although it's certainly nice to have his work featured in such a high-stakes contest, Steve says there is a downside to having a million-dollar prize available over an ever-growing number of years. "The effect of that is to get inundated with messages from people who claim they've solved it, which is sort of tiresome, so I'm kind of nasty about that. About half of them have 'proved' that P equals NP and half have done it the other way—and recently one person claimed he did both!" It's unclear what Steve's version of being "nasty" entails, but it likely involves a polite response explaining that their solution is not correct. He also recalls one person who "had a program for solving the 'Mine Sweeper' problem, which is NP-complete. He didn't know what to do with it and he didn't want to tell me the algorithm because he was afraid I would steal it and take the million-dollar award" [CBI02].

The other high-profile result of Steve's work in NP-completeness was his ACM AM Turing Award, which he won in 1982. It was still quite early in his career, a little more than a decade after he'd become a tenured professor, which is a strange time to achieve something that is considered the pinnacle of one's profession. Some of Steve's more senior peers, including Dick Karp and Velvel Kahan, would receive their own Turing Awards after Steve, later in the 1980s. For the computer science department at the University of Toronto, it was quite a coup.

Allan Borodin says, "Everybody was really excited because everybody knows the importance of the Turing Award—everybody in computer science. (And now lately everybody outside of computer science.) It was clear anyway, but that makes it much more official or documented, that this is a profound result and that he deserves so much credit for it."

Outside the computer science department, however, the enormity of Steve's achievement was not well understood. "When I heard about the Turing Award, I called up the person [at the university] who I thought was in charge of promoting these things," Allan continues, "and I'm babbling on rapidly, I'm so excited about this. I knew how much it meant—not just to Steve but to the department and the university, and in fact, to the country. For a long time, Steve was the only Turing Award winner in Canada. Now we have Geoff Hinton [a University of Toronto professor who was a co-winner in 2018]. The importance of this to everybody, I thought, was pretty clear. And she stopped me and said, 'How much is the award worth?' Well, the award in 1982 was worth $2,000, I think. And she said, 'Sorry, I only handle awards of $10,000 or more, I'll give you somebody else to talk to.' I said, 'You don't get it, this is the Nobel prize of computer science,' but she said, 'I'm sorry, this is what I do.'" There would be no such difficulty now since the award is currently worth an inarguably significant $1 million.

Steve himself was typically understated about the award. "I think people were happy. It really helped the department, to be honest, because now there's a Turing Award winner. The Turing Award wasn't as big a deal then as it is now—now you get $1 million and I think I got $1,000 and a nice silver platter. But it was enormously helpful to get it," he says. Nobody seems sure whether the award was $1,000 or $2,000 at the time because the significance of the win didn't come from the money but from the prestige associated with the award.

Thankfully, not everyone at the university was reluctant to celebrate Steve's success. There was a party in the computer science department that was attended by the university's president and other key people. There was also a more personal event at Allan Borodin's home, where Steve and his peers, and even some students, gathered to celebrate. This happened to take place while Silvio Micali was at the

University of Toronto, and he remembers the event having a familial feel, which reflected the close ties between everyone in the department.

Silvio recalls that when the award was announced, Steve didn't feel quite right about claiming the spotlight for himself. He was uncomfortable taking full credit for the influence of NP-completeness without an equal acknowledgment of Dick Karp's role in bringing the breadth of NP-complete problems to the attention of the research community. Silvio says, "He mentioned that he felt that a better outcome would have been if he and Dick got the award together. He really felt strongly about Dick's contribution, too." Dick Karp would not have to wait long for his own Turning Award. He was named the winner in 1985 for his wide-ranging accomplishments, including his contributions to the theory of NP-completeness.

Allan Borodin says that Steve's humble approach to his research and his students was unaffected by the award and the fuss surrounding it. "I don't think it impacted his personality one bit. He is remarkably unassuming and modest. I think he wanted to continue to work on the problem and related problems in complexity theory. I don't think he ever thought this is the crowning achievement of his career."

1.5 The Quiet Influencer

Steve appreciated the recognition he received as a Turing Award winner, but in reality it changed very little for him on a personal level. His first son, Gordon, was just four years old in 1982. His second son, James, was born in 1985. They were a priority in Steve's life, and his status as an award-winning researcher was not more important than his role as a father.

Even though he was working hard at the university, he made sure to get home and spend quality time with the family, eating dinner together and helping to put the young boys to bed. James remembers his father reading stories about sailing to him at bedtime, from a series of books called *Swallows and Amazons*, by Arthur Ransome. Steve's earlier years on a farm also made him partial to reading books to his boys by Laura Ingalls Wilder, best known for her *Little House on the Prairie* stories.

The family went sailing on their boat around Lake Ontario and took active vacations where they would go climbing in the mountains. They would also visit Steve's family in New York state so that the boys could spend time with their grandparents.

Steve shared with his sons his love of mathematics, computer science, and sailing. Gordon was particularly keen on sailing, and he started competing in races at an early age. His passion for the sport had the family attending regattas on a regular basis as Gordon was growing up. He reached an elite level and represented

Canada in the "Optimist Worlds" competition on two occasions. After completing an engineering degree, he focused on sailing once again and competed in the Olympic Games in 2008 in Beijing, returning in 2012 in London, which made him the only sailor to represent Canada twice in the 49er class.

James also sailed, but he followed more closely in his father's academic footsteps. Having a parent in a computer science department made James an early example of a "digital native." They had a personal computer in the house in the 1980s, at a time when few parents were well-versed in computers—especially in programming. In fact, James remembers learning the alphabet on the computer keyboard as a preschooler. Lessons in creating simple programs followed soon after.

"I got started with really basic things when I was seven and a half or something," James says. "At some point we were writing a program together to translate English words into 'pig Latin.' We messed it up in that we'd forgotten to put in something for deciding when the program would stop. You were supposed to type in 'stop' for it to stop—except that, the way we'd done it, you had to type something that would turn into 'stop' in pig Latin, and 'stop' isn't a word in pig Latin, so it was impossible. But the code was kind of buggy, so he managed to type in a word that would turn into 'stop' and end the program."

Steve had felt a certain pressure from his own family to excel academically when he was growing up. A higher education, including graduate studies, was expected of Steve and his three brothers. But Steve and Linda made a point of not having unrealistic expectations that their sons should succeed at any cost, either in sailing or in academics. James says, "I think my parents were pretty careful about trying not to pressure me and my brother. In hindsight, I think they were pretty happy, especially my dad, that I went through college and grad school and everything."

In fact, James ended up studying computer science at the University of Toronto, and even took a couple of his father's courses. He confirms that the calm, careful Steve Cook that students saw in the classroom was the same person he saw at home. After completing his undergraduate degree, James then went on to graduate school in the very department that Steve hadn't managed to join as faculty—the computer science department at UC Berkeley. But rather than pursue an academic career like his father, James went into the industry workforce, landing a job at Google.

Steve, meanwhile, continued his role at the University of Toronto. Things returned to the normal routine soon after he had received the Turing Award. In fact, Steve's career has remained remarkably focused; for more than 50 years, he has explored a set of fundamental topics that continue to fascinate him. He says, "There are just two directions I go. One is theory of computation, which always is trying to prove that problems can't be solved easily, which is always interesting. And

there's the related problem of mathematical logic, and there the interest is theorem proving. So the question is how easy is it to prove something: Are there methods of finding proofs automatically? Proofs are easily recognized, but the problem is how easy is it to find it, and how long is the proof. These questions are all related."

One of Steve's former doctoral students, Bruce M. Kapron, says "Steve was pretty driven by deciding for himself what was interesting and continuing to pursue that, rather than being influenced by trends and what was currently considered to be important or 'hot.'" He explains that Steve's work builds on "the foundations of mathematics that happened in the early to mid-20th century—there was this explosion of work with Church and Turing and Kleene and those people. In some ways it's a continuation of the kind of foundational questions that those people were asking, but with the added idea that there's a cost to computation. This fundamental work thought about what's computable in principle, without any concern about resources. But Steve was always very interested in the cost of computation. What does this mean for logic? How do we address this using logical tools? Or tools that come out of computation theory? He's sort of the modern descendant of those people who were doing that early work on the nature of computation and the foundations of mathematics."

Toniann Pitassi has seen firsthand that Steve's focus is firmly on his research, not his notoriety or the progression of his career. She says, "The driving force behind him is his research, and the research questions. Even now, he's still passionate about the questions. So he doesn't let his reputation get in the way of that. He doesn't promote himself, he doesn't get worried about whether he gets a grant or whether people cited him. Most people want to make sure their career is going well and that they're getting credit for things. When they give a talk, they'll make sure that they cite themselves, and I've never seen him do that at all."

According to Allan Borodin, this extends to whether his name even appears on papers that he contributed to. He says, "Steve definitely wanted the rest of us to succeed also—it was never about him making sure he was getting his credit. I had an early result, an influential result, where Steve really improved my result and made it much more attractive. And he said no, no, no, I don't want my name on it. I think having somebody like Steve in the department, with no pretentions, makes it very hard for anyone else to have pretentions."

Steve's work did get noticed within the theoretical computer science community, even without self-promotion. In addition to the Turing Award, Steve has accumulated a number of impressive honors. In 1999, he was awarded the CRM-Fields Prize (now the CRM–Fields–PIMS Prize), which is the highest honor for mathematics research in Canada. Bruce M. Kapron thinks that receiving that prize was particularly special to Steve. He says, "I think what Steve did appreciate was getting

recognition, not just from the computer science world, but from the mathematical world." The award may have provided a form of validation—or vindication—after the Berkeley mathematics department had let him go all those years before.

Steve's prizes sometimes benefited more than just his own career. For instance, in 2012 he was awarded the Gerhard Herzberg Canada Gold Medal for Science and Engineering, which comes with a research budget of $1 million, distributed over a five-year period. The research budget is shared within the department, so Steve's colleagues and the graduate students all had reason to celebrate when he was recognized with that award. Of course, having an established and respected researcher like Steve on the faculty also makes it easier for the department to get other research grants, all of which get shared among the faculty. Steve's growing list of honors was therefore good for everyone.

Steve is also a fellow of the Royal Society of Canada and the Royal Society of London and is a member of the National Academy of Sciences (US), the American Academy of Arts and Sciences, and the Gottingen Academy of Sciences. Although he doesn't dwell much on these honors, he does admit that some of them come with rather pleasant perks. Steve says that, as a fellow of the Royal Society of London, "one of the effects of it is that when Linda and I go visit London, the Royal Society buildings have rooms for fellows to stay in, and they're very nicely located right next to the palace. So we always stay there." Likewise, Steve and Linda annually travel to the lovely town of Heidelberg, Germany, to attend the Heidelberg Laureate Forum, an annual event that brings together former recipients of the most prestigious awards in mathematics and computer science: the Abel Prize, ACM AM Turing Award, ACM Prize in Computing, Fields Medal, and Nevanlinna Prize. There, the laureates meet young researchers and provide mentorship and inspiration and deliver lectures on a variety of topics.

Steve has continued to stay focused on his research and on mentoring upcoming researchers. Silvio Micali says that Steve's approach to research was influential early in his own academic career. "Steve's style was to publish few and excellent-quality contributions. Steve averaged about one paper per year, and each of his papers was very thoughtful, thought-provoking and people paid a lot of attention. It was very helpful to me to see that quality over quantity is an important thing." In fact, Silvio explains, if it weren't for the consistently high quality of Steve's papers, Dick Karp may not have been an early enthusiast for NP-complete problems. "Dick said that Steve Cook was on his short reading list—a short list of people whose papers he read no matter what they were about. The paper about NP, if it wasn't Steve who wrote it, he wouldn't have read it."

Certainly Steve's slow and thoughtful approach to publishing is unusual in the "publish or perish" environment of modern academics. There are many

researchers who feel a need to put results out regularly and rely on a certain number of citations to cement their importance; Steve has instead taken his time with each problem and published only when he thought there was a significant result to share.

Toniann Pitassi describes him as "both lightning fast and very slow at the same time. He thinks about things for a very long time, and in some ways it's slow because he doesn't publish a lot of papers—but when he does, they're really important. But he's lightning fast with his intuition [about which problems are significant]. So I've learned to appreciate that you don't have to be the fastest person and produce hundreds of papers all the time. You can take your time, and if you have something really interesting and novel to say, even if it takes a long time to get to that point, that's worthwhile."

He has passed this approach on to his students over the years. Toniann says that having Steve as a supervisor gave her the space to really explore her research without feeling rushed or panicking about getting a paper out. "I remember as a grad student wondering why he never asked me to publish papers, but it's not the way he worked. It's very uncommon in research. Most people want to publish papers and want to publish in top places. I never felt like that was an issue with him. I remember one time, after a couple of years—because his area is very theoretical even within theory of computer science—I remember worrying, how am I ever going to get a job in this? Because all of these other people have many more papers, and papers in areas that are more applied. And I remember he just said that he thought that you have to work on what you like, and if you make a little bit of progress on a really hard problem, it's just as good or better than making more progress on a not-so-hard problem."

It was a big relief for Toniann, who was insecure at first about her background (or lack of it) in computer science when she started her doctorate. "I was an undergrad in chemistry, so I didn't have the background that a lot of other people had. You were assigned to an advisor. I don't think that they even asked me who I wanted and, truthfully, I was so intimidated by Steve that I would not have even put him down. But I was assigned to him. And I remember being thrilled and terrified at the same time. But he was so nice and so sweet that I didn't even consider switching."

Bruce M. Kapron had been more direct in his approach to working with Steve. He completed a master's degree in mathematics at Simon Fraser University in British Columbia, Canada. His interest was in mathematical logic, but his professors encouraged him to pursue the computer science aspect of logic because it offered more career opportunities. He was also encouraged to submit some of his work to the *Journal of Symbolic Logic*. "Steve happened to be on the editorial board. You have to submit it to some editor, so I submitted it to him. And there

were these very good referee reports that came back, so I think he probably said, 'This looks like somebody who would be a good person to work with based on their background in logic.'"

Bruce says that having Steve as a supervisor was a great experience because he doesn't set himself above his students. Instead, he treats them as fellow researchers. "I felt like I was working *with* Steve, not working *for* Steve. We'd come in and it would be the same kind of interaction that he would have if Charlie [Rackoff] or Al [Borodin] walked into the room and were discussing some problems. He didn't act like I was a student, he sort of acted like I was just as much an expert at this as he was. So that was really great. Really intimidating, but still a really great way to work with somebody."

Steve is known for his lack of pomposity. To him, everyone is deserving of patience and respect. Toniann says, "the way he treats students, whether they're grad students or undergrad, he treats everybody the same. He gives everybody respect. And if you're interested in the problems he's interested in, he will give you lots of time and be happy to talk to you." He not only supports his students through their research but also celebrates their successes. Whenever one of his graduate students completed their Ph.D., Steve would invite them for a dinner at his home. Often, he would also take them out sailing. The "your success is my success" attitude that had attracted Steve to the department had clearly become part of his own approach to supervision.

Jim Hoover completed both his master's degree and his doctorate in Toronto and was assigned Steve as his master's supervisor when he arrived in 1978. He found that working with Steve was a very enriching experience because graduate students were treated like "junior colleagues" by the supervisors in the group. "There wasn't really a supervisor–student relationship, at least within the theory group. You were expected to pull your share of the intellectual enterprise. One thing all of Steve's students learned was that you should never be satisfied with a result. You should push it to see if you could make it better, or more general, and you should think about its implications on other work. He's always been a 'quality over quantity' person." One paper that Jim worked on with Steve and Paul Beame had already been accepted for publication when they realized that they could make it significantly better. "Steve insisted that we withdraw the original manuscript since it was no longer relevant considering our new result."

Working with Steve over several years, Jim got to know him quite well. He says that people's impression of Steve as a truly nice guy rings true. "My overall impression of the classic gentleman and scholar was consistent over time, it just became fuller. That says a lot because it means that Steve was always the genuine thing."

While Steve has always been a dedicated researcher, he has not shied away from mixing business with pleasure. The students in the theory group were often recruited onto Steve's sailing crew—including Jim Hoover, who had just learned to sail during the summer before he started his master's degree. Steve was a member of the Royal Canadian Yacht Club in Toronto and participated in both long-distance races and course races with his student crews.

"I have to admit that in the summer I probably saw Steve more at the yacht club than at the university," Jim says. He discovered that Steve's approach to training his crew was not much different from the way he supervised students. "Sailing is all about making and learning from your mistakes. After a race we would debrief about what went right and what went wrong, and how to do it better next time. That's a practice that you can apply to all aspects of life." Jim elaborates, "If we were out just sailing, Steve would often hand the boat over to you so you could learn how to skipper. He would happily follow your dumb orders and let you make mistakes. He would intervene only if the boat was at risk. On the other hand, in a supervisor situation, Steve would let you go off and follow wild ideas that often would end up as unproductive tangents. This was all part of the process of learning to do research. So in that sense, his method of teaching was the same for both sailing and research."

Steve's son James says that sailing brings out Steve's competitive side in a way that doesn't often show in other facets of his life. Although Steve is generally calm and patient, during a race things are a bit different. James says, "there's not really any other activity I do with him where there's urgency, and something needs to happen quickly. It's not like he'll be mad if anything goes wrong, but there's a sense of, we'd better get this done quickly and right so that we don't fall behind in the race. You don't really feel that from him in any other situation."

1.6 Profound and Complex

Over the course of his career, Steve has supervised or co-supervised 35 doctoral students and taught courses for thousands of undergraduate and graduate students. That alone would be a strong legacy to leave the theoretical computer science community. But Steve's influence has gone far beyond those who have had the benefit of learning from him directly. Thanks to his foundational work in complexity theory, many of his contemporaries have credited him with changing the entire field and opening doors to new areas of study.

Bruce M. Kapron believes that Steve's work has had a significant impact on the areas that other researchers are now studying. "Theoretical computer science now is a really diverse field," he says. "The kind of problems that people are working on in complexity theory now, they may not be directly the P versus NP

problem anymore, but that whole approach started to grow with the theory of NP-completeness. That was the first real new contribution in complexity theory that was deep and serious. A lot of the work that gets done, even if it doesn't cite Steve, is somehow a legacy of his work. It really permeates a lot of things. It's interesting because he's had these papers that led to whole fields, like his papers on proof complexity, like the work that he did with Bob Reckhow."

Toniann Pitassi agrees that Steve's research has been remarkably influential, in many cases through results that are not well known. "I feel like what he'll be remembered for and what I consider his legacy are very different things. Partly that's because he's so modest, and he has these amazing results that are not the one that he got the Turing Award for. They're even better, but it might take another 20 years for people to notice. Right now people remember him for his NP-completeness result. But I think he has this body of work in feasible mathematics that's fascinating—a big program that's been very influential. People who know it think it's amazing, but it's a small number of people. I think it's slowly being realized just how profound it is. And I don't know if Steve will get the credit for it because he doesn't promote himself, but I think he should get the credit for all sorts of things that have happened in the last 20 years: directions that theory has taken off in, that really when you look back to the origins, they're his work."

Toniann says that one of the secrets to Steve's influential body of work is his ability to choose the right problems to work on. "He's very intuitive—that's one thing about him, amazing intuition! He'll know immediately, even if he doesn't understand any details, whether it's true or whether it's false or whether it's fishy. He just has this amazing high-level intuition both for which are the interesting questions and whether a proof has any shot at being correct or not."

Jim Hoover believes that Steve is a strong role model for academics because he has achieved so much without sacrificing his life outside of academia. "So many academics' entire existence depends on being an academic—they are lost if they have to do something else. Steve shows that you can be an amazing researcher yet not have that consume your life. You can have hobbies, and a family, play music—and just generally enjoy life."

Perhaps Steve's true legacy over the more than 50 years of his career is seen in the influence he's had on his students, his colleagues, his peers in the computer science community, and his family. He has embodied the values of hard work, authenticity, egalitarianism, intellectual rigor, generosity of spirit, community, and a balanced lifestyle. Steve Cook is much more than the list of papers, awards, and honors on his CV, and those who have had the experience of studying under him, working with him, or sailing with him will continue to spread his influence wider when they reflect his values in their own lives and work.

2 ACM Interview of Stephen A. Cook by Bruce M. Kapron

BK = Bruce Kapron (Interviewer)
SC = Stephen Cook (A.M. Turing Recipient)

BK: Hello, this is Bruce Kapron. It is February the 25th, 2016 and this is a recording of an interview that I had with Professor Steve Cook at the University of Toronto, where Steve is University Professor in the Computer Science and Mathematics Departments. This is part of the ACM Turing Award winners' project. Professor Cook received the ACM Turing Award in 1982 in recognition for his contributions to the theory of computational complexity, and in particular the theory of NP-completeness, which he introduced in his 1971 paper "The Complexity of Theorem-Proving Procedures." So now what you'll be seeing is my interview with him.

SC: Hello.

BK: You were born and you grew up not far from Toronto in Western New York. As a child, did you already have an interest in science or mathematics?

SC: Yeah. Well, I certainly had an interest in science. My father was a chemist and worked for a Union Carbide branch, so I was interested in science. I was good in mathematics, but I didn't think in high school that that was going to be my chief interest.

But the big influence on me when I was in high school was Wilson Greatbatch, who was a resident of Clarence, New York, same as me. He was an electrical engineer, but a very creative one. Transistors were a very new thing then, and he designed a transistor circuit that went "Bip, bip, bip," and eventually turned it into an artificial pacemaker for hearts which was implantable. That had never been done before. Well, of course you couldn't

This chapter contains a transcript of an interview conducted on February 25, 2016, on behalf of the ACM. A recording of the interview may be found at the ACM Turing Award website: amturing.acm.org. The transcript has been lightly edited for clarity.

do it with vacuum tubes, obviously. So he eventually got ushered into the Inventors Hall of Fame in the United States for inventing this thing.
But while he was doing this and I was in high school, I helped him out. He had a little shop in the top of his garage in which he worked. He would draw pictures of transistorized circuits and I would solder them together. Then he'd try them out and I could see on an oscilloscope this "Bip, bip, bip."

BK: This was the late 1950s?

SC: That's right. That's right, because I was in high school. So like I graduated from high school about 1957. Then I don't think it actually got turned into a pacemaker till some years after that. Also, he worked for an electronic firm and he got me summer jobs there. So I was very interested in electronics and I thought that was going to be my profession.

BK: You mentioned your father. And your mother was also a teacher?

SC: Yeah. Well, yes.

BK: She was a teacher, yes?

SC: Yes. Well, she had two master's degrees. My father and mother met at the University of Michigan. She got a master's degree in English and history then I guess. Yeah. And of course where my father got his PhD in chemistry. Then later on when... I have three brothers and she was mostly taking care of us, but when we were all off, going off to various places, then she was a teacher. Oh, she got a second master's degree in English and then she was teaching at a community college in the area.

BK: So you mentioned that you had quite an early introduction to I guess at the time what was very high technology electronics. What was your first exposure to computers and computer programming?

SC: Yes. Even in high school, I had a good math teacher who was somewhat interested in computers, and he took us to downtown Buffalo to some meeting. So I already had some idea about computers. But it really wasn't till I went to University of Michigan, which was my undergraduate school... Since both my parents were alumni, there seemed to be no choice. My very first year, which would have been I guess '58, the spring term, I took a course, a one-hour course in programming. This was from Bernard Galler. We learned how to program, and I think it was the IBM 650, which was a vacuum-tube machine whose memory was on a drum, just to put you in the category of what computers were like then.
That was probably my real access to computers. But I enjoyed it a lot and I started writing fun programs like... I remember early on, I wrote a program to test Goldbach's conjecture, which was that every even integer is the sum of two primes. So I tested it up to some large number and it turned out to be verified.

BK: And at Michigan, you started off in electrical engineering?

SC: That's right, I actually start-... It was engineering science I think, but I was thinking in terms of electrical engineering. But my very first year I took a calculus course from Nicholas Kazarinoff. It was I guess not just the completely typical calculus course, slightly more advanced. Anyway, I got quite interested in the course and Kazarinoff got interested in me. He would give me special homework theorems to work on. In fact, he was so impressed that he said I could skip the second-year calculus course – which I kind of regret now because I never learned two-variable calculus – and go on... there was a third-year course. And also then I took a third-year course in linear algebra. So yeah, I got off to a good start in mathematics.

BK: And eventually you switched into...?

SC: That's right. After... I think I took two and a half years, and then I finally switched out of engineering and became an official math major. That's right.

BK: So you were successful enough that you ended up going to Harvard for your graduate work.

SC: Yes, indeed. Yeah, yeah. I was quite excited about that. I must have gotten good reference letters. So yes, I guess it would have been '61 I joined, I became a student for a master's degree in the math department. I got that in '62.

Then I needed a thesis advisor, but I'd taken a logic course from Hao Wang. He was not in the math department. He was in the Division of Applied Physics. But I mean he was really a logician. Logic was his major interest. But he also was interested in computers. And before coming to Harvard, he worked I guess for both Bell Telephone Labs and the IBM Watson Research Laboratory in Yorktown Heights, New York. There he wrote a program to prove theorems in propositional calculus automatically. That was one of his interests, trying to make automatic theorem provers. It apparently was very successful because he used an IBM 704, which is another vacuum-tube machine, and in just three minutes it proved all the hundred or so propositional tautologies in Russell and Whitehead's Principia Mathematica. That was a big deal because other people, Shaw and Simon had tried to make automatic theorem provers for propositional calculus and apparently they weren't very successful.

BK: Professor Wang was also very interested in foundational issues in mathematics in logic and computability. I'm wondering... It seems that both with the automated theorem proving and with his interest in

computability, that really had a big influence on a lot of the work that you've done.

SC: That's right. I mean it certainly did. There was no question about that, that yeah, he was interested in the *Entscheidungsproblem* of Hilbert, which is the problem of determining whether a given predicate calculus formula is satisfiable or not. Then of course Turing and others proved that it's unsatisfiable. So then how simple a formula, class of formulas can you make to make it still unsatisfiable? And there was the so-called AEA case, for all exists for all case, and then proved it was still unsatisfiable. So he was part of that. And yes, that did have an influence on me later on.

BK: But computational complexity has always been a theme in your work. I'm wondering if you can say a little bit more about what your PhD research was based on.

SC: Ah! Yeah, okay. So some background for that. Computational complexity was a new subject of course, and this was in the early 1960s. In '63 I think it was, Hartmanis and Stearns published their famous paper in which they introduced the word "computational complexity" on "The Computational Complexity of Functions" or something like that. So Hartmanis came to Harvard and gave a talk, and so I became quite interested in that.
Then the other influence was by Alan Cobham. Alan Cobham had a degree from Harvard and he was a graduate student there. He wrote a thesis, but he never got his PhD because the math department, in addition to a thesis, it requires a minor thesis and Alan never bothered with a minor thesis. But he went off to work for, again, IBM Watson Research Lab. He was connected with... He was a friend of Hao Wang and he would come, so I got to know him. His famous paper was "The Intrinsic Computational Difficulty of Functions." That was his paper he wrote. He was interested in... I think this was really before Hartmanis–Stearns, yeah, that happened, but he was interested in "In what sense are some computational problems harder than others?" and he compared multiplication and addition as an example, "In what way is multiplication harder than addition?" But his result, which... I mean one thing he did that really had an influence was he introduced the notion of what we would now call polynomial-time computable functions. He argued that this is an interesting class, a complexity class, because it seems to be all feasible functions, and if you're not polynomial-time computable, you're not going to be feasible. He made that argument. Then he also came up with an interesting characterization of the polynomial-time computable functions. Namely a function is polynomial-time computable if and only if it can be obtained from certain

initial functions and applying the operations of composition and limited recursion on notation. And that's something he made up, a new kind of recursion. Primitive recursion of course had been well known for many years, but to characterize the poly-time computable functions, you needed this other notion, limited recursion on notation. So that also had an influ-... I was very interested in that.

BK: But in your PhD, you considered the question of the difficulty of multiplication.

SC: That is correct, yes. Maybe that was because of Alan Cobham's question. Yes, of course. It's very hard to prove lower bounds. We all know that now and that was certainly true then. The only way you'd get a lower bound on multiplication was to look at the case where the inputs are only read... What do you call that? So the inputs are restricted so that after each digit is given, you'll have to give the output for the multiplication.

BK: Oh, it's...

SC: Online. I'm just thinking online. Sorry for the... Yeah. So this was an online model of a Turing machine, which you think of an unlimited number of digits. You're multiplying two numbers in decimal or binary, whatever, it doesn't matter. So the first n, string of n digits would be the n least significant digits in the two numbers, and that's enough information to get their product. So you had to output their product and then go on to the next two digits and then keep outputting the product and so on. That was the online model. In that model, I was able to get a lower bound of $n \log n$ over... just over $n \log n$. What was it? I can't remember exactly. $n \log n$ over $\log \log n$ I think it was. Yeah. So that was a non-trivial lower bound. That was part of my thesis. The other parts again talked about computational, number theoretic, or real function problems.

BK: After graduating, you became a faculty member in the math department at the University of California, Berkeley.

SC: That's right.

BK: What year was that?

SC: That was 1966. Yeah, so I finished my PhD in 1966. I had been offered a job. Now the job was half mathematics in the math department and half in the... It wasn't a computer science department, although they did have a budding computer science department just starting. It was a research job in the Computer Center or something, something like that, so it was only half in mathematics. So yes. And so I started that out in the fall of 1966.

BK: So you were in Berkeley in the late '60s, which must have been an interesting time intellectually and culturally as well.

SC: Indeed. Of course yeah. This definitely was... The free speech movement was in full form and there were crowds of people there. In some cases, the police were called in. There was teargas and all kinds of stuff. Yes, all that was going on in the '60s. Yes.

BK: As soon as you got there, I assume... or even, as you say, working on your PhD you were already thinking of a lot of questions about computational complexity and even thinking about polynomial time, so...

SC: Yes. Well, of course yeah, the polynomial time definitely came from Cobham, so I was interested in that. Early on, I did circulate a mimeographed set of notes on something like classes of primitive recursive functions, so there were complexity classes of primitive recursive functions. So there was work at Princeton at the time. There was a logician at Princeton, and I had read parts of Bennett's thesis.
Anyway, his thesis was called *On Spectra,* which was actually a predicate calculus sign, but it had a lot of complexity theory results based on logical... So there was kind of two different, yeah, classes of people then. I mean these were the logicians doing complexity theory. So he didn't talk about Turing machines ever that I remember, but I remember one of the complexity classes he talked about were the extended positive rudimentary relations, the class of extended positive rudimentary relations, which had some fancy definition, logical definition. Then I eventually realized, "You know what this is? This is nondeterministic polynomial time." The same characterization. So that's probably the first time I became interested in what we now call "NP."

BK: And what year was that?

SC: That would have been, well, '67, because I circulated this stuff, it has the date on it, 1967. So...

BK: So then you were already aware of the question of what the relationship would be between P and NP?

SC: Well, absolutely. I even put it out in there that "Oh, this is interesting." So one assumes that there are problems in nondeterministic poly time that can't be done in deterministic polynomial time, but might be aware they're hard to prove that. And I turned out to be right on that point.

BK: And while you were at Berkeley, you were already doing work that then ended up at the ACM's Symposium on Theory of Computing...

SC: Yeah, that's right. I had a couple ACM STOC conference papers. One of them was this characterization, another characterization of polynomial time, which I think is quite intriguing. So a problem can be solved in deterministic polynomial time if and only if it can be solved by a so-called

auxiliary logspace pushdown machine. So what is this? That's a Turing machine that has read/write input tape and it has a work tape where they commonly use log-space, order log-space symbols on its work tape and do the work. But then... So that model is a characterization of what we call "logspace," which is, as far as we know, a proper subset of polynomial time. Of course we can't prove that either, but we *assume* logspace is certainly a subset of polynomial time, but... we assume it's a proper subset. But if you add the pushdown stack, I proved you exactly get polynomial time. And not only that, the nondeterministic version also gave you deterministic polynomial time. So I thought that was kind of a neat result. That got published in the ACM journal.

BK: So you left Berkeley and you came to University of Toronto in 1970. Were you recruited by people at Toronto?

SC: The story there is that the math department denied me tenure, because otherwise I would have stayed certainly, especially since I had just married my wife Linda, who was raised in Berkeley, the centre of universe, and she had no desire to leave Berkeley. So this was quite... I should say, to be fair, when I proposed to her, I told her I might not get tenure. So she was forewarned.

In any case, yeah, so it looked like I had to get a job somewhere else. So I applied at other places, including Yale and University of Washington and I think IBM Research. Then I went on a trip. But the Toronto connection, I didn't have any sense of going to Toronto, even though I have to say that I had no problem with Canada because we lived in Buffalo, we used to go to resorts, the summer resorts in Ontario, so that's fine. But anyway, I didn't think of Toronto. But yeah, in fact I was recruited. Somebody in the computer science department at Berkeley who had just come from... who had been at U of T. He was Canadian but he was recruited by the Berkeley computer science department. So anyway, he called Tom Hull, the chair of the computer science department, and gave my name and said, "Maybe you should look into this." Yeah, so Tom eventually got hold of me, not on that recruiting trip but later on. So Linda and I went to Toronto and eventually thought, "Oh, this is a great place." Actually it was a very good budding department.

BK: It must have been young at the time.

SC: Yeah, it was quite new, but they still had a number of people. Tom Hull had just moved there, but Kelly Gotlieb was there and Patt Hume. And in fact before I came, Allan Borodin who is now quite well known as a complexity theorist and also Derek Corneil was there, and he is a graph theorist,

well known. So it was clearly an up and coming department. Yeah, so certainly it turned out to be a very good choice.

BK: And making the switch to computer science seemed like a natural...?

SC: Oh, that's true. But the people who recruited me were all in computer science. Actually... So my first appointment was half in math and half in computer science, and then yes, I quickly switched to computer science, realizing the grass was greener in the computer science department, yes.

BK: You presented "The Complexity of Theorem-Proving Procedures" at ACM STOC in 1971, and that's the paper that introduces what we now call the theory of NP-completeness. How was your paper received at the time?

SC: Actually it was received very well. Yeah, I gave it to a large audience. I gave the talk to a large audience and there were people there. I remember Michael Rabin was one and he seemed quite impressed. He had been thinking along similar lines actually. So I got positive feedback from that paper, for sure.

BK: Can you briefly describe a little bit about what's in the paper, since it's not in the form of NP-completeness that we know today?

SC: For sure. Not at all. Those symbols were never uttered in my paper. So here's the story. I was interested in the complexity of theorem proving. In fact, that's the name of the paper, right? "The Complexity of Theorem-Proving Procedures," which doesn't sound very much like "NP-completeness." So when I submitted the paper to STOC, I actually didn't have my result there. I had a section on propositional calculus and complexity, and I had a section on predicate calculus, but I didn't have any really big results.

But after they accepted my paper despite this... Because STOC was much easier to get into in those days than now. Its standards have gone way up. But then when I started thinking about writing the final version, I had this idea of completeness, of complete problems. And of course where did the idea come from? It came from completeness for recursively enumerable sets, and in fact the – what is it? – the unsatisfiable predicate calculus formulas are complete for recursively enumerable problems. I knew that and my advisor was very interested in that, so I credit him for giving me the idea "Well, why can't we do this at a lower level for propositional formulas?" and then the analogue of recursively enumerable becomes nondeterministic polynomial time. And then I proved that the... Well, what I actually proved was that the valid propositional tautology, validity of a propositional tautology – which is in co-NP, it's not in NP – was complete for this class. But the reductions I put in that paper were not the many-one

reductions that Karp used and I use now, but they were Turing, polynomial-time Turing reductions, which are much more general reductions.

Yeah, so I didn't have the words "NP" and "P" – that was due to Dick Karp later – and I didn't have the same reduction, I had a more general reduction, and I had only three complete problems. And of course Dick Karp later, a year later had 21. I did have three in there. Of course there was the tautologies and subgraph isomorphism – given two graphs, is the first one isomorphic to a subgraph of the other? – and then three... Oh. And then, well, of course you could also instead of tautologies in general, you could look tautologies in disjunctive normal form, that was complete. And also, I did also have three DNFs. So conjunctive normal form with just three literals in a conjunct, that was also NP-complete, so that's what I had in my paper.

BK: And was there a conjecture in there about primality?

SC: I mentioned other possibilities. Yes, primality testing. I said that's a candidate maybe for completeness. I should look and see whether I was doubtful, because in fact there are randomized algorithms for primes. I may have said that, I don't remember. And the other, an open question was graph isomorphism. Neither of those are thought to be NP-complete now. In fact, primes are in poly time. So yeah, I did mention those. And of course graph isomorphism is definitely thought not to be NP-complete, although nobody's gotten a poly-time algorithm for it.

BK: You mentioned Richard Karp's paper following onto yours. How long did it take for researchers to sort of realize the significance of NP-completeness?

SC: Oh, I think it came very fast. I mean it was obvious, and Karp's paper was very well written. He had 21 examples of NP-complete problem. He cleaned up the terminology. He introduced "P" for poly-time for "NP" for nondeterministic polynomial time. That was new notation, very clean. He also introduced "many- one poly-time reducibility" whereas I had the more general kind of Turing reducibility, poly-time Turing reducibility. And all those were very clean, nice definitions. So that paper definitely caught on very rapidly.

BK: So in the 1970s, there was explosion of research. Were you surprised at the impact?

SC: Yeah, I would say I hadn't quite anticipated that there were so many NP-complete problems. You know, I think there were two kinds of people working in this field. There were the logicians and the people... algorithmic guy. And Karp was definitely an algorithmic guy. My training had all been

with the logicians, so I think that's my excuse for not realizing all of these examples of NP-completeness. And I should say on this term, a similar example for the notion of polynomial time of course was independently introduced by Jack Edmonds, about the same time as Alan Cobham wrote his paper. But they were from two different areas – Cobham was a logician and Jack was an algorithms guy. I mean they're two different fields, and so quite independently they came up with polynomial time. But I didn't know about Jack's stuff at all until much later. Cobham was my source for polynomial time.

BK: The question of whether P is equal to NP is one that's intrigued and frustrated a lot of researchers, so I have to ask you, what's your opinion on the status of P versus NP?

SC: Oh. Well, that's easy. I think P not equal to NP, and I think a majority of complexity theorists believe it. Well, so here's my tune on this. First of all, we're really good at finding algorithms for things. I mean there's a whole algorithms course we teach undergraduates in all these methods of finding algorithms, and lots and lots of examples. But for lower bounds, we aren't good at finding lower bounds. And here's my proof. If you look at the sequence of complexity-classes log-space, which is a subset of polynomial time P, which is a subset of NP, nondeterministic poly-time, which is a subset of polynomial space. So here we have a sequence of three inclusions starting from log-space and ending in polynomial time. There's an easy proof that log-space is a proper subset of polynomial space just by diagonalization. Therefore one of those intermediate three inclusions has to be proper. We can't prove any of them are proper. QED.

BK: But I guess even in 1967 now, you were telling us that your feeling was that P is not equal to NP.

SC: That's right. Based on attempts really seemed much harder to solve NP-hard problems, or NP problems in general. So yeah, I guess I conjectured that way back then. Yes.

BK: So not only is that problem P versus NP's central problem or the central problem in theoretically computer science, with the introduction of the Millennium Problems, Millennium by the Clay Foundation, the seven problems, it sort of has been acknowledged as one of the most open problems...

SC: Yeah, I guess one of the most interesting open prob-... important, important open problems, yes.

BK: So were you consulted about that inclusion, or did you just...?

SC: No, I was not consulted about the inclusion, but after they decided it, they did ask me to make a write-up for it. So I did contribute a write-up for that question, for the background.

BK: And I guess before that, Professor Smale had already... he had a list of 21 problems I think and included it.

SC: Oh, I guess he did. Yeah, that's right. He had already listed it as one of the... So definitely there's a consensus this is an important problem.

BK: What about the bigger significance? I mean it even gets into popular culture in *The Simpsons* or whatever. So how do you feel about...

SC: [laughs]

BK: ...I guess a little bit what's the real significance of the P versus NP problem?

SC: Well, I don't know. Yeah, you're right, it gets into... I guess everybody, many people know P and NP and they don't understand what it is. Well, obviously it's an important question. And, well, for one thing, if P equals NP, it's going to rule out a lot of cryptography. It's hard to imagine how we could have any of the cryptographic protocols like RSA and so on. Public key encryption seemed to be impossible. So that's on the one hand if it turns out P equals NP. And on the other hand, if P not equal to NP, of course you want to know more, you want to know just how hard is NP and so on. You can't help but learning a lot more about the problems. Either way. Either way it goes, P equals NP or P not equal to NP, we're going to learn a lot more about computation if the problem is solved.

BK: Right from the start of your research career, I guess what you'd call proof complexity has been a central focus. I think many computer scientists know about computational complexity, but maybe not so many know about proof complexity. So I'm wondering if you could describe a little bit what the concerns are and what some of the basic questions...

SC: Well, of course proof complexity in the propositional form is quite related to the P-NP question because you want to know... I mean a good aspect of proof complexity is you look at proof systems for proving tautologies, for example, which is equivalent to proving negations are unsatisfiable. So we have lots of standard proof systems for doing this.

Now the issue there is though "How long is the shortest proof? Can you get an upper bound on the length of the shortest proof in some proof system?" The conjecture there is there's no polynomial upper bound no matter what proof system, you know, efficient proof system. Under a reasonable definition of proof system, could you get a polynomial upper bound on the length of every tautology, a polynomial in the length of the tautology of

course? So we conjecture, no, that's sort of like "P not equal to NP" conjecture. I mean that sort of would imply P not equal to NP. But the... So you're... If there were such things, what I'm trying to say, if there were a proof system, efficient proof system to get poly-time proofs to all tautologies, then NP would equal co-NP, meaning that the complement of any NP problem would also be in NP. And again, there's lots and lots of examples, and we just don't think that's true. So we conjecture...
So this is all... proof complexity is certainly tied up with computational complexity. That's just one example.

BK: And you mentioned Professor Wang's program for proving the propositional tautologies in *Principia*, and these days both automated theorem proving or satisfiability solving in the propositional case has really become an important technology.

SC: Absolutely. Yeah, of course. And what was surprising is yeah, there are theorem provers especially for sets of clauses, for propositional problems that are in conjunctive normal form that are incredibly efficient. I mean they can find proofs or disproofs for even tens of thousands of clauses. So that's an interesting, very interesting, important subject in its own right. Of course, despite their great successes for some cases, you can always stump them...

BK: Of course.

SC: ...by coming up with hard examples. Like the pigeonhole tautologies is a good example.

BK: Besides the work in computational complexity, you've worked in other areas including parallel computing and theory of programming languages. Is there a theme that goes through all this work that holds it together?

SC: Well, it's all... I mean certainly the parallel computing part is an interesting... it's certainly mainstream, interesting complexity theory, because in practice now, computers have many, many processors and you're very interested in how much time you save if you have a whole bunch of processors. And there are conjectures of course... Well, for some problems you can't save much of any time. For other problems, you can save time hugely. So that's a very important and natural complexity question for parallel. What was the other...?

BK: The other one was theory of programming languages.

SC: Oh yeah. Well, that's Hoare. My contribution there was to Hoare logic.

BK: Right. And I think I've heard that referred to as "Cook's theorem of the relative completeness of Hoare's logic."

SC: [laughs]

BK: And I was quite surprised when I heard it referred to as "Cook's theorem."

SC: Well, apparently it's had some effect. I mean that's really not my field, programming langua-... It's a very important field, proving correctness of programs of course, and Hoare had this logic, a whole method of proving correctness. By putting in certain kinds of assertions here and there, you could use it to prove correctness of programs. So I succeeded that in a certain sense his logic was complete, so you can always do it if it's correct. So yeah, that was just one paper, but it seemed to have made a bit of a difference.

BK: Despite all the advances in computing technology – and here I have '71, but now I should say '67, which is when you originally formulated the problem – you know, the ideas about P and NP are still central questions in computer science. I'm wondering why they have such enduring significance.

SC: Well, I mean I guess – I tried to answer that before, right? – because they're obviously important, they're very important questions, and whichever way they resolve, we're going to learn a lot more about computation. And of course it's especially relevant at cryptographic protocols, which really would be much more difficult if it turns out to P equal NP.

BK: Apart from your research, you also have a lot of impact in terms of teaching and graduate supervision. I don't know the count now, but you've had over 30 PhD students.

SC: Yeah, it's 33 I think. 33, 34 graduate students. Some of them co-supervised I have to say. Especially later, in my more recent years, I have to credit other people like Toni Pitassi for being really good and helpful co-supervisors for students. So indeed, yes.

BK: And I'm wondering how you see how those three, teaching and supervision and research, interact with each other.

SC: Yeah. Well, there's no question supervising graduate students interacts with research, because your graduate students become co-authors in the papers. I mean it's extremely enlightening and important to have graduate students. I'm very grateful to my graduate students, including yourself because we have a joint paper. There's no question that working with graduate students is very enlightening and rewarding. And yeah, I also teach classes to undergraduates and, no, I enjoy that as long as I don't teach too many. Especially if I'm teaching in the areas I really like, which are complexity theory and logic. Those are my two favourite subjects, and I do enjoy teaching those courses because I think they're really neat and I try to get the students to agree.

BK: Well, I have to say from my personal experience that I think from the first time I walked into your office, you just treated me more as a colleague than a student, and I think it all was driven by real interest in the research questions.

SC: Absolutely, yeah. Well, you're definitely a helpful colleague indeed.

BK: So you're known to be an avid sailor. When did you first start sailing?

SC: Well, the first serious time I started sailing is when I went to Berkeley. I had tried to sail a few times. Even at Harvard, you sail. The sailing there on the river isn't too good, so I decided not to. But when I went to Berkeley, then you could stand up on the hill and see the San Francisco Bay opening up, the high road opening up for you, and all the boats out there really looked like fun. And I said, "Okay, I'm going to learn to sail." So I joined the student sailing club, UC- whatever it is. UCYC, University of California Yacht Club. But they let faculty members in. So then I joined and they had a very nice teaching program on Sundays. So you went to Sundays and you got in a boat and you had a teacher, and it was all very reasonably priced and they would teach you how to sail. So I really took to that right away and became a good sailor.

And the other thing is I met my wife, because she was secretary. She was an undergraduate. She was secretary of the club. So that was a great thing. Two good reasons why I learned to sail. But yes, I absolutely enjoyed it.

BK: You're a member of the Royal Canadian Yacht Club.

SC: I am. Yes, that's true. And so yeah, Linda and I are members of the Royal Canadian Yacht Club here. That was a recruiting tool by Tom Hull by the way to come to Toronto. He pointed out, "We do have a lake here. We don't have the ocean, but we do have Lake Ontario and they do sail." And he invited a member of the RCYC to lunch to explain how good it was. So indeed.

BK: And you raced for a long time.

SC: I still do.

BK: You still do?

SC: I still do. Yes, from the very beginning I took to sailboat racing, and I still do. Right now I sail... The club owns a fleet of... called Ideal 18s, two-person keelboats that are easy to sail. But they're all alike and we all go out and race. I have races Thursday evenings and sometimes on Saturdays. I still do that and I get a kick out of it.

BK: You mentioned your wife Linda. You still live with Linda in Toronto. And for a long time, she worked at University of Toronto, is that correct?

SC: Yeah. She worked at the admissions office at first and then at the registrar's office at University College. Then we started having children, and after a while we realized that it was tough, it was better if she quit. So she eventually quit and stayed home.

BK: And you have two sons, and I guess both of them have followed in your footsteps in their own way. I wonder if you could tell us a little bit more about them.

SC: Well, Gordon is the older one, born in 1978. And he took to sailing very early. Well, we encouraged him of course, Linda and I, in particular sailing in the Optimist dinghy, which is for children 15 and under, and they have a world-class racing... It's a world-class racing boat. Optimist dinghies, they're... What? They're eight feet long and not very big and have one sail, and they're for children. But they have international regattas. Anyway, Gordon took to that right away and practised and raced in the local races. Eventually he competed to represent Canada in the Optimist Worlds, and only five boats are allowed from each country. So he did that and he actually did that twice, and once it was in Greece and the second time it was in Spain, and I got to go and it was great.

Then he graduated. He got his engineering degree from Queen's University and started a little company making carbon-fibre stuff. But then he really liked sailing, so he decided to try out for the Olympics. So he worked very hard to... This is in the 49er class, is one of the Olympic-class racing boats, a 49er. It's about 14–18 feet long and very hot, very fast. So that was Gordon's choice, and he worked very, very hard and eventually competed for Canada in two different Olympics. In 2008 it was in China and 2012 it was in England.

BK: And James.

SC: And James. James also started out sailing, but he didn't take to it quite in the same way. But he was certainly interested in computers from age 3 on and also mathematics. So that was his thing. So he got his undergraduate degree at University of Toronto in math and computer science. Then he went on to Berkeley and got his PhD in computer science. Now he's working at Google in Silicon Valley and really enjoying it.

BK: I have one last question. What advice do you have for young researchers who are interested in starting a career in computer science or mathematics?

SC: Yeah, that's a really hard question. I guess my only comment there, as far as computer science goes, I think they need no encouragement because right now, as you know and I know, undergraduate institutions are being

flooded with people who want to take computer science. In our department, really it's even more than we can handle. We have to turn them away. So somehow computer science has become an enormously interesting field, and probably with some good reason. I mean I think the big driver now is artificial intelligence and machine learning. We're all seeing exciting things happening from that. And of course it's a good field to go into, no question, but there's going to be lots of competition.

BK: So I guess... I don't know if... I mean I've asked all my questions, but I don't know if there's anything else you want to add.

SC: Oh, I think you covered it pretty well.

BK: Good. Well, I really enjoyed talking to you.

SC: Oh, really enjoyed your questions, and very nice of you to come. How much time have you taken? I don't know.

BK: I'm not sure. Probably... It's 2:28, so it's probably been about an hour.

SC: Well, I hope they consider that to be enough.

BK: [laughs]

SC: Don't want to bore people too much.

[end of recording]

PART II

THE TURING AWARD LECTURE

The 1982 ACM Turing Award Lecture

Stephen Arthur Cook 1982 ACM Turing Award Recipient

The 1982 ACM Turing Award was presented to Stephen Arthur Cook, Professor of Computer Science at the University of Toronto, at the ACM Annual Conference in Dallas on October 25, 1982. The award is the Association's foremost recognition of technical contributions to the computing community.

The citation of Cook's achievements noted that "Dr. Cook has advanced our understanding of the complexity of computation in a significant and profound way. His seminal paper The Complexity of Theorem Proving Procedures presented at the 1971 ACM SIGACT Symposium on the Theory of Computing laid the foundations for the theory of NP-completeness. The ensuing exploration of the boundaries and nature of the NP-complete class of problems has been one of the most active and important research activities in computer science for the last decade.

Cook is well-known for his influential results in fundamental areas of computer science. He has made significant contributions to complexity theory, to time–space tradeoffs in computation, and to logics for programming languages. His work is characterized by elegance and insights and has illuminated the very nature of computation."

CR Categories and Subject Descriptors: F. O. [General]
General Term: Theory
Additional Key Words and Phrases: computational complexity
Originally published in Communications of the ACM June 1983 Volume 26 Number 6

During 1970-1979, Cook did extensive work under grants from the National Research Council. He was also an E.W.R. Staecie Memorial Fellowship recipient for 1977-1978. The author of numerous landmark papers, he is currently involved in proving that no "good" algorithm exists for NP-complete problems.

The ACM Turing Award memorializes A.M. Turing, the English mathematician who made major contributions to the computing sciences.

An Overview of Computational Complexity

Stephen A. Cook

Abstract

An historical overview of computational complexity is presented. Emphasis is on the fundamental issues of defining the intrinsic computational complexity of a problem and proving upper and lower bounds on the complexity of problems. Probabilistic and parallel computation are discussed.

This is the second Turing Award lecture on Computational Complexity. The first was given by Michael Rabin in 1976.[1] In reading Rabin's excellent article [Rab77] now, one of the things that strikes me is how much activity there has been in the field since. In this brief overview I want to mention what to me are the most important and interesting results since the subject began in about 1960. In such a large field the choice of topics is inevitably somewhat personal; however, I hope to include papers which, by any standards, are fundamental.

1 Early Papers

The prehistory of the subject goes back, appropriately, to Alan Turing. In his 1937 paper, *On computable numbers with an application to the Entscheidungsproblem* [Tur37], Turing introduced his famous Turing machine, which provided the most convincing formalization (up to that time) of the notion of an effectively (or algorithmically) computable function. Once this notion was pinned down precisely, impossibility proofs for computers were possible. In the same paper Turing proved

1. Michael Rabin and Dana Scott shared the Turing Award in 1976.

that no algorithm (i.e., Turing machine) could, upon being given an arbitrary formula of the predicate calculus, decide, in a finite number of steps, whether that formula was satisfiable.

After the theory explaining which problems can and cannot be solved by computer was well developed, it was natural to ask about the relative computational difficulty of computable functions. This is the subject matter of computational complexity. Rabin [Rab59, Rab60] was one of the first persons (1960) to address this general question explicitly: what does it mean to say that f is more difficult to compute than g? Rabin suggested an axiomatic framework that provided the basis for the abstract complexity theory developed by Blum [Blu67] and others.

A second early (1965) influential paper was *On the computational complexity of algorithms* by J. Hartmanis and R. E. Stearns [HS65].[2] This paper was widely read and gave the field its title. The important notion of complexity measure defined by the computation time on multitape Turing machines was introduced, and hierarchy theorems were proved. The paper also posed an intriguing question that is still open today. Is any irrational algebraic number (such as $\sqrt{2}$) computable in real time, that is, is there a Turing machine that prints out the decimal expansion of the number at the rate of one digit per 100 steps forever.

A third founding paper (1965) was *The intrinsic computational difficulty of functions* by Alan Cobham [Cob65]. Cobham emphasized the word "intrinsic," that is, he was interested in a machine-independent theory. He asked whether multiplication is harder than addition, and believed that the question could not be answered until the theory was properly developed. Cobham also defined and characterized the important class of functions he called $\mathscr{L}$: those functions on the natural numbers computable in time bounded by a polynomial in the decimal length of the input.

Three other papers that influenced the above authors as well as other complexity workers (including myself) are Yamada [Yam62], Bennett [Ben62], and Ritchie [Rit63]. It is interesting to note that Rabin, Stearns, Bennett, and Ritchie were all students at Princeton at roughly the same time.

2 Early Issues and Concepts

Several of the early authors were concerned with the question: What is the right complexity measure? Most mentioned computation time or space as obvious choices, but were not convinced that these were the only or the right ones. For example, Cobham [Cob65] suggested "... some measure related to the physical notion of work [may] lead to the most satisfactory analysis." Rabin [Rab60] introduced axioms which a complexity measure should satisfy. With the perspective of

2. See Hartmanis [Har81] for some interesting reminiscences.

20 years experience, I now think it is clear that time and space—especially time—are certainly among the most important complexity measures. It seems that the first figure of merit given to evaluate the efficiency of an algorithm is its running time. However, more recently it is becoming clear that parallel time and hardware size are important complexity measures too (see Section 6).

Another important complexity measure that goes back in some form at least to Shannon [Sha49] (1949) is Boolean circuit (or combinational) complexity. Here it is convenient to assume that the function f in question takes finite bit strings into finite bit strings, and the complexity $C(n)$ of f is the size of the smallest Boolean circuit that computes f for all inputs of length n. This very natural measure is closely related to computation time (see [Pip79, PF79, Sch76]), and has a well developed theory in its own right (see Savage [Sav76]).

Another question raised by Cobham [Cob65] is what constitutes a "step" in a computation. This amounts to asking what is the right computer model for measuring the computation time of an algorithm. Multitape Turing machines are commonly used in the literature, but they have artificial restrictions from the point of view of efficient implementation of algorithms. For example, there is no compelling reason why the storage media should be linear tapes. Why not planar arrays or trees? Why not allow a random access memory?

In fact, quite a few computer models have been proposed since 1960. Since real computers have random access memories, it seems natural to allow these in the model. But just how to do this becomes a tricky question. If the machine can store integers in one step some bound must be placed on their size. (If the number 2 is squared 100 times the result has 2^{100} bits, which could not be stored in all the world's existing storage media.) I proposed charged RAM's in [Coo72a], in which a cost (number of steps) of about $\log|x|$ is charged every time a number x is stored or retrieved. This works but is not completely convincing. A more popular random access model is the one used by Aho, Hopcroft, and Ullman in [AHU74], in which each operation involving an integer has unit cost, but intergers are not allowed to become unreasonably large (for example, their magnitude might be bounded by some fixed polynomial in the size of the input). Probably the most mathematically satisfying model is Schönhage's storage modification machine [Sch80a], which can be viewed either as a Turing machine that builds its own storage structure or as a unit cost RAM that can only copy, add or substract one, or store or retrieve in one step. Schönhage's machine is a slight generalization of the Kolmogorov–Uspenski machine proposed much earlier [KU58] (1958), and seems to me to represent the most general machine that could possibly be construed as doing a bounded amount of work in one step. The trouble is that it probably is a little too powerful. (See Section 3 under "large number mutiplication.")

Returning to Cobham's question "what is a step," I think what has become clear in the last 20 years is that there is no single clear answer. Fortunately, the competing computer models are not wildly different in computation time. In general, each can simulate any other by at most squaring the computation time (some of the first arguments to this effect are in [HS65]). Among the leading random access models, there is only a factor of log computation time in question.

This leads to the final important concept developed by 1965—the identification of the class of problems solvable in time bounded by a polynomial in the length of the input. The distinction between polynomial time and exponential time algorithms was made as early as 1953 by von Neumann [vNeu53]. However, the class was not defined formally and studied until Cobham [Cob65] introduced the class $\mathscr{L}$ of functions in 1964 (see Section 1). Cobham pointed out that the class was well defined, independent of which computer model was chosen, and gave it a characterization in the spirit of recursive function theory. The idea that polynomial time computability roughly corresponds to tractability was first expressed in print by Edmonds [Edm65a], who called polynomial time algorithms "good algorithms." The now standard notation P for the class of polynomial time recognizable sets of strings was introduced later by Karp [Kar72].

The identification of P with the tractable (or feasible) problems has been generally accepted in the field since the early 1970's. It is not immediately obvious why this should be true, since an algorithm whose running time is the polynomial n^{1000} is surely not feasible, and conversely, one whose running time is the exponential $2^{0.0001n}$ is feasible in practice. It seems to be an *empirical* fact, however, that naturally arising problems do not have optimal algorithms with such running times.[3] The most notable practical algorithm that has an exponential worst case running time is the simplex algorithm for linear programming. Smale [Sma82a, Sma82b] attempts to explain this by showing that, in some sense, the average running time is fast, but it is also important to note that Khachian [Kha79] showed that linear programming is in P using another algorithm. Thus, our general thesis, that P equals the feasible problems, is not violated.

3 Upper Bounds on Time

A good part of computer science research consists of designing and analyzing enormous numbers of efficient algorithms. The important algorithms (from the point of view of computational complexity) must be special in some way; they generally supply a surprisingly fast way of solving a simple or important problem. Below I

3. See [GJ79], pp. 6–9 for a discussion of this.

list some of the more interesting ones invented since 1960. (As an aside, it is interesting to speculate on what are the all time most important algorithms. Surely the arithmetic operations $+$, $-$, $*$, and $\div$ on decimal numbers are basic. After that, I suggest fast sorting and searching, Gaussian elimination, the Euclidean algorithm, and the simplex algorithm as candidates.)

The parameter n refers to the size of the input, and the time bounds are the worst case time bounds and apply to a multitape Turing machine (or any reasonable random access machine) except where noted.

(1) **The Fast Fourier Transform** [CT65], requiring $O(n\log n)$ arithmetic operations, is one of the most used algorithms in scientific computing.

(2) **Large number multiplication.** The elementary school method requires $O(n^2)$ bit operations to multiply two n digit numbers. In 1962 Karatsuba and Ofman [KO62] published a method requiring only $O(n^{1.59})$ steps. Shortly after that Toom [Too63] showed how to construct Boolean circuits of size $O(n^{1+\epsilon})$ for arbitrarily small $\epsilon > 0$ in order to carry out the mutiplication. I was a graduate student at Harvard at the time, and inspired by Cobham's question "Is mutiplication harder than addition?" I was naively trying to prove that multiplication requires $\Omega(n^2)$ steps on a multitape Turing machine. Toom's paper caused me considerable surprise. With the help of Stal Aanderaa [CA69], I was reduced to showing that multiplication requires $\Omega(n\log n/(\log\log n)^2)$ steps using an "on-line" Turing machine.[4] I also pointed out in my thesis that Toom's method can be adapted to multitape Turing machines in order to multiply in $O(n^{1+\epsilon})$ steps, something that I am sure came as no surprise to Toom.

The currently fastest asymptotic running time on a multitape Turing machine for number multiplication is $O(n\log n\ \log\log n)$, and was devised by Schönhage and Strassen [SS71] (1971) using the Fast Fourier Transform. However, Schönhage [Sch80a] recently showed by a complicated argument that his storage modification machines (see Section 2) can multiply in time $O(n)$ (linear time!). We are forced to conclude that either multiplication is easier than we thought or that Schönhage's machines cheat.

(3) **Matrix multiplication.** The obvious method requires $n^2(2n-1)$ arithmetic operations to multiply two $n \times n$ matrices, and attempts were made to prove the method optimal in the 1950's and 1960's. There was surprise when Strassen [Str69] (1969) published his method requiring only $4.7n^{2.81}$ operations. Considerable work has been devoted to reducing the exponent of 2.81,

4. This lower bound has been slightly improved. See [PFM74] and [RS82].

and currently the best time known is $O(n^{2.496})$ operations, due to Coppersmith and Winograd [CW82]. There is still plenty of room for progress, since the best known lower bound is $2n^2-1$ (see [BD78]).

(4) **Maximum matchings in general undirected graphs.** This was perhaps the first problem explicitly shown to be in P whose membership in P requires a difficult algorithm. Edmonds' influential paper [Edm65a] gave the result and discussed the notion of a polynomial time algorithm (see Section 2). He also pointed out that the simple notion of augmenting path, which suffices for the bipartite case, does not work for general undirected graphs.

(5) **Recognition of prime numbers.** The major question here is whether this problem is in P. In other words, is there an algorithm that always tells us whether an arbitrary n-digit input integer is prime, and halts in a number of steps bounded by a fixed polynomial in n? Gary Miller [Mil76] (1976) showed that there is such an algorithm, but its validity depends on the extended Riemann hypothesis. Solovay and Strassen [SS77] devised a fast Monte Carlo algorithm (see Section 5) for prime recognition, but if the input number is composite there is a small chance the algorithm will mistakenly say it is prime. The best provable deterministic algorithm known is due to Adleman, Pomerance, and Rumely [APR83] and runs in time $n^{O(\log\log n)}$, which is slightly worse than polynomial. A variation of this due to H. Cohen and H. W. Lenstra Jr. [CL82] can routinely handle numbers up to 100 decimal digits in approximately 45 seconds.

Recently three important problems have been shown to be in the class P. The first is linear programming, shown by Khachian [Kha79] in 1979 (see [PS82] for an exposition). The second is determining whether two graphs of degree at most d are isomorphic, shown by Luks [Luk80] in 1980. (The algorithm is polynomial in the number of vertices for fixed d, but exponential in d.) The third is factoring polynomials with rational coefficients. This was shown for polynomials in one variable by Lenstra, Lenstra, and Lovasz [LLL82] in 1982. It can be generalized to polynomials in any fixed number of variables as shown by Kaltofen's result [Kal82a, Kal82b].

4 Lower Bounds

The real challenge in complexity theory, and the problem that sets the theory apart from the analysis of algorithms, is proving lower bounds on the complexity of specific problems. There is something very satisfying in proving that a yes–no problem cannot be solved in n, or n^2, or 2^n steps, no matter what algorithm is used.

There have been some important successes in proving lower bounds, but the open questions are even more important and somewhat frustrating.

All important lower bounds on computation time or space are based on "diagonal arguments." Diagonal arguments were used by Turing and his contempories to prove certain problems are not algorithmically solvable. They were also used prior to 1960 to define hierarchies of computable 0–1 functions.[5] In 1960, Rabin [Rab60] proved that for any reasonable complexity measure, such as computation time or space (memory), sufficiently increasing the allowed time or space etc. always allows more 0–1 functions to be computed. About the same time, Ritchie in his thesis [Rit60] defined a specific hierarchy of functions (which he showed is nontrivial for 0–1 functions) in terms of the amount of space allowed. A little later Rabin's result was amplified in detail for time on multitape Turing machines by Hartmanis and Stearns [HS65], and for space by Stearns, Hartmanis, and Lewis [SHL65].

4.1 Natural Decidable Problems Proved Infeasible

The hierarchy results mentioned above gave lower bounds on the time and space needed to compute specific functions, but all such functions seemed to be "contrived." For example, it is easy to see that the function $f(x,y)$ which gives the first digit of the output of machine x on input y after $(|x|+|y|)^2$ steps cannot be computed in time $(|x|+|y|)^2$. It was not until 1972, when Albert Meyer and Larry Stockmeyer [MS72] proved that the equivalence problem for regular expressions with squaring requires exponential space and, therefore, exponential time, that a nontrivial lower bound for general models of computation on a "natural" problem was found (natural in the sense of being interesting, and not about computing machines). Shortly after that Meyer [Mey75] found a very strong lower bound on the time required to determine the truth of formulas in a certain formal decidable theory called WSIS (weak monadic second-order theory of successor). He proved that any computer whose running time was bounded by a fixed number of exponentials (2^n, 2^{2^n}, $2^{2^{2n}}$, etc.) could not correctly decide WSIS. Meyer's Ph.D. student, Stockmeyer went on to calculate [Sto74] that any Boolean circuit (think computer) that correctly decides the truth of an arbitrary WSIS formula of length 616 symbols must have more than 10^{123} gates. The number 10^{123} was chosen to be the number of protons that could fit in the known universe. This is a very convincing infeasibility proof!

Since Meyer and Stockmeyer there have been a large number of lower bounds on the complexity of decidable formal theories (see [FR79] and [Sto79] for summaries). One of the most interesting is a doubly exponential time lower bound on

5. See, for example, Grzegorczyk [Grz53].

the time required to decide Presburger arithmetic (the theory of the natural numbers under addition) by Fischer and Rabin [FR74]. This is not far from the best known time upper bound for this theory, which is triply exponential [Opp78]. The best space upper bound is doubly exponential [FR79].

Despite the above successes, the record for proving lower bounds on problems of smaller complexity is appalling. In fact, there is no nonlinear time lower bound known on a general purpose computation model for any natural problem in *NP* (See Section 4.4), in particular, for any of the 300 problems listed in [GJ79]. Of course, one can prove by diagonal arguments the existence of problems in *NP* requiring time n^k for any fixed k. In the case of space lower bounds, however, we do not even know how to prove the existence of *NP* problems not solvable in space $O(\log n)$ on an off-fine Turing machine (see Section 4.3). This is despite the fact that the best known space upper bounds in many natural cases are essentially linear in n.

4.2 Structured Lower Bounds

Although we have had little success in proving interesting lower bounds for concrete problems on general computer models, we do have interesting results for "structured" models. The term "structured" was introduced by Borodin [Bor82] to refer to computers restricted to certain operations appropriate to the problem at hand. A simple example of this is the problem of sorting n numbers. One can prove (see [Knu73]) without much difficulty that this requires at least $n\log n$ comparisons, provided that the only operation the computer is allowed to do with the inputs is to compare them in pairs. This lower bound says nothing about Turing machines or Boolean circuits, but it has been extended to unit cost random access machines, provided division is disallowed.

A second and very elegant structured lower bound, due to Strassen [Str73] (1973), states that polynomial interpolation, that is, finding the coefficients of the polynomial of degree $n-1$ that passes through n given points, requires $\Omega(n\log n)$ multiplications, provided only arithmetic operations are allowed. Part of the interest here is that Strassen's original proof depends on Bezout's theorem, a deep result in algebraic geometry. Very recently, Baur and Strassen [BS82] have extended the lower bound to show that even the middle coefficient of the interpolating polynomial through n points requires $\Omega(n\log n)$ multiplications to compute.

Part of the appeal of all of these structured results is that the lower bounds are close to the best known upper bounds,[6] and the best known algorithms can be implemented on the structured models to which the lower bounds apply. (Note

6. See Borodin and Munro [BM75] for upper bounds for interpolation.

that radix sort, which is sometimes said to be linear time, really requires at least $n\log n$ steps, if one assumes the input numbers have enough digits so that they all can be distinct).

4.3 Time–Space Product Lower Bounds

Another way around the impasse of proving time and space lower bounds is to prove time lower bounds under the assumption of small space. Cobham [Cob66] proved the first such result in 1966, when he showed that the time–space product for recognizing n-digit perfect squares on an "off-line" Turing machine must be $\Omega(n^2)$. (The same is true of n-symbol palindromes.) Here the input is written on a two-way read-only input tape, and the space used is by definition the number of squares scanned by the work tapes available to the Turing machine. Thus, if, for example, the space is restricted to $O(\log^3 n)$ (which is more than sufficient), then the time must be $\Omega(n^2/\log^3 n)$ steps.

The weakness in Cobham's result is that although the offline Turing machine model is a reasonable one for measuring computation time or space separately, it is too restrictive when time and space are considered together. For example, the palindromes can obviously be recognized in $2n$ steps and constant space if two heads are allowed to scan the input tape simultaneously. Borodin and I [BC82] partially rectified the weakness when we proved that sorting n integers in the range one to n^2 requires a time–space product of $\Omega(n^2/\log n)$. The proof applies to any "general sequential machine," which includes off-line Turing machines with many input heads, or even random access to the input tape. It is unfortunately crucial to our proof that sorting requires many output bits, and it remains an interesting open question whether a similar lower bound can be made to apply to a set recognition problem, such as recognizing whether all n input numbers are distinct. (Our lower bound on sorting has recently been slightly improved in [RS82].)

4.4 NP-Completeness

The theory of *NP*-completeness is surely the most significant development in computational complexity. I will not dwell on it here because it is now well known and is the subject of textbooks. In particular, the book by Garey and Johnson [GJ79] is an excellent place to read about it.

The class *NP* consists of all sets recognizable in polynomial time by a nondeterministic Turing machine. As far as I know, the first time a mathematically equivalent class was defined was by James Bennett in his 1962 Ph.D. thesis [Ben62]. Bennett used the name "extended positive rudimentary relations" for his class, and

his definition used logical quantifiers instead of computing machines. I read this part of his thesis and realized his class could be characterized as the now familiar definition of *NP*. I used the term $\mathscr{L}^+$ (after Cobham's class $\mathscr{L}$) in my 1971 paper [Coo71b], and Karp gave the now accepted name *NP* to the class in his 1972 paper [Kar72]. Meanwhile, quite independent of the formal development, Edmonds, back in 1965 [Edm65b], talked informally about problems with a "good characterization," a notion essentially equivalent to *NP*.

In 1971 [Coo71b], I introduced the notion of *NP*-complete and proved 3-satisfiability and the subgraph problem were *NP*-complete. A year later, Karp [Kar72] proved 21 problems were *NP*-complete, thus forcefully demonstrating the importance of the subject. Independently of this and slightly later, Leonid Levin [Lev73], in the Soviet Union (now at Boston University), defined a similar (and stronger) notion and proved six problems were complete in his sense. The informal notion of "search problem" was standard in the Soviet literature, and Levin called his problems "universal search problems."

The class *NP* includes an enormous number of practical problems that occur in business and industry (see [GJ79]). A proof that an *NP* problem is *NP*-complete is a proof that the problem is not in *P* (does not have a deterministic polynomial time algorithm) unless every *NP* problem is in *P*. Since the latter condition would revolutionize computer science, the practical effect of *NP*-completeness is a lower bound. This is why I have included this subject in the section on lower bounds.

4.5 #*P*-Completeness

The notion of *NP*-completeness applies to sets, and a proof that a set is *NP*-complete is usually interpreted as a proof that it is intractable. There are, however, a large number of apparently intractable *functions* for which no *NP*-completeness proof seems to be relevant. Leslie Valiant [Val79a, Val79b] defined the notion of #*P*-completeness to help remedy this situation. Proving that a function is #*P*-complete shows that it is apparently intractable to compute in the same way that proving a set is *NP*-complete shows that it is apparently intractable to recognize; namely, if a #*P*-complete function is computable in polynomial time, then $P = NP$.

Valiant gave many examples of #*P*-complete functions, but probably the most interesting one is the permanent of an integer matrix. The permanent has a definition formally similar to the determinant, but whereas the determinant is easy to compute by Gaussian elimination, the many attempts over the past hundred odd years to find a feasible way to compute the permanent have all failed. Valiant gave the first convincing reason for this failure when he proved the permanent #*P*-complete.

5 Probabilistic Algorithms

The use of random numbers to simulate or approximate random processes is very natural and is well established in computing practice. However, the idea that random inputs might be very useful in solving deterministic combinatorial problems has been much slower in penetrating the computer science community. Here I will restrict attention to probabilistic (coin tossing) polynomial time algorithms that "solve" (in a reasonable sense) a problem for which no deterministic polynomial time algorithm is known.

The first such algorithm seems to be the one by Berlekamp [Ber70] in 1970, for factoring a polynomial f over the field $GF(p)$ of p elements. Berlekamp's algorithm runs in time polynomial in the degree of f and $\log p$, and with probability at least one-half it finds a correct prime factorization of f; otherwise it ends in failure. Since the algorithm can be repeated any number of times and the failure events are all independent, the algorithm in practice always factors in a feasible amount of time.

A more dramatic example is the algorithm for prime recognition due to Solóvay and Strassen [SS77] (submitted in 1974). This algorithm runs in time polynomial in the length of the input m, and outputs either "prime" or "composite." If m is in fact prime, then the output is certainly "prime," but if m is composite, then with probability at most one-half the answer may also be "prime." The algorithm may be repeated any number of times on an input m with independent results. Thus if the answer is ever "composite," the user knows m is composite; if the answer is consistently "prime" after, say, 100 runs, then the user has good evidence that m is prime, since any fixed composite m would give such results with tiny probability (less than 2^{-100}).

Rabin [Rab76] developed a different probabilistic algorithm with properties similar to the one above, and found it to be very fast on computer trials. The number $2^{400}-593$ was identified as (probably) prime within a few minutes.

One interesting application of probabilistic prime testers was proposed by Rivest, Shamir, and Adleman [RSA78] in their landmark paper on public key cryptosystems in 1978. Their system requires the generation of large (100 digit) random primes. They proposed testing random 100 digit numbers using the Solovay–Strassen method until one was found that was probably prime in the sense outlined above. Actually with the new high powered deterministic prime tester of Cohen and Lenstra [CL82] mentioned in Section 3, once a random 100 digit "probably prime" number was found it could be tested for certain in about 45 seconds, if it is important to know for certain.

The class of sets with polynomial time probabilistic recognition algorithms in the sense of Solovay and Strassen is known as R (or sometimes RP) in the literature. Thus a set is in R if and only if it has a probabilistic recognition algorithm that

always halts in polynomial time and never makes a mistake for inputs not in R, and for each input in R it outputs the right answer for each run with probability at least one-half. Hence the set of composite numbers is in R, and in general $P \subseteq R \subseteq NP$. There are other interesting examples of sets in R not known to be in P. For example, Schwartz [Sch80b] shows that the set of nonsingular matrices whose entries are polynomials in many variables is in R. The algorithm evaluates the polynomials at random small integer values and computes the determinant of the result. (The determinant apparently cannot feasibly be computed directly because the polynomials computed would have exponentially many terms in general.)

It is an intriguing, open question whether $R = P$. It is tempting to conjecture yes on the philosophical grounds that random coin tosses should not be of much use when the answer being sought is a well defined yes or no. A related question is whether a probabilistic algorithm (showing a problem is in R) is for all practical purposes as good as a deterministic algorithm. After all, the probabilistic algorithms can be run using the pseudorandom number generators available on most computers, and an error probability of 2^{-100} is negligible. The catch is that pseudorandom number generators do not produce truly random numbers, and nobody knows how well they will work for a given probabilistic algorithm. In fact, experience shows they seem to work well. But if they *always* work well, then it follows that $R = P$, because pseudorandom numbers are generated deterministically so true randomness would not help after all. Another possibility is to use a physical process such as thermal noise to generate random numbers. But it is an open question in the philosophy of science how truly random nature can be.

Let me close this section by mentioning an interesting theorem of Adleman [Adl78] on the class R. It is easy to see [PF79] that if a set is in P, then for each n there is a Boolean circuit of size bounded by a fixed polynomial in n which determines whether an arbitrary string of length n is in the set. What Adleman proved is that the same is true for the class R. Thus, for example, for each n there is a small "computer circuit" that correctly and rapidly tests whether n digit numbers are prime. The catch is that the circuits are not uniform in n, and in fact for the case of 100 digits it may not be feasible to figure out how to build the circuit.[7]

6 Synchronous Parallel Computation

With the advent of VLSI technology in which one or more processors can be placed on a quarter-inch chip, it is natural to think of a future computer composed of many thousands of such processors working together in parallel to solve

7. For more theory on probabilistic computation, see Gill [Gil77].

a single problem. Although no very large general purpose machine of this kind has been built yet, there are such projects under way (see Schwartz [Sch80c]). This motivates the recent development of a very pleasing branch of computation complexity: the theory of large scale synchronous parallel computation, in which the number of processors is a resource bounded by a parameter $H(n)$ (H is for *hardware*) in the same way that space is bounded by a parameter $S(n)$ in sequential complexity theory. Typically $H(n)$ is a fixed polynomial in n.

Quite a number of parallel computation models have been proposed (see [Coo81a] for a review), just as there are many competing sequential models (see Section 2). There are two main contenders, however. The first is the class of shared memory models in which a large number of processors communicate via a random access memory that they hold in common. Many parallel algorithms have been published for such models, since real parallel machines may well be like this when they are built. However, for the mathematical theory these models are not very satisfactory because too much of their detailed specification is arbitrary: How are read and write conflicts in the common memory resolved? What basic operations are allowed for each processor? Should one charge $\log H(n)$ time units to access common memory?

Hence I prefer the cleaner model discussed by Borodin [Bor77] (1977), in which a parallel computer is a uniform family $\langle B_n \rangle$ of acyclic Boolean circuits, such that B_n has n inputs (and hence takes care of those input strings of length n). Then $H(n)$ (the amount of hardware) is simply the number of gates in B_n, and $T(n)$ (the parallel computation time) is the depth of the circuit B_n (i.e., length of the longest path from an input to an output). This model has the practical justification that presumably all real machines (including shared memory machines) are built from Boolean circuits. Furthermore, the minimum Boolean size and depth needed to compute a function is a natural mathematical problem and was considered well before the theory of parallel computation was around.

Fortunately for the theory, the minimum values of hardware $H(n)$ and parallel time $T(n)$ are not wildly different for the various competing parallel computer models. In particular, there is an interesting general fact true for all the models, first proved for a particular model by Pratt and Stockmeyer [PS76] in 1974 and called the "parallel computation thesis" in [Gol77]; namely, a problem can be solved in time polynomial in $T(n)$ by a parallel machine (with unlimited hardware) if and only if it can be solved in space polynomial in $T(n)$ by a sequential machine (with unlimited time).

A basic question in parallel computation is: Which problems can be solved substantially faster using many processors rather than one processor? Nicholas Pippenger [Pip79] formalized this question by defining the class (now called *NC*,

for "Nick's class") of problems solvable ultra fast [time $T(n) = (\log n)^{O(1)}$] on a parallel computer with a feasible [$H(n) = n^{O(1)}$] amount of hardware. Fortunately, the class *NC* remains the same, independent of the particular parallel computer model chosen, and it is easy to see that *NC* is a subset of the class *FP* of functions computable sequentially in polynomial time. Our informal question can then be formalized as follows: Which problems in *FP* are also in *NC*?

It is conceivable (though unlikely) that $NC = FP$, since to prove $NC \neq FP$ would require a breakthrough in complexity theory (see the end of Section 4.1). Since we do not know how to prove a function f in *FP* is not in *NC*, the next best thing is to prove that f is log space-complete for *FP*. This is the analog of proving a problem is *NP*-complete, and has the practical effect of discouraging efforts for finding super fast parallel algorithms for f. This is because if f is log space-complete for *FP* and f is in *NC*, then $FP = NC$, which would be a big surprise.

Quite a bit of progress has been made in classifying problems in *FP* as to whether they are in *NC* or log space-complete for *FP* (of course, they may be neither). The first example of a problem complete for *P* was presented in 1973 by me in [Coo74], although I did not state the result as a completeness result. Shortly after that Jones and Laaser [JL77] defined this notion of completeness and gave about five examples, including the emptiness problem for context-free grammars. Probably the simplest problem proved complete for *FP* is the so-called circuit value problem [Lad75b]: given a Boolean circuit together with values for its inputs, find the value of the output. The example most interesting to me, due to Goldschlager, Shaw, and Staples [GSS82], is finding the (parity of) the maximum flow through a given network with (large) positive integer capacities on its edges. The interest comes from the subtlety of the completeness proof. Finally, I should mention that linear programming is complete for *FP*. In this case the difficult part is showing that the problem is in *P* (see [Kha79]), after which the completeness proof [DLR79] is straightforward.

Among the problems known to be in *NC* are the four arithmetic operations ($+$, $-$, $*$, $\div$) on binary numbers, sorting, graph connectivity, matrix operations (multiplication, inverse, deteminant, rank), polynomial greatest common divisors, context-free languages, and finding a minimum spanning forest in a graph (see [BvGH82, Coo81a, Rei82, Ruz81]). The size of a maximum matching for a given graph is known [BvGH82] to be in "random" *NC* (*NC* in which coin tosses are allowed), although it is an interesting open question of whether finding an actual maximum matching is even in random *NC*. Results in [VS81] and [Ruz81] provide general methods for showing problems are in *NC*.

The most interesting problem in *FP* not known either to be complete for *FP* or in (random) *NC* is finding the greatest common divisor of two integers. There are many other interesting problems that have yet to be classified, including finding a maximum matching or a maximal clique in a graph (see [Val82]).

7 The Future

Let me say again that the field of computational complexity is large and this overview is brief. There are large parts of the subject that I have left out altogether or barely touched on. My apologies to the researchers in those parts.

One relatively new and exciting part, called "computational information theory," by Yao [Yao82], builds on Shannon's classical information theory by considering information that can be accessed through a feasible computation. This subject was sparked largely by the papers by Diffie and Hellman [DH76] and Rivest, Shamir, and Adleman [RSA78] on public key cryptosystems, although its computational roots go back to Kolmogoroff [Kol65] and Chaitin [Cha69, Cha75], who first gave meaning to the notion of a single finite sequence being "random," by using the theory of computation. An interesting idea in this theory, considered by Shamir [Sha81] and Blum and Micali [BM82], concerns generating pseudorandom sequences in which future bits are provably hard to predict in terms of past bits. Yao [Yao82] proves that the existence of such sequences would have positive implications about the deterministic complexity of the probabilistic class *R* (see Section 5). In fact, computational information theory promises to shed light on the role of randomness in computation.

In addition to computational information theory we can expect interesting new results on probabilistic algorithms, parallel computation, and (with any luck) lower bounds. Concerning lower bounds, the one breakthrough for which I see some hope in the near future is showing that not every problem in *P* is solvable in space $O(\log n)$, and perhaps also $P \neq NC$. In any case, the field of computational complexity remains very vigorous, and I look forward to seeing what the future will bring.

Acknowledgments

I am grateful to my complexity colleagues at Toronto for many helpful comments and suggestions, especially Allan Borodin, Joachim von zur Gathen, Silvio Micali, and Charles Rackoff.

References

[Adl78] L. Adleman. 1978. Two theorems on random polynomial time. In *Proceedings of 19th IEEE Symposium on Foundations of Computer Science*. IEEE Computer Society, Los Angeles, 75–83. DOI: https://doi.org/10.1109/SFCS.1978.37.

[APR83] L. Adleman, C. Pomerance, and R. S. Rumley. January. 1983. On distinguishing prime numbers from composite numbers. *Ann. Math.* 117, 173–206. DOI: https://doi.org/10.2307/2006975.

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

[Ben62] J. H. Bennett. 1962. *On Spectra*. Ph.D. thesis. Princeton University.

[Ber70] E. R. Berlekamp. 1970. Factoring polynomials over large finite fields. *Math. Comp.* 24, 713–735. DOI: https://doi.org/10.2307/2004849.

[Blu67] M. Blum. April. 1967. A machine-independent theory of the complexity of recursive functions. *J. ACM* 14, 2, 322–336. DOI: https://doi.org/10.1145/321386.321395.

[BM82] M. Blum and S. Micali. 1982. How to generate cryptographically strong sequences of pseudo random bits. In *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science*. IEEE Computer Society, Los Angeles, 112–117. DOI: https://doi.org/10.1109/SFCS.1982.72.

[Bor77] A. Borodin. 1977. On relating time and space to size and depth. *SIAM J. Comput.* 6, 4, 733–744. DOI: https://doi.org/10.1137/0206054.

[Bor82] A. Borodin. 1982. Structured vs. general models in computational complexity. In *Logic and Algorithmic*, Monographie no. 30 de L'Enseignement Mathematique Université de Genève.

[BC82] A. Borodin and S. A. Cook. 1982. A time–space tradeoff for sorting on a general sequential model of computation. *SIAM J. Comput.* 11, 2, 287–297. DOI: https://doi.org/10.1137/0211022.

[BvGH82] A. Borodin, J. von zur Gathen, and J. Hopcroft. 1982. Fast parallel matrix and GCD computations. In *23rd IEEE Symposium on Foundations of Computer Science.* IEEE Computer Society, Los Angeles, 65–71. DOI: https://doi.org/10.1109/SFCS.1982.17.

[BM75] A. Borodin and I. Munro. 1975. *The Computational Complexity of Algebraic and Numeric Problems*. Elsevier, New York.

[BD78] R. W. Brockett and D. Dobkin. 1978. On the optimal evaluation of a set of bilinear forms. *Linear Algebra Appl.* 19, 207–235. DOI: https://doi.org/10.1016/0024-3795(78)90012-5.

[Cha69] G. J. Chaitin. January. 1969. On the length of programs for computing finite binary sequences. *J. ACM* 13, 4, 547–569; *J. ACM* 16, 1, 145–159 (October 1966). DOI: https://doi.org/10.1145/321356.321363.

[Cha75] G. J. Chaitin. July. 1975. A theory of program size formally identical to informational theory. *J. ACM* 22, 3, 329–340. DOI: https://doi.org/10.1145/321892.321894.

[Cob65] A. Cobham. 1965. The intrinsic computational difficulty of functions. In *Proceedings of the 1964 International Congress for Logic, Methodology, and Philosophy of Science*. North-Holland, Amsterdam, 24–30.

[Cob66] A. Cobham. 1966. *The Recognition Problem for the Set of Perfect Squares*. Conference Record, IEEE 7th Annual Symposium on Switching and Automata Theory, 78–87.

[CL82] H. Cohen and H. W. Lenstra Jr. 1982. Primarily testing and Jacobi sums. Report 82–18, University of Amsterdam, Dept. of Math.

[Coo71b] S. A. Cook. 1971b. Characterizations of pushdown machines in terms of time-bounded computers. *J. ACM* 18, 1, 4–18. DOI: https://doi.org/10.1145/321623.321625.

[Coo72a] S. A. Cook. 1972a. Linear time simulation of deterministic two-way pushdown automata. In *Information Processing 71 (Proc. IFIP Congress, Ljubljana, 1971).* Vol. 1: Foundations and Systems. North-Holland, Amsterdam, 75–80.

[Coo74] S. A. Cook. 1974. An observation on time–storage trade off. *J. Comput. Syst. Sci.* 9, 3, 308–316. DOI: https://doi.org/10.1016/S0022-0000(74)80046-2.

[Coo81] S. A. Cook. 1981. Towards a complexity theory of synchronous parallel computation. *Enseign. Math.* 2, 27, 1–2, 99–124. ISSN 0013-8584.

[CA69] S. A. Cook and S. O. Aanderaa. 1969. On the minimum computation time of functions. *Trans. Am. Math. Soc.* 142, 291–314. ISSN 0002-9947. DOI: https://doi.org/10.2307/1995359.

[CT65] J. M. Cooley and J. W. Tukey. 1965. An algorithm for the machine calculation of complex Fourier series. *Math. Comput.* 19, 297–301. DOI: https://doi.org/10.1090/S0025-5718-1965-0178586-1.

[CW82] D. Coppersmith and S. Winograd. 1982. On the asymptomatic complexity of matrix multiplication. *SIAM J. Comp*. 11, 472–492. DOI: https://doi.org/10.1109/SFCS.1981.27.

[DH76] W. Diffie and M. E. Hellman. 1976. New directions in cryptography. *IEEE Trans. Inform. Theory IT* 22, 6, 644–654. DOI: https://doi.org/10.1109/TIT.1976.1055638.

[DLR79] D. Dobkin, R. J. Lipton, and S. Reiss. 1979. Linear programming is log-space hard for P. *Inf. Process. Lett.* 8, 96–97. DOI: https://doi.org/10.1016/0020-0190(79)90152-2.

[Edm65a] J. Edmonds. 1965a. Paths, trees, and flowers. *Can. J. Math.* 17, 449–467. DOI: https://doi.org/10.4153/CJM-1965-045-4.

[Edm65b] J. Edmonds. 1965b. Minimum partition of a matroid into independent subsets. *J. Res. Natl. Bur. Stand. Sect. B* 69, 67–72. DOI: https://doi.org/10.6028/JRES.069B.004.

[FR79] J. Ferrante and C. W. Rackoff. 1979. The Computational Complexity of Logical Theories. *Lecture Notes in Mathematics*. #718, Springer Verlag, New York.

[FR74] M. J. Fischer and M. O. Rabin. 1974. Super-exponential complexity of Presburger arithmetic. In R. Karp (Ed.), *Complexity of Computation. SIAM–AMS Proc.* 7, 27–42.

[GJ79] M. R. Garey and D. S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman. ISBN 0-7167-1044-7.

[Gil77] J. Gill. 1977. Computational complexity of probabilistic Turing machines. *SIAM J. Comput.* 6, 675–695. DOI: https://doi.org/10.1137/0206049.

[Gol77] L. M. Goldschlager. 1977. *Synchronous Parallel Computation*. Ph.D. Dissertation. University of Toronto. Also, Technical Report 114, Department of Computer Science, University of Toronto.

[GSS82] L. M. Goldschlager, R. A. Shaw, and J. Staples. 1982. The maximum flow problem is log space complete for P. *Theor. Comput. Sci.* 21, 1, 105–111. DOI: https://doi.org/10.1016/0304-3975(82)90092-5.

[Grz53] A. Grzegorczyk. 1953. Some classes of recursive functions. *Rozprawy Mat.* 4, 46. ISSN 0860-2581.

[Har81] J. Hartmanis. January. 1981. Observations about the development of theoretical computer science. *Annals Hist. Comput.* 3, 1, 42–51. DOI: https://doi.org/10.1109/MAHC.1981.10005.

[Har65] J. Hartmanis and R. E. Stearns. 1965. On the computational complexity of algorithms. *Trans. AMS* 117, 285–306. DOI: https://doi.org/10.2307/1994208.

[JL77] N. D. Jones and W. T. Laaser. 1977. Complete problems for deterministic polynomial time. *Theor. Comput. Sci.* 3, 105–117.

[Kal82a] E. Kaltofen. May 5–7. 1982a. A polynomial reduction from multivariate to bivariate integer polynomial factorization. In *Proceedings of the 14th ACM Symposium in Theory Computing*. San Francisco, CA, 261–266.

[Kal82b] E. Kaltofen. 1982b. A polynomial-time reduction from bivariate to univariate integral polynomial factorization. In *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science.* IEEE Computer Society, Los Angeles, 57–64. DOI: https://doi.org/10.1109/SFCS.1982.56.

[KO62] A. Karatsuba and Y. Ofman. 1962. Multiplication of multidigit numbers on automata. *Doklady Akad. Nauk* 145, 2, 293–294. Translated in *Soviet Phys. Doklady.* 7, 7 (1963), 595–596.

[Kar72] R. M. Karp. 1972. Reducibility among combinatorial problems. In *Proceedings of a Symposium on the Complexity of Computer Computations*, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York. Springer, 85–103. DOI: https://doi.org/10.1007/978-1-4684-2001-2_9.

[Kha79] L. G. Khachian. 1979. A polynomial time algorithm for linear programming. *Doklady Akad. Nauk SSSR*. 244, 5, 1093–1096. Translated in *Soviet Math. Doklady*. 20, 191–194.

[Knu73] Knuth, D. E. 1973. *The Art of Computer Programming,* Vol. 3 Sorting and Searching. Addison-Wesley, Reading, MA.

[Kol65] A. N. Kolmogorov. 1965. Three approaches to the concept of the amount of information. *Probl. Pered. Inf. (Probl. of Inf. Transm.)* 1.

[KU58] A. N. Kolmogorov and V. A. Uspenski. 1958. On the definition of an algorithm, *Uspehi Mat. Nauk.* 13, 4(82), 3–28. ISSN 0042-1316. AMS Transl. 2nd ser. 29 (1963), 217–245.

[Lad75] R. E. Ladner. 1975. The circuit value problem is log space complete for P. *SIGACT News* 7, 1, 18–20. DOI: https://doi.org/10.1145/990518.990519.

[LLL82] A. K. Lenstra, H. W. Lenstra, and L. Lovasz. 1982. *Factoring Polynomials with Rational Coefficients*. Report 82–05. University of Amsterdam, Dept. of Math.

[Lev73] L. A. Levin. 1973. Universal sequential search problems. *Probl. Inf. Transm.* 9, 3, 115–116. [Universal'nye perebornye zadachi. *Problemy Peredachi Informatsii* 9, 3, 115–116, 1973].

[Luk80] E. M. Luks. 1980. Isomorphism of graphs of bounded valence can be tested in polynomial time. In *Proceedings of the 21st IEEE Symposium on Foundations of Computer Science.* IEEE Computer Society, Los Angeles, 42–49. DOI: https://doi.org/10.1109/SFCS.1980.24.

[Mey75] A. R. Meyer. 1975. *Weak Monadic Second-Order Theory of Successor is not Elementary-Recursive*. Lecture Notes in Mathematics, Vol. 453. Springer Verlag, New York, 132–154. DOI: https://doi.org/10.1007/BFb0064872.

[MS72] A. R. Meyer and L. J. Stockmeyer. 1972. The equivalence problem for regular expressions with squaring requires exponential space. In *Proceedings of the 13th IEEE Symposium on Switching and Automata Theory.* 125–129. DOI: https://doi.org/10.1109/SWAT.1972.29.

[Mil76] G. L. Miller. 1976. Riemann's Hypothesis and tests for primality. *J. Comput. System Sci.* 13, 300–317. DOI: https://doi.org/10.1016/S0022-0000(76)80043-8.

[vNeu53] J. von Neumann. 1953. A certain zero-sum two-person game equivalent to the optimal assignment problem. In H. W. Kuhn and A. W. Tucker (Eds.), *Contributions to the Theory of Games*, Vol. II, Princeton University Press, 5–12. DOI: https://doi.org/10.1515/9781400881970-002.

[Opp78] D. C. Oppen. 1978. A $2^{2^{2^{pn}}}$ upper bound on the complexity of Presburger arithmetic. *J. Comput. Syst. Sci.* 16, 323–332. DOI: https://doi.org/10.1016/0022-0000(78)90021-1.

[PS82] C. H. Papadimitriou and K. Steiglitz. 1982. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Englewood Cliffs, NJ.

[PFM74] M. S. Paterson, M. J. Fischer, and A. R. Meyer. 1974. *An Improved Overlap Argument for On-line Multiplication*. *SIAM–AMS Proc. 7*, Amer. Math. Soc., Providence, 97–111.

[Pip79] N. Pippenger. October 29–31. 1979. On simultaneous resource bounds (preliminary version). In *20th Annual Symposium on Foundations of Computer Science*. IEEE, San Juan, Puerto Rico, 307–311. DOI: https://doi.org/10.1109/SFCS.1979.29.

[PF79] N. J. Pippenger and M. J. Fischer. April. 1979. Relations among complexity measures. *J. Assoc. Comput. Mach.* 26, 2, 361–381. DOI: https://doi.org/10.1145/322123.322138.

[PS76] V. R. Pratt and L. J. Stockmeyer. 1976. A characterization of the power of vector machines. *J. Comput. Syst. Sci.* 12, 2, 198–221. DOI: https://doi.org/10.1016/S0022-0000(76)80037-2.

[Rab59] M. O. Rabin. 1959. Speed of computation and classification of recursive sets. In *Third Convention Science Society*. Israel, 1–2.

[Rab60] M. O. Rabin. 1960. Degree of difficulty of computing a function and a partial ordering of recursive sets. Tech. Rep. No. 1, O.N.R., Jerusalem.

[Rab76] M. O. Rabin. 1976. Probabilistic algorithms. In J. F. Traub (Ed.), *Algorithms and Complexity, New Directions and Recent Trends*. Academic Press, New York, 21–39.

[Rab77] M. O. Rabin. September. 1977. Complexity of computations. *Comm. ACM* 20, 9, 625–633. DOI: https://doi.org/10.1145/359810.359816.

[Rei82] J. H. Reif. May 5–7. 1982. Symmetric complementation. *Proceedings of the 14th ACM Symposium on Theory of Computing*. San Francisco, CA, 201–214. DOI: https://doi.org/10.1145/800070.802193.

[RS82] S. Reisch and G. Schnitger. 1982. Three applications of Kolmorgorov complexity. *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science.* IEEE Computer Society, Los Angeles. 45–52.

[Rit60] R. W. Ritchie. 1960. *Classes of Recursive Functions of Predictable Complexity*. Doctoral Dissertation, Princeton University.

[Rit63] R. W. Ritchie. 1963. Classes of predictably computable functions. *Trans. Am. Math. Soc.* 106, 139–173. DOI: https://doi.org/10.2307/1993719.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. February. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Comm. ACM* 21, 2, 120–126. DOI: https://doi.org/10.1145/359340.359342.

[Ruz81] W. L. Ruzzo. 1981. On uniform circuit complexity. *J. Comput. System Sci.* 22, 3, 365–383. DOI: https://doi.org/10.1016/0022-0000(81)90038-6.

[Sav76] J. E. Savage. 1976. *The Complexity of Computing.* Wiley, New York.

[Sch76] C. P. Schnorr 1976. The network complexity and the Turing machine complexity of finite functions. *Acta Informatica* 7, 95–107. DOI: https://doi.org/10.1007/BF00265223.

[Sch80a] A. Schönhage. 1980a. Storage modification machines. *SIAM J. Comput.* 9, 3, 490–508. DOI: https://doi.org/10.1137/0209036.

[SS71] A. Schönhage and V. Strassen. 1971. Schnelle Multiplication grosser Zahlen. *Computing* 7, 281–292. DOI: https://doi.org/10.1007/BF02242355.

[Sch80b] J. T. Schwartz. October. 1980b. Probabilistic algorithms for verification of polynomial identifies. *J. ACM* 27, 4, 701–717.

[Sch80c] J. T. Schwartz. October. 1980c. Ultracomputers. *ACM Trans. on Prog. Languages and Systems* 2, 4, 484–521. DOI: https://doi.org/10.1145/357114.357116.

[Sha81] A. Shamir. July. 1981. On the generation of cryptographically strong pseudo random sequences. In *Proceedings of the 8th Int. Colloquium on Automata, Languages, and Programming*. Lecture Notes in Computer Science. Vol. 115, Springer, Verlag, New York, 544–550. DOI: https://doi.org/10.1007/3-540-10843-2_43.

[Sha49] C. E. Shannon. 1949. The synthesis of two-terminal switching circuits. *BSTJ* 28, 59–98. DOI: https://doi.org/10.1002/j.1538-7305.1949.tb03624.x.

[Sma82a] S. Smale. 1982a. On the average speed of the simplex method of linear programming. Preprint.

[Sma82b] S. Smale. 1982b. The problem of the average speed of the simplex method. Preprint.

[SS77] R. Solovay and V. Strassen. 1977. A fast monte-carlo test for primality. *SIAM J. Comput*. 6, 84–85. DOI: https://doi.org/10.1137/0206006.

[SHL65] R. E. Stearns, J. Hartmanis, and P. M. Lewis. October 6–8. 1965. Hierarchies of memory limited computations. In *Proceedings of the 6th Annual Symposium on Switching Circuit Theory and Logical Design*. IEEE, Ann Arbor, Michigan, 179–190. DOI: https://doi.org/10.1109/FOCS.1965.11.

[Sto74] L. J. Stockmeyer. 1974. *The complexity of decision problems in automata theory and logic*. Doctoral Thesis. Dept. of Electrical Eng., Report TR-133, MIT Laboratory for Computer Science, MIT, Cambridge, MA.

[Sto79] L. J. Stockmeyer. 1979. Classifying the computational complexity of problems. Research Report RC 7606, Math. Sciences Dept., IBM T. J. Watson Research Center, Yorktown Heights, NY.

[Str69] V. Strassen. 1969. Gaussian elimination is not optimal. *Numer. Math.* 13, 4, 354–356. DOI: https://doi.org/10.1007/BF02165411.

[Str73] V. Strassen. 1973. Die Berechnungskomplexität von elementarsymmetrischen Funktionen und von Interpolationskoeffizienten. *Numer. Math.* 20, 238–251. DOI: https://doi.org/10.1007/BF01436566.

[BS82] W. Baur and V. Strassen. 1982. The complexity of partial derivatives. Preprint.

[Too63] A. L. Toom. 1963. The complexity of a scheme of functional elements realizing the multiplication of integers. *Sov. Math. Dokl.* 3, 4, 714–716.

[Tur37] A. M. Turing. 1937. On computable numbers with an application to the Entscheidnungsproblem. *Proc. London Math. Soc. Ser.* 2, 42 (1936–7), 230–265. A correction. *ibid*. 43, 544–546.

[Val79a] L. G. Valiant. 1979a. The complexity of enumeration and reliability problems. *SIAM J. Comput.* 8, 410–421. DOI: https://doi.org/10.1137/0208032.

[Val79b] L. G. Valiant. 1979b. The complexity of computing the permanent. *Theor. Comput. Sci.* 8, 189–202. DOI: https://doi.org/10.1016/0304-3975(79)90044-6.

[Val82] L. G. Valiant. 1982. Parallel Computation. In *Proceedings of the 7th IBM Japan Symposium*. Academic 6 Scientific Programs, IBM Japan, Tokyo.

[VS81] L. G. Valiant, S. Skyum. 1981. Fast Parallel Computation Of Polynomials Using Few Processors. Lecture Notes in Computer Science. Vol. 118, Springer, Berlin, Heidelberg, 132–139. DOI: https://doi.org/10.1007/3-540-10856-4_79.

[Yam62] H. Yamada. 1962. Real time computation and recursive functions not real-time computable. *IRE Trans. Electron Comput.* EC-11, 753–760. DOI: https://doi.org/10.1109/TEC.1962.5219459.

[Yao82] A. C. Yao. 1982. Theory and application of trapdoor functions (Extended abstract). In *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science.* IEEE Computer Society, Los Angeles, 80–91. DOI: https://doi.org/10.1109/SFCS.1982.45.

PART II

PERSPECTIVES ON COOK'S WORK

4 Cook's NP-completeness Paper and the Dawn of the New Theory

Christos H. Papadimitriou

4.1 History

In May 1971, Stephen A. Cook, by then at the University of Toronto after four years at UC Berkeley Mathematics, presented at the 3rd Symposium of the Theory of Computing, held at Shaker Heights, Ohio, a paper entitled "The complexity of theorem-proving procedures" [Coo71b]. In this now famous work, he points out that any decision problem solved by a polynomial-time nondeterministic Turing machine can be reduced, in polynomial time, to the problem of verifying Boolean tautologies, and thus the latter problem is the least likely to be solved in polynomial time among all such problems. "SAT is NP-complete," we would have said in today's terminology of our field—a field and a terminology that have been transformed dramatically and profoundly by this paper. In the last section of his paper, Cook goes on to define a framework for the complexity of proofs in first-order logic, and prove upper and lower bounds in it.

To understand where this paper came from, one needs to look seven decades back. In 1900, David Hilbert declared war on all open problems in mathematics, and mathematicians all over the world (actually, mostly Europe) resolved to dissect and automate mathematical truth. An explosion in mathematical logic resulted, culminating three decades later in Kurt Gödel's Incompleteness Theorem, which established that Hilbert's dream is unattainable. Spurred by that momentous result, mathematicians focused on extinguishing the last remaining hope for the automation of truth project: One could still hope that perhaps theorems can be proved mechanically, *as long as* they can be proved at all. This is

precisely Hilbert's *Entscheidungsproblem* featured in the title of Turing's paper—as we shall see, a topic actually revisited in Section 4.3 of Cook's paper. But the impossibility of the Entscheidungsproblem cannot be proved through logic alone, like Gödel's theorem. This is how logicians were finally forced to leave logic behind and define computation. And define they did: Alonzo Church, Emil Post, Stephen Kleene (working from his notes for Gödel's seminar, as turmoil in Europe had by then shifted much of the activity to New Jersey), A. A. Markov in the Soviet Union, and of course Alan Turing; all of these very different conceptions of computation were soon to be proved equivalent. Turing's entry, however, was particularly compelling and influential as his definition was vividly physical and visually engaging, and his paper contained a pitch for universality and software. Importantly, it boasted the paradigmatic proof that the halting problem is undecidable: Computer science is the only known field of scientific discourse that was born aware of its own limitations.

War interrupted this intellectual pursuit but intensified the race for actual computing machines. It was immediately after the end of the war that John von Neumann, the protean and overpowering mathematician who was the first to absorb Gödel's proof and Turing's ideas, consolidated, focused, and advanced the era's thinking about computers with his "First draft of a report on the EDVAC" and the ensuing creation of the first computers.

In the years after that, a new mathematical field started crystalizing around the now extant computers, seeking algorithms for their applications but also mathematical foundations and guidance for creating better hardware and software—especially compilers, the towering engineering problem of early computer science, soon to be joined by the design of operating systems, databases, chips, networks. During the 1950s and 1960s, and for a couple of decades to follow, developing the foundations of hardware and software was the job of computer science theorists—as opposed to the theoretically inclined researchers in these applied fields, as is the norm today. As the new field was looking around for inspiration in the late 1950s, another intellectual giant of that era, Noam Chomsky, articulated his hierarchy of languages—hinting at a parallel one of computation—and computer scientists were impressed and inspired. Automata and language theory, so pertinent to computation and especially to the compilers project, became choice subjects for theoreticians. To understand the extent to which this had happened, just look at the titles of the 14 papers surrounding Cook's in the proceedings of 1971 STOC:

- "Some results in tree automata"
- "Block structure: retention or deletion?"
- "On the parallel computation of local operations"

- "An iteration theorem in one-counter languages"
- "Intersection-closed full AFL[1] and the recursively enumerable languages"
- "Absolutely parallel grammars and deterministic finite-state transducers"
- "Addressable data graphs"
- *Cook's paper*
- "The care and feeding of LR(k) grammars" (by 2020 Turing award winners Alfred Aho and Jeffrey Ullman)
- "Domolki's algorithm applied to generalized overlap resolvable grammars"
- "An algorithm generating the decision table of a deterministic bottom-up parser of a subset of context-free languages"
- "A decision procedure for generalized mapability-onto of regular sets"
- "Complexity of formal translations and speed-up results"
- "Classification of computable functions by primitive recursive classes"
- "Complexity classes of partial recursive functions"

Theoreticians were already working on complexity, as can be seen in these last titles, albeit on an early version inspired and heavily influenced by Kleene's recursive function theory. That this theory can be simplified and sharpened tremendously if one starts from the basis of polynomial-time computation was not yet part of the field's culture.

Polynomial time had already appeared in the field's horizon a number of times. In 1953, John von Neumann had bragged about an algorithm of his being polynomial, compared to the exponential incumbent [vNeu53]; Alan Cobham, a logician working for IBM, had defined in 1964 the class of polynomial-time computable functions and called it $\mathcal{L}$ [Cob65]—that is why Cook, in his 1971 paper, denotes what we now know as P by $\mathcal{L}_*$, the decision variant of $\mathcal{L}$. In 1965–1967, Jack Edmonds had informally defined P and NP and conjectured that they are different [Edm67]. Finally—even though few people, if any, besides Gödel knew this in the year 1971 [Har93]—Gödel had written 15 years earlier a letter to von Neumann (who was at that point terminally sick) pondering whether a problem in logic that today can be seen as an NP-complete generalization of satisfiability—namely, deciding whether there is a proof of length n of a given theorem in a given axiomatic system of first-order logic—can be solved in time polynomial in n (Gödel himself appears in the letter to be surprisingly open to this possibility).

1. Abstract families of languages.

4.2 Cook's Other 1971 Paper

What are the origins of Cook's apparent interest in, and familiarity with, polynomial time? Cook was trained in logic (his Ph.D. advisor at Harvard was Hao Wang, with whom he had written a paper in set theory [CW66]) but was also interested in forms of complexity beyond recursion theory—witness his Ph.D. thesis on the ways in which integer multiplication is harder than addition [CA69]. During his time at Berkeley during the late 1960s, he taught a graduate course on complexity, which was squarely within the dominant paradigm: the title of his notes for this class, included in this volume, is "A survey of classes of primitive recursion functions." In these notes, however, Cook dwells upon the lower expanses of this domain, especially on classes of primitive recursive functions that correspond to Turing machine space and time bounds, in particular, polynomial and linear bounds, respectively. This includes Cobham's class as well as Bennett's classes of extended positive rudimentary relations and functions [Ben62]. Significantly, Cook observes that these classes are exactly those Turing machine computable in nondeterministic polynomial time. Connections to classical computability also play a role in his NP-completeness paper, where Cook defines the concept of *polynomial degrees* ("degree" being a common term in recursive function theory), precisely the step needed for connecting recursion-inspired complexity with the realm of polynomial-time computation.

Like most theoreticians in the 1960s, Cook had also been active in automata theory, albeit with a forward-looking twist: during the late 1960s, he published a series of papers relating pushdown automata of various grades with Turing machine computation [Coo69, Coo70].

Four months before his talk at Shaker Heights, Cook published the most mature of these works in the *Journal of the ACM* [Coo71a]. He endows the pushdown automaton with a $\log n$-bounded Turing machine tape and shows that the resulting device *is tantamount to polynomial-time computation!* This follows from the fact that, with the help of a pushdown store, logarithmic space, both deterministic and nondeterministic, is equivalent to polynomial *deterministic* time—in fact, this is where Cook first introduced the notation $\mathcal{L}_*$. The results are stated for an arbitrary space bound no smaller than $\log n$, but there is clear if implicit emphasis on $\log n$ space and polynomial time. That the computation of a pushdown automaton can be fathomed by an algorithm that analyses its configurations had been known, and a logarithmic tape makes this a polynomial-time deterministic algorithm. To prove the inverse direction, that polynomial deterministic time can be simulated by a logarithmic tape and a pushdown machine, Cook (1) standardizes the polynomial-time Turing machine so that the square inspected at time t

is predictable, a familiar maneuver known as obliviousness, and (2) shows how the deterministic augmented pushdown device can explore the polynomial-time machine's reachable configurations until an accepting one is found. The last part of this paper points out that this method resolves several open problems in the theory of pushdown and stack automata. Cook concludes by pondering, I believe for the first time, whether logarithmic space can be the same as P.

This is a remarkable paper for at least three reasons besides its stunning originality and power: It connects the core of automata theory, a subject that had arguably become the dead-end research direction of theory, with the fresh, dawning paradigm of polynomial-time computation. Second, it takes some care to see why this result *does not* provide a path to P = NP: why can't NP also be simulated by a tape and a stack? Finally, since an extended abstract of this paper had been published in the *Proceedings of the 1st STOC in 1969* [Coo69], it tells us that Stephen Cook had spent time thinking about how to understand the relationship between (what is now called) P and NP at least two years before he penned the famous paper that he presented at the 3rd STOC.

4.3 The Paper at the 3rd STOC

The title, "The complexity of theorem-proving procedures," comes across as a rather unexpected description of the paper's momentous ideas and results, but the narrative itself is quite direct. The first sentence of the summary announces the main result, that nondeterministic polynomial time reduces to tautologies, while the second sentence defines what would become known as "Cook reductions"—even though, as far as I can tell, only "Karp reductions" are used in this paper.[2] The third sentence states that the clique problem has the same polynomial degree of difficulty as the tautologies problem—they both belong to an equivalence class that would soon be known as "NP-complete."

The mathematical framework is laid out next. A problem is a set of strings—this is the legacy of formal language theory that, in my opinion, has rendered Complexity Theory a bit more awkward than it had to be. Polynomial degrees are defined, and so is P—alias $\mathcal{L}_*$, the polynomial degree of the empty language. Then Cook recites several problems resisting classification in P: The subgraph isomorphism problem exemplified by clique, the graph isomorphism problem, DNF tautologies, 3DNF tautologies (primality is also mentioned). Since the class NP is not defined or dealt with explicitly, except to reduce later from its generic problem, the author

2. See the chapter by Nicholas Pippenger for a full discussion of these reductions.

is cavalier with complement, and as a result his list contains specimens from both NP and coNP.

The proof of the main result—that any nondeterministic polynomial time computation can be rendered as a DNF—starts and proceeds in the manner we all know and love, until one comes across an interesting oversight: at the most crucial moment of the proof, when the nondeterminism of the machine should be confronted head on by Boolean logic, a plainly deterministic machine is instead simulated! Since at that point only Steve Cook knew how to prove this theorem, in the coming years the mistake would give pause, and a little fright, to many a reader. But of course it can be fixed easily with just another layer of Boolean logic.

The next theorem contains two reductions from DNF tautologies: to 3DNF and to clique. These are the first reductions from SAT—legions would follow, of course. In the preamble of the proof, the author admits, possibly with faint self-reproach, that he failed to devise similar reductions to graph isomorphism and primality, while he later points out that 2DNF is polynomial-time solvable through the Davis–Putnam algorithm. The two theorems provide clear evidence, the author argues, that Boolean tautologies are not easy to prove and that subgraphs are hard to identify, and the same must hold for "any combinatorial problem to which tautologies is P-reducible." Finally, Cook proposes that it is worth investing *considerable effort trying to prove* that such problems cannot be solved in polynomial time—a question that, he finally points out, cannot, alas, be settled by the diagonal argument.

4.4 The Mystery of Section 4.3

I find the last part of this paper brilliant and fascinating, and yet a rather odd coda of the paper that launched modern complexity theory. In a dense section entitled "The predicate calculus," Cook turns to the complexity of proving inconsistency in first-order logic, which is tricky to define since Turing proved that the problem is undecidable. The complexity of falsifying an inconsistent first-order formula is a function not of the length of the formula but of an uncomputable syntactic measure of its "logical complexity" (the minimum size of a canonical Herbrand expansion of the functional form of the formula containing a Boolean contradiction). Very roughly, Cook proves that this complexity function cannot be much smaller than the square root function (Theorem 3A) while it is at most an exponential (Theorem 3B).

The proof of the lower bound is interesting as it vaguely parallels the proof of the paper's Theorem 1, albeit in the logic and decidability domain: it entails a reduction from the halting problem of Turing machines to the consistency of first-order formulae. The logical complexity (in the above sense) of the constructed formula is quadratic in the number of steps of the Turing machine (hence the square root

in the lower bound, while logarithmic factors are lost in the concluding diagonal argument). The reduction itself parallels Turing's 1937 construction for the proof of the Entscheidungsproblem, but Cook's proof instead cites a simpler 1964 reduction by Hao Wang from Turing machine computation to a much more restricted class of first-order formulae, the so-called *AEA case* [Wan62].

There is an exponential gap between the upper and lower bounds in Theorem 3A and 3B, and Cook points out that there are important reasons for this: a better upper bound in 3B would imply that nondeterministic time can be simulated in better than exponential deterministic time. And a larger lower bound in 3A would come close to showing that Boolean tautologies require superpolynomial time. There is a typo in this last statement: "1A" is written instead of "3A."

It seems magical that this part's concluding argument brings up, in a very natural way and yet starting from very far, the two important questions ("can tautologies be recognized in polynomial time?" and "can nondeterministic time be simulated better than exponentially?") that have been already shown equivalent in the paper's Theorem 1.

Outside science, what I enjoy most is writing stories. After spending time with Steve Cook's paper, I have come to suspect that there is a beautiful, intriguing story behind it. Perhaps the paper did not start as an assault on the complexity of Boolean logic and related combinatorial problems. Maybe it started with the elegant results in Section 4.3, a principled way of assessing the performance of the automated theorem provers in first-order logic that had become fashionable in the 1950s and 1960s—hence the paper's title. Proving and contemplating these results created an explosive brew. This genre of theorem proving relates truth in first-order logic to tautology in Boolean logic. The ancient reduction from Turing machines to logic was remembered—starting of course from deterministic machines, which can even help explain the famous deterministic slip in the proof of Theorem 1. To fathom the origin of the exponential gap between the two bounds of Theorem 3, the problem of simulating deterministically a nondeterministic machine was considered side-by-side with the question of the complexity of tautologies. Perhaps this unique combination of mathematical ingredients was soon followed by a spark of inspiration, which led to what became Theorem 1.

And perhaps "1A" is not a chance typo but the telltale fossil of the moment when Theorem 1 of a nice paper on "the complexity of theorem proving procedures" was re-numbered as Theorem 3 of a larger opus destined to become a revered classic.

4.5 Aftermath

Today, the theory of computation is a thriving, dynamic, sophisticated, and cohesive mathematical discipline, continually expanding its horizons and adapting its

reach yet always centered on the dual subject of *algorithms and complexity.* It is also clear that the emphasis on these two subjects was not there during the 1960s. No single event ever launches a field, but I believe that two landmark research developments must have eased this transition: the development of the first surprising and mathematically sophisticated algorithms exemplified by Strassen's in 1969 [Str69] and the four Russians' [ADKF70] in 1970, and the rise of NP-completeness in 1971–1972.

Already at the next theory conference after the 3rd STOC, the 12th SWAT[3] in October 1971, two of the five sessions had algorithmic themes: Bob Tarjan spoke about depth-first search, Hopcroft and Karp presented their $n^{\frac{5}{2}}$ bipartite matching algorithm, while algebraic complexity also had a presence (as it already had five months earlier at the 3rd STOC). Something new was undoubtedly in the air. This was not lost to IBM—at that point the major corporate sponsor of research in theory, and many of the exponents of the new ideas were invited to a workshop at Yorktown Heights the following March. There, barely ten months after Cook's talk, Richard Karp presented his new paper, "Reducibility among combinatorial problems" [Kar72].

After reading Cook's paper, Karp took a careful look around him and noticed something extraordinary: For almost every problem that he knew—and Karp did know many problems—either there was a polynomial time algorithm or a reduction from SAT.[4] Karp's paper established, with its list of 21 complete problems, that the pattern identified by Cook was not an isolated phenomenon but a classification methodology of surprising power and reach. This was not at all obvious at the time. In a 1975 paper [Lad75a], Richard Ladner proved that, if $\mathsf{P} \neq \mathsf{NP}$, then there is an infinity of intermediate degrees between P and the complete problems, a situation quite familiar from the study of recursive functions. And yet, somehow, this plethora of halfway problems has a way of keeping out of sight. Apart from graph isomorphism, a few difficult total functions requiring their own treatment see, for example, Goldberg and Papadimitriou [GP18], and the still unfolding epic of approximability, there are very few corners of the NP realm that have not been completely sorted out with respect to their complexity through the use of NP-completeness.

Meanwhile, in the Soviet Union, a brilliant if laconic paper was published in 1973 by a 25-year-old mathematician named Leonid Levin proving that several problems, including a variant of the tautology problem, are complete for

3. "Switching and Automata Theory," a conference that would soon be renamed FOCS.

4. Karp also cites three problems—Graph Isomorphism, Linear Inequalities, and Nonprimes—that are in NP but not NP-complete.

nondeterministic polynomial time [Lev73]—even though the rest of the world did not take notice for another few years. One stylistic difference is that Levin, unhampered by the formal language tradition, speaks not of sets of strings but of *search problems*, multivalued functions from inputs to verifiable solutions—see Trakhtenbrot [Tra84] for a paper recounting the research tradition on *perebor* (Russian for "exhaustive search") that culminated in this paper. Levin had been speaking publicly about his results since two years before his paper appeared, and hence Theorem 1 of Cook's paper is now known as *the Cook–Levin theorem*.

Back in the West, our field had started to take its distinctive shape. Don Knuth, the ingenious chronicler of the mathematical nature of computer science, got busy with the task of giving the phenomenon a name; after a long and almost democratic process, the adjective NP-*complete* emerged victorious [Knu74]. A couple of years later, two young researchers at Bell Labs, Mike Garey and David Johnson, started drafting chapters of a book on the subject [GJ79]. This book would become arguably one of the most leafed through scientific documents of all times, as NP-completeness permeated, illuminated, sharpened, informed, and transformed all of computer science, and much of mathematics and science—as well as general scientific culture.

The mathematical problem that Stephen Cook modestly encouraged his readers to "spend considerable effort" pondering may be, ultimately, the most weighty consequence of his NP-completeness paper. The P vs. NP question—"Can exhaustive search always be avoided?"—has emerged over the past five decades as an essential scientific conundrum, next to the origin of life and the unification of force fields. The kind of profound, consequential puzzle, concrete yet universal, that adds meaning not only to science but to human life itself.

5 The Cook–Reckhow Definition

Jan Krajíček

The Cook–Reckhow paper [CR79] introduced the notions of *propositional proof systems* and *polynomial simulations* among them, described several classes of logical propositional calculi and compared them with regards to their efficiency, and introduced the *pigeonhole principle tautology* PHP_n, which remains the prime example of a tautology hard to prove in weaker systems, rivaled only by the tautology proposed earlier by Tseitin [Tse70]. It also showed that the central question of whether there exists a propositional proof system allowing polynomial size proofs of all tautologies is equivalent to a central question of complexity theory, namely whether the class NP is closed under complementation.

Classical proof theory of first-order logic developed in the first half of the 20th century assigned to proofs several combinatorial characteristics and some of them can be perceived as measures of complexity; for example, the height of a proof tree. Primary emphasis was on constructions producing various normal forms of proofs and the combinatorial characteristic helped to measure the progress of normalization constructions. The question about the minimum length of a proof of a statement (measured by either the number of steps or by the size, i.e., the number of symbols) was also studied, primarily in the context of speed-up results; the studies by Gödel [Göd36], Mostowski [Mos52], Ehrenfeucht and Mycielski [EM71], and Parikh [Par71, Par73] can serve as illustrations of this research. The results rest on constructions underlying the undecidability of the Halting problem or Gödel's Incompleteness theorem and do not give any insight[1] into analogous problems in propositional logic.

1. In fact, Parikh [Par71] introduced the theory PB (now called $I\Delta_0$, cf. Krajíček [Kra95, Kra19]) that later in the 1980s turned out to be important for the development of proof complexity.

It was the Cook–Reckhow 1979 paper [CR79] that defined the area of research we now call proof complexity. There were earlier papers that contributed to the subject as we understand it today, the most significant being Tseitin's [Tse70]. But none of them introduced general notions[2] that would provide an explicit and universal link between lengths-of-proofs problems and computational complexity theory.

In this chapter, we shall highlight three particular definitions from the paper: of proof systems, p-simulations, and the formula PHP_n, and discuss their role in defining the field. We will also mention some related developments and open problems. In particular, we shall show that the general definition of proof systems that has seemingly little to do with how ordinary logical calculi are defined is actually equivalent to the calculi definition with a more general treatment of logical axioms than is usual (Section 5.1), we shall present the optimality problem stemming from the notion of simulations (Section 5.2), and we shall discuss the role of the PHP_n formula in proof complexity lower bounds, its limitations and modern variants aimed at strong proof systems (Section 5.3). The paper [CR79] also discusses a few measures of complexity of proofs other than proof size and describes some relations among them; we shall not discuss this and instead we refer the reader to available literature.

The Cook–Reckhow paper [CR79] had a precursor [CR74], an extended abstract that summarized some research presented later in Cook and Reckhow [CR79], as well as from Reckhow's Ph.D. thesis [Rec76]. This earlier paper differs from Cook and Reckhow [CR79] in several aspects: it uses simulations (see Section 5.2) as opposed to the latter's finer notion of p-simulations, and it contains neither the Extended Frege system nor the PHP tautology. It presents a rather succinct version of the construction underlying Reckhow's theorem that was replaced in Cook and Reckhow [CR79] by a similar statement for Extended Frege systems with much easier (and more illuminating) proof (cf. Section 5.2). It also treats in detail the sequent calculus that is only glanced over in Cook and Reckhow [CR79], and it derives some super-polynomial lower bounds, using Tseitin's [Tse70] results, for weak proof systems such as tree-like resolution or semantic trees.

The aim of this chapter is to give an idea to the nonexpert reader about the main ideas stemming from Cook and Reckhow [CR79] and about the fundamental problems of proof complexity. Further information about proof complexity, its basic

2. Tseitin's [Tse70] paper offers no motivation for the research reported there but one of the motivations were questions we now formulate as the P vs. NP problem (another motivation was computer processing of natural languages) and the special role the *Entscheidungsproblem for propositional calculus* plays in them (personal communication).

as well as advanced parts, and about topics in mathematical logic and complexity theory it relates to can be found in Krajíček [Kra95, Kra19].

5.1 Definition of Proof Systems

The main example of a logical propositional calculus to keep in mind is a *Frege system*. It is any calculus operating with propositional formulas over a complete basis of logical connectives (i.e., all Boolean functions can be defined in the language), having a finite number of sound axiom schemes and inference rules that are *implicationally complete*. The latter term means that if a formula A is a logical consequence of formulas $B_1, \ldots, B_m$, then it can be derived from them in the calculus. An example of a complete language is the DeMorgan language with constants $0, 1$ (corresponding to false and true) and connectives $\neg, \vee$, and $\wedge$. We shall denote by TAUT the set of tautologies in this language, and we shall tacitly assume that TAUT $\subseteq \{0,1\}^*$, with formulas being encoded by binary strings in some natural way.

There are a number of such systems described in logic textbooks, and they are often called *Hilbert-style*, referring to Hilbert's work in proof theory [HA50, HB34, HB39]. The form of calculi is based on Frege's [Fre79], hence the name Cook and Reckhow [CR79] chose for this class of propositional calculi.

The calculi are *sound* (every provable formula is a tautology) and *complete* (every tautology is provable). In addition, the key property singled out by Cook and Reckhow [CR79] is that to recognize whether a string of symbols is a valid proof in the calculus or not is computationally feasible: it can be done by a p-time algorithm. This leads to the following fundamental definition.

Definition 5.1 **Cook–Reckhow** [CR79]
A *propositional proof system* is any p-time computable function

$$f : \{0,1\}^* \to \{0,1\}^*$$

such that

$$TAUT = Rng(f).$$

Any $w \in \{0,1\}^*$ such that $f(w) = A$ is called an f-proof of A.

Cook and Reckhow [CR79, Def. 1.3] actually define more generally a proof system for any $L \subseteq \{0,1\}^*$ by the condition $L = Rng(f)$, and consider proof systems for the set of tautologies in any fixed language.

A Frege system F can be represented by a function f that takes a string w and maps it to the last formula of w, if w is a sequence of formulas that forms

a valid F-proof, or to the constant 1 if w is not an F-proof. The soundness of F implies that $Rng(f) \subseteq$ TAUT and its completeness implies the opposite inclusion TAUT $\subseteq Rng(f)$.

It is easy to see that a number of other classes of propositional calculi considered in mathematical logic literature fit the definition in the same sense as Frege systems do. These calculi include resolution, sequent calculus, and natural deduction. Less usual examples of propositional proof systems can be constructed as follows. Take a consistent first-order theory axiomatized by a finite number of axioms and axiom schemes that is sound and contains some simple base theory (in order to guarantee both the correctness and the completeness) and interpret it as a proof system: a proof of formula A is a proof in the theory of the formalized statement $A \in$ TAUT. Other examples are logic calculi that are set up to prove the unsatisfiability of formulas: these can be interpreted as proof systems by accepting a refutation of $\neg A$ as a proof of A.

In addition, the general form of the definition allows us to interpret various calculations in algebra as propositional proofs. Here it is more natural to speak about refutation systems. If we have a CNF formula that is a conjunction of clauses C_i, we can represent each C_i by a constraint of an algebraic form and use a suitable algebraic calculus to derive the unsolvability of the formula. For example, a clause

$$p \vee \neg q \vee r$$

together with the requirement that we look for 0–1 solutions can be represented by polynomial equations

$$(1-p)q(1-r) = 0\ ,\ p^2 - p = 0\ ,\ q^2 - q = 0\ ,\ r^2 - r = 0$$

the first equation states that the clause contains a true literal while the last three equations force 0–1 solutions over any integral domain. In this case we can use a calculus deriving elements of the ideal generated by the equations representing similarly all clauses of the formula, trying to derive 1 as a member of the ideal and thus demonstrating the unsolvability of the equations and hence the unsatisfiability of the formula.

Another approach is to represent the clause as integer linear inequalities

$$p + (1-q) + r \geq 1\ ,\ 1 \geq p, q, r \geq 0$$

and use some integer linear programing algorithm to derive the unsolvability of the system of inequalities representing the whole CNF formula. It is a great advantage of Definition 5.1 that it puts all these quite different formal systems under one umbrella.

Proof systems can be also defined equivalently in a relational form. A *relational propositional proof system* is a binary relation $P(x, y)$ that we interpret as the provability relation *y is a proof of x*. It is required that it is p-time decidable and that for any formula A it holds that:

$$A \in \text{TAUT} \quad \text{iff} \quad \exists w P(A, w).$$

This is closer in form to logical calculi (and can be represented by the function version as Frege systems were before) but it is equally general: a functional proof system f is represented by the relation $f(y) = x$.

A proof system f is *p-bounded* iff there exists $c \geq 1$ such that for all A, $|A| > 1$,

$$A \in \text{TAUT} \quad \Rightarrow \quad \exists w (|w| \leq |A|^c) f(w) = A.$$

In the relational form this would read

$$A \in \text{TAUT} \quad \Rightarrow \quad \exists w (|w| \leq |A|^c)\ P(A, w)$$

and combining this with soundness we get

$$A \in \text{TAUT} \quad \Leftrightarrow \quad \exists y (|y| \leq |A|^c) P(A, y).$$

The right-hand side expression has the well-known general form in which any NP set can be defined. Hence, we get as a simple but important corollary to the definition the following statement (the second equivalence uses Cook's theorem: the NP-completeness of SAT, cf. Cook [Coo71b]).

Theorem 5.1 **Cook–Reckhow [CR79]**
A p-bounded proof system exists iff *TAUT* $\in$ NP iff NP $=$ coNP.

This theorem determines

Problem 5.1 **Main problem of proof complexity**
Is there a p-bounded proof system for TAUT?

By Theorem 5.1, showing that no p-bounded proof system exists would imply, in particular, that P $\neq$ NP because P is closed under complementation. On the other hand, defining a p-bounded proof system f would allow to witness various coNP-properties by short witnesses (f-proofs); Cook and Reckhow [CR74] mentions the property that two graphs are not isomorphic.

One may consider variants of the definition of proof systems when the provability relation is not necessarily decidable by a p-time algorithm but only by more general algorithm; for example, using some randomness. My view is that this

changes the basic problems of proof complexity substantially. While it may link propositional proof systems with various other proof systems considered in different parts of complexity theory, it is not clear that it will shed light on proof complexity proper. This may change if some of these other parts of complexity theory advance significantly on their own fundamental open problems.

The Cook–Reckhow definition is handy for establishing Theorem 5.1 and the connection to complexity theory but the reader may wonder if it does not deviate from logical form of calculi too much. In fact, it can be shown that every proof system can be p-simulated (in the sense of the next section) by a Frege system whose set of axioms is not given just by a finite number of axiom schemes but is a possibly infinite but easy to recognize (in p-time, in particular) sparse subset of TAUT. Doing this precisely is rather technical and we refer the reader to Krajíček and Pudlák [KP89] and Krajíček [Kra95, Kra19].

5.2 Simulations among Proof Systems

When studying the problem whether some proof system is p-bounded, it is useful to be able to compare two proof systems with respect to their efficiency. The following two notions[3] are aimed at that.

Definition 5.2 **Cook–Reckhow [CR79]**
Let f, g be two proof systems. A *simulation* of g by f is any function

$$h : \{0,1\}^* \to \{0,1\}^*$$

such that for all $w \in \{0,1\}^*$, $|h(w)| \leq |w|^c$, for some independent constant $c \geq 1$ and all $|w| > 1$, and such that

$$f(h(w)) = g(w).$$

Simulation h a is *p-simulation* if it is p-time computable.

Proof system f *(p-)simulates* g ($f \geq g$ and $f \geq_p g$ in symbols, respectively) iff there is a (p-)simulation of g by f.

In other words, the statement that $f \geq g$ says that if we replace f by g we can speed-up proofs at most polynomially, while the statement that $f \geq_p g$ says that we can even efficiently translate g-proofs into f-proofs. Both these relations are quasi-orderings (we get partial orderings after factoring by the equivalence relations of mutual simulations).

The significance of p-simulation in the context of the system PV is discussed in S. Buss's Chapter 6 in the current volume.

3. p-simulations are also defined in Cook [Coo75b].

There are other options for defining a quasi-ordering on proof systems. In particular, if we did not insist in Definition 5.1 that all proof systems prove tautologies in the same language (we have defined TAUT using the DeMorgan language only) but allowed tautologies in different languages, then a (p-)simulation should allow translations of formulas as well as proofs. By insisting that the target set is TAUT, we forced the incorporation of such a translation of formulas into the definition of particular proof systems that may operate with formulas in other languages or even with polynomials or other objects. In fact, considering, instead of propositional proof systems, proof systems for any coNP-complete set, we ought to allow p-reductions between such sets and TAUT.

However, for positive results (as is Theorem 5.2 bellow) p-simulations allow the formulation of the strongest possible statements while the strongest negative results (obtained by proving a super-polynomial lower bound for f-proofs of formulas for which there are polynomial size g-proofs) talk about super-polynomial speed-ups and hence about the nonexistence of simulations. Thus, the two types of simulations serve their purpose very well.

Cook and Reckhow [CR79] compared various logical proof systems in terms of p-simulations; the following statement summarizes their most memorable results in this respect.

Theorem 5.2 **Cook–Reckhow [CR79]**

1. All Extended Frege systems in all languages p-simulate each other.
2. Frege systems and the propositional parts of natural deduction and sequent calculus mutually p-simulate each other.
3. Extended Frege system EF and Tseitin's Extended Resolution ER are p-equivalent, and they are p-simulated by any Frege system with the substitution rule.

Extended Frege systems EF were defined in Cook and Reckhow [CR79] in a direct analogy with the Extended Resolution system ER of Tseitin [Tse70]. Any such system starts with a Frege system and in addition allows the abbreviation of formulas by new atoms that may be subsequently used in a proof. In particular, during an EF-proof we can take a new atom q (an *extension atom*) not used so far and not occurring in the target formula A to be proved, any formula D not containing q, and introduce the equivalence $q \equiv D$ (represented in the language of the system) as a new *extension axiom*. Note that EF is not a Frege system as the introduction of extension axioms does not fit the schematic way Frege axioms are supposed to be defined. The first statement in the theorem is a weaker version of Reckhow's theorem [Rec76], which is stated for Frege systems. The version for Extended Frege

systems is much easier to prove (see Krajíček [Kra95, Kra19] for published proofs of the stronger version).

For the definition of natural deduction see Prawitz [Pra65], for sequent calculus see any of Gentzen [Gen36] and Krajíček [Kra95, Kra19] (the sequent calculus part of the statement is just mentioned in Cook and Reckhow [CR79] while natural deduction is treated in detail). The substitution rule allows the inference from a formula $B(p_1, \ldots, p_m)$ of an arbitrary substitution instance $B(C_1, \ldots, C_m)$ in one proof step. A Substitution Frege system SF is a Frege system augmented by this rule. It was proved later in Dowd [Dow79] (indirectly) and in Krajíček and Pudlák [KP89] (an explicit p-simulation) that EF actually p-simulates SF as well.

An illuminating description of EF is that it is essentially a Frege system that operates with circuits rather than with formulas; this has been made precise in Jeřábek [Jeř04]. Perhaps even more useful is the statement that the minimum size s of an EF-proof of formula A is proportional to the minimum number of steps in a Frege proof of A and $|A|$, or to the minimum number of different formulas that need to occur as subformulas in any Frege proof of A and $|A|$, cf. Cook and Reckhow [CR79] or Krajíček [Kra95, Kra19]. Hence moving from F to EF means that we are replacing the size as the measure of complexity of Frege proofs by the number of steps. This is interesting because from the point of view of mathematical logic the number of steps is a very natural complexity measure.

EF is also important because of its relation to a particular theory PV introduced by Cook [Coo75b] at the same time (he used ER in his paper). This is discussed in S. Buss's Chapter 6 in the current volume. Theory PV (standing for polynomially verifiable) allows the formalization of a number of standard computational complexity constructions and arguments. Understanding the power of the proof system EF and, in particular, showing that it is not p-bounded is considered in the field as the pivotal step towards solving the Main Problem and proving that $\mathsf{NP} \neq \mathsf{coNP}$. In particular, it is also known that any super-polynomial lower bound for EF implies that $\mathsf{NP} \neq \mathsf{coNP}$ is consistent with PV (cf. Krajíček [Kra19, section 12.4]).

We shall mention one problem formulated only later in Krajíček and Pudlák [KP89] that is, however, natural and is implicit in the definition of simulations.

Problem 5.2 **Optimality Problem**
Is there a proof system that (p-)simulates all other proof systems?

Such a maximal proof system is called *(p-)optimal* after Krajíček and Pudlák [KP89]. We have (names for) three types of proof systems whose existence is considered by most researchers unlikely: p-bounded, p-optimal, and optimal. Every p-bounded or p-optimal proof system is also optimal, and this rules out three out

of eight possibilities for the existence/nonexistence of objects of these three types. At present we cannot rule out any of the remaining five scenarios:

- *A p-bounded, p-optimal proof system P_1 exists.*

  Having such an ideal proof system, we do not need to consider any other: even searching for proofs in any other proof system can be reduced to searching for P_1-proofs. (We ignore here that p-reductions themselves increase polynomially the time complexity of a proof search algorithm and may transform a combinatorially transparent one into a complex one, cf. the last paragraph of this section.)

- *A p-bounded proof system P_2 exists but no p-optimal does.*

  While p-size P_2-proofs would exist for each tautology, finding them may be difficult and it may help to consider different proof systems for different (classes of) tautologies.

- *A p-optimal proof system P_3 exists but no p-bounded does.*

  Here we can restrict our attention to P_3: it is also optimal and search for proofs in any proof system can be replaced by a search for P_3-proofs.

- *An optimal proof system P_4 exists but no p-bounded or p-optimal does.*

  Proving lengths-of-proofs lower bounds (or upper bounds, for that matter) can be restricted to P_4 but proof search may benefit from considering different proof systems for different classes of tautologies.

- *None of these ideal objects exist.*

  This appears to be the most likely scenario.

At present we cannot rule out that a Frege system is one of $P_1, \ldots, P_4$. The Optimality problem is related to a surprising number of varied topics in proof theory (quantitative Gödel's theorem), finite model theory, structural complexity, and some other (cf. Krajíček [Kra19, chapter 21]).

An interesting question left out by Cook and Reckhow [CR79] as well as in later literature is how to compare proof search algorithms. A tentative definition was proposed in Krajíček [Kra19, section 21.5] .

5.3 Hard Tautologies and the PHP$_n$ Formula

In order to prove lengths-of-proofs lower bounds for a proof system, we start with a suitable candidate tautology that we conjecture to be hard to prove (i.e., requiring long proofs) therein. A particular tautology for this purpose based on the pigeonhole principle was proposed in Cook and Reckhow [CR79]. The formula, to be denoted PHP$_n$, is built from atoms p_{ij} with $i \in [n] := \{1, \ldots, n\}$ and $j \in [n-1]$,

for $n \geq 2$. Thinking of p_{ij} as representing the atomic statement that *i maps to j*, we can express that the map is defined at i by the clause

$$\bigvee_j p_{ij} \tag{5.1}$$

the fact that j can be the value of at most one i by

$$\bigvee_{i_1 \neq i_2} \neg p_{i_1 j} \vee \neg p_{i_2 j} \tag{5.2}$$

and the fact that i maps to at most one value by

$$\bigvee_{j_1 \neq j_2} \neg p_{i j_1} \vee \neg p_{i j_2}. \tag{5.3}$$

Taking the conjunction of these clauses for all choices of i and j states that

$$\{(i,j) \in [n] \times [n-1] \mid p_{ij} = 1\}$$

is the graph of an injective map from $[n]$ into $[n-1]$. No such map exists and hence the negation of the conjunction is a tautology. This leads to the following definition.

Definition 5.3 **Cook–Reckhow [CR79]**
For any $n \geq 2$, PHP_n is the disjunction of negations of clauses in (5.1) for all $i \in [n]$, in (5.2) for all $j \in [n-1]$, and in (5.3) for all $i \in [n]$.

In fact, to reach a contradiction we do not need the assumption that it is the graph of a function, a multi-function suffices (if i occupies more values j it is harder to be injective). In other words, we do not need to include the clauses from (5.3) and Cook and Reckhow [CR79] did not include them. Nowadays, the definition of PHP_n as formulated above is more customary and proving lower bounds for it yields stronger results than for the more economical version (the principle assumes more and hence it is logically weaker).

Cook and Reckhow [CR79] showed that it is possible to prove PHP_n in Extended Frege systems by a proof of size polynomial in n (note that the size of PHP_n is also polynomial in n). In fact, they introduced EF in order to formalize smoothly the inductive argument: from an assignment violating PHP_n we can define (using the extension rule) an assignment violating PHP_{n-1}. Hence PHP_n has also a proof in Frege systems with a polynomial number of steps (but having large size). Buss [Bus87] improved the result (by a substantially different construction formalizing

counting) and proved that Frege systems actually also admit polynomial size[4] proofs of PHP_n.

On the other hand, in a breakthrough result, Haken [Hak85] proved a first lower bound for resolution using PHP_n, and the same formula was proved to be hard for constant depth subsystems of any Frege system in the DeMorgan language by Ajtai [Ajt88] (Haken's lower bound was exponential while Ajtai's super-polynomial—its rate was later improved to exponential too by Krajíček et al. [KPW95] and Pitassi et al. [PBI93]). The same formula (represented by polynomial equations similarly as in Definition 5.1) were used by Razborov [Raz98] for his lower bound for polynomial calculus, an algebraic proof system manipulating polynomials.

There is an important variant of the PHP formula considered first by Paris et al. [PWW88] in the context of bounded arithmetic: allow i to range over a much bigger set than $[n]$; for example, over $[2n]$ or even $[n^2]$. Similarly as PHP is related to counting these *weak* pigeonhole principles relate to approximate counting and Paris et al. [PWW88] showed that they can sometimes be used in place of PHP proper and that, crucially, they are easier to prove. Their proof (formulated using bounded arithmetic) gives quasi-polynomial size proofs in constant depth Frege systems of formulas formalizing these weaker principles for $n \geq 2$.

Even if the formula PHP_n itself cannot be used as a hard example for proof systems like Frege or EF, formulas formalizing a form of a weak PHP in a different way possibly can. It has been an insight of Wilkie (result reported in Krajíček [Kra95, section 7.3]) that the *dual weak PHP* for p-time functions is important in bounded arithmetic (this has been much extended by Jeřábek [Jeř04, Jeř07, Jeř09]). The principle says that no p-time function g when restricted to any $\{0,1\}^n$ can be onto $\{0,1\}^{2n}$. Now take an arbitrary $b \in \{0,1\}^{2n} \setminus Rng(g_n)$, where g_n is the restriction of g to $\{0,1\}^n$, and define propositional formula

$$\tau_b(g_n)$$

expressing $\forall x \in \{0,1\}^n g_n(x) \neq b$. The formula uses n atoms for bits of x and a further $poly(n)$ atoms for bits of the computation y of g_n on x and says, in a DNF form, that either y is not a valid computation on input x or the output of the computation differs from b. Clearly,

$$\tau_b(g_n) \in \mathrm{TAUT} \;\Leftrightarrow\; b \notin Rng(g_n).$$

These formulas were defined in Alekhnovich et al. [ABRW04] and Krajíček [Kra01] and lead to the theory of proof complexity generators proposing several candidate

4. In Buss [Bus15], he showed that the idea of the original Cook–Reckhow proof in EF can be formalized in Frege systems by quasi-polynomial size proofs, utilizing st-connectivity.

tautologies of the form above as possibly hard for strong (or all) proof systems. The reader may find an overview of the theory in Krajíček [Kra10, chapters 29 and 30] (no need to read the first 28 chapters).

Acknowledgments

I thank Sam Buss, Bruce M. Kapron, Igor C. Oliveira, and Jan Pich for comments on earlier versions of this paper.

6 Polynomially Verifiable Arithmetic

Sam Buss

6.1 Introduction

Steve Cook's 1975 paper, "Feasibly constructive proofs and the propositional calculus" [Coo75b], was a landmark in the study of weak formal systems and propositional proof complexity. It introduced the equational proof system PV for reasoning about polynomial time identities and established an unexpected and remarkable connection between provability in PV and polynomial size extended resolution (extended Frege) proofs of propositional formulas. It established that PV can prove the consistency of extended resolution, and that extended resolution is the strongest propositional proof system which can be proved consistent by PV. As a consequence, extended resolution can give polynomial size proofs of its own partial consistency.

Subsequent work discovered close connections between PV and first-order fragments of bounded arithmetic[1] such as $I\Delta_0$ and especially S^1_2. Many of these fragments of bounded arithmetic have their own connections to propositional proof complexity. Cook's paper [Coo75b] serves as the foundational template for these still on-going developments.

The first motivation—and perhaps the primary motivation—for Cook [Coo75b] was the approach suggested by Cook and Reckhow [CR74] for resolving the P versus NP question. That paper, along with the later follow-up paper [CR79], proposed proving that $\mathsf{NP} \neq \mathsf{coNP}$ by proving that there is no propositional proof system for which all tautologies have polynomial size proofs. For more on propositional proof

1. Descriptions of many of these theories of bounded arithmetic can be found in Buss [Bus86a], Hájek and Pudlák [HP93], Krajícek [Kra95], and Cook and Nguyen [CN10].

complexity and Cook and Reckhow's program for resolving P versus NP question, see the article in the present volume by Krajíček about Cook and Reckhow [CR79].

The second, and more direct, motivation was to study the concept of "feasibly constructive" proofs of universal statements. A proof of a universal statement $\forall x\, A(x)$ justifies the statement that $A(x)$ evaluates to true for every value of x. A *constructive* proof gives finitary or combinatorial justifications for the truth of each instance of $A(x)$. For *feasibly constructive* proofs, these justifications should be constructible in polynomial time; in addition, the correctness of the justifications should rest only on polynomial time computable concepts.

Let's make this second motivation more concrete. Assume $F_1, F_2, F_3, \ldots$ is a sequence of unsatisfiable propositional formulas in conjunctive normal form. Suppose that each F_n uses $O(n)$ variables and has total size $n^{O(1)}$; and that the F_n's are polynomial time uniform in that there is a procedure for constructing the F_n's that runs in time $n^{O(1)}$. For x a binary string of length n, let $A(x)$ assert that F_n is unsatisfiable.

The formulas $A(x)$ certainly have *constructive* proofs, namely by using the method of truth tables to evaluate F_n (where $n = |x|$). Proofs based on truth tables, however, are exponentially long. For the formulas $A(x)$ to have *feasibly constructive* proofs, we need polynomial size proofs of the unsatisfiability of the formulas F_n. Furthermore, we need (polynomial size) uniform justifications that these proofs of unsatisfiability are in fact valid proofs.

Cook [Coo75b] proposed an equational theory PV as the formal system for giving feasibly constructive proofs: namely, a proof in PV specifies polynomial size uniform proofs along with uniform justifications of the correctness of the proofs. Then, in a truly striking and insightful step, Cook that PV can prove consistent.

The developments are discussed in the next two sections.

6.2 The Equational Theory PV for Polynomial Time Computability

As just discussed, feasibly constructive proofs are not only required to be polynomial size but also the validity of the proofs must depend on only polynomial time concepts. For this purpose, Cook introduced equational theories, called PV and PV1, of polynomial time functions.[2] These theories were defined analogously to the equational theory PRA of primitive recursive arithmetic, as developed by Skolem [Sko23] and Goodstein [Goo54], but using polynomial time functions instead of

2. As shown by Cook [Coo75b], the two theories PV and PV1 are essentially equivalent. PV is a purely equational system. PV1 is defined by extending PV to allow Boolean combinations of equations. In this article, we often refer to just "PV," but most of our comments apply equally well to "PV1."

primitive recursive functions. PV is defined with special axioms that allow introducing symbols for polynomial time functions. These axioms define the polynomial time functions algorithmically in terms of recursion on notation as used by Cobham [Cob65] for the definition of the class $\mathcal{L}$ of polynomial time computable functions.[3] The axioms of PV also allow using induction on notation to prove facts about polynomial time equations. As a result, PV can introduce function symbols for all polynomial time computable functions and prove many straightforward properties of them. In addition, since PV can introduce *only* polynomial time computable functions and reason *only* with polynomial time computable concepts, theorems of PV can be viewed as feasibly constructive.

The *Verifiability Thesis* of Cook [Coo75b] stated that provability in PV exactly captures the intuitive concept of feasibly constructive theorems. Specifically, an equation $f(x) = g(x)$ with f and g polynomial time functions in $\mathcal{L}$ is called *polynomially verifiable*, or just *p-verifiable*, provided there is a proof that provides a feasibly constructive proof of $f(x) = g(x)$ in the sense described above.

Verifiability Thesis: [Coo75b]

An equation $t = u$ of PV is provable in PV if and only if it is p-verifiable.

One direction of the Verifiability Thesis is straightforward to prove: namely, any consequence $t = u$ of PV is p-verifiable. This follows from the fact that a PV proof of $t = u$ uses only polynomial time computable constructions and uses only simple notions of logic and a feasibly effective version of induction. In fact, as we discuss below, Cook [Coo75b] gives an even sharper form of this argument by showing that any PV proof of $t = u$ can be translated into a family of polynomial size ER proofs of the propositional translations of $t = u$.

The other direction of the Verifiability Thesis is essentially a philosophical statement about the nature of feasible constructivity. This is similar in spirit to the Church–Turing thesis about the nature of effective computability; however, the Verifiability Thesis is more subtle since it does not concern just polynomial-time *computability* but rather seeks to characterize polynomial-time *constructive reasoning*. The Verifiability Thesis states that PV can formalize all feasibly constructive reasoning about polynomial time computable objects. This seems very plausible as no one has been able to formulate feasibly constructive proofs that are not formalizable in PV. In addition, the present author formulated an ostensibly stronger theory S^1_2 for polynomial time functions in Buss [Bus86a]: it turns out that S^1_2 (when

3. The definition of $\mathcal{L}$ in Cobham [Cob65] used a base 10 representation of integers, whereas PV defined $\mathcal{L}$ using a dyadic notation for integers. The differences between the two definitions are inessential, however; they both capture polynomial time computability in a natural and feasible manner.

expanded to the language of PV) is conservative over PV and thus cannot prove any new identities $t = u$ beyond those already provable in PV. This provides evidence that PV is a powerful but natural theory, and thus that PV already incorporates all feasibly constructive reasoning.

Sections 2–4 of Cook [Coo75b] introduced the theories PV and PV1 and proved a number of fundamental properties of these theories. These developments include:

- The theory PV treats (nonnegative) integers as being represented by *dyadic strings* instead of by binary representations. Dyadic strings use the alphabet $\{1, 2\}$, based on a notation of Smullyan.[4] Each nonnegative integer has a unique dyadic representation. For instance, the integers 0, 1, 2, 3, 4, 5, 6, 7 are represented by the strings $\varepsilon, 1, 2, 11, 12, 21, 22, 111$, where ε denotes the empty string. There are two successor functions s_1 and s_2 that map a string w to the string $s_i(w) = wi$. Thus, if w is the dyadic representation of an integer m, $s_i(w)$ is the dyadic representation of $2m + i$.
- The function symbols of PV are defined from a few base functions (including s_1 and s_2) using composition and *limited iteration on dyadic notation* in the style of Cobham [Cob65]. The function symbols thus explicitly represent polynomial functions, as computed by algorithms expressed in terms of composition and limited iteration. Conversely, by Cobham's characterization of polynomial time computability, every polynomial time function is represented by a function symbol of PV.
- The axioms and rules of inference for PV provide defining axioms for all polynomial time functions (all functions in $\mathcal{L}$), and *induction on (dyadic) notation*.
- Every PV provable formula $t = u$ is p-verifiable, and thus valid. Consequently, PV does not prove $0 = 1$ and is consistent.
- More general forms of recursion can be used in PV, including 2-recursion and n-recursion that allow definition by a recursion that act on multiple variables at once.
- PV1 is defined as the conservative extension of PV to allow formulas constructed using propositional connectives. The axioms of PV1 include substitution, tautological implication, and a form of induction.

4. The dyadic string representation was an optional choice; PV could have been defined to use binary representations instead of dyadic representations. The advantage of the dyadic representation is that integers have unique dyadic representations. This is not true for binary representations because of the possibility of leading zeros. This makes the two successor functions s_1 and s_2 more elegant for the dyadic representation.

- Gödel's second incompleteness theorem holds for PV; that is, PV does not prove CON(PV). The proof is based on an intensional approach, cf. Feferman [Fef60].

6.3 Extended Resolution and PV

The final sections of Cook [Coo75b] introduced a strikingly novel and important connection between PV and ER. ER is essentially equivalent to the extended Frege proof system (EF). In this article we will mostly talk about EF, unlike Cook [Coo75b] who worked with ER, as EF is more convenient for talking about propositional proof systems; however, working with EF instead of ER makes no essential difference.

Frege and Extended Frege, introduced by Cook and Reckhow [CR74, CR79], are propositional proof systems using, say, the connectives $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, and $\neg$. Frege proofs use a finite set of schemes of tautological axioms (for instance, $\varphi \wedge \psi \rightarrow \varphi$) and a finite set of schemes for inference rules (for instance, modus ponens). An extended Frege system includes the axioms and inference rules of a Frege system, plus the extension rule allowing inferring any formula

$$x \leftrightarrow \varphi,$$

where x is new variable; that is, x does not appear in φ, in the proof so far, or in the conclusion of the proof. A convenient way to think about extended Frege (and hence ER) proofs is that they are proofs in which each line represents a Boolean function computed by a polynomial size circuit. (See Jeřábek [Jeř04] for an explicit formalization of this. Here, "polynomial size" means polynomial size in terms of the size of the extended Frege proof.) More details on extended Frege proofs can be found in the article by Krajíček in the present volume.

The first part of the connection between PV and ER (stated below as Theorem 6.1) can be summarized in modern terms as follows. We start with the observation that each line in a PV proof is an identity $t(\vec{x}) = u(\vec{x})$ between polynomial time functions using variables $\vec{x}$. For simplicity, we assume $\vec{x}$ is a single variable and denote it just x. It is a general principle that any polynomial time computable function f has polynomial size circuits [cf. Coo71b]. For f a function symbol of PV, the polynomial size circuits for f are constructed so as to simulate the definition of f as a member of $\mathcal{L}$ using composition and limited recursion on dyadic notation. This gives, for any $n \geq 0$, a polynomial size circuit C_n which takes an input x of dyadic length n, evaluates the values of $t(x)$ and $u(x)$, and outputs *True* if the two values are equal. Since the identity $t(x) = u(x)$ is valid, the circuit C_n always outputs *True*.

The inputs to C_n are n Boolean inputs $x_1, \ldots, x_n$ representing the values of the n dyadic digits of x. We convert C_n to a valid propositional formula denoted $[\![t(x) = u(x)]\!]_n$. This formula uses the variables x_i plus extension variables that express the computation of C_n. Each gate $\mathfrak{g}$ in the circuit C_n has an associated extension variable $y_\mathfrak{g}$ representing the value computed by $\mathfrak{g}$. Let $Corr(\vec{y}, \vec{x})$ be the (polynomial size) DNF formula expressing that each $y_\mathfrak{g}$ has the correct value as computed by the gate $\mathfrak{g}$ from its inputs. (This is similar to the construction used by Cook [Coo71b] to prove the NP-hardness of satisfiability.) Then $[\![t(x) = u(x)]\!]_n$ is the propositional formula $Corr(\vec{y}, \vec{x}) \to y_{\mathfrak{g}_{\text{out}}}$, where $\mathfrak{g}_{\text{out}}$ is the output gate of C_n. In other words, $[\![t(x) = u(x)]\!]_n$ states that if the extension variables correctly encode the circuit's computation, then the circuit outputs the value *True*.

Note that $[\![t(x) = u(x)]\!]_n$ is a tautology (i.e., is valid) since C_n always outputs *True*. Indeed, the validity of the formulas $[\![t(x) = u(x)]\!]_n$ is equivalent to the validity of the equations $t(x) = u(x)$.

Theorem 6.1 (ER Simulation Theorem): [Coo75b]

If PV proves $t(x) = u(x)$, then the formulas $[\![t(x) = u(x)]\!]_n$ have polynomial size extended Frege proofs.[5]

The intuition for proving Theorem 6.1 is that the extended Frege proof traces the PV proof P of $t(x) = u(x)$ line by line, establishing the truth of each line in P. The value of n is fixed. Each line in P is an equation $v(x, \vec{z}) = w(x, \vec{z})$ using the free variable x with dyadic length n and free variables $\vec{z}$ whose dyadic lengths are implicitly polynomially bounded in terms of n. The proof introduces extension variables for every intermediate value used in P, including an extension variable for every gate in the (polynomial size) circuits that compute the terms $v(x, \vec{z})$ and $w(x, \vec{z})$.

There is also a version of Theorem 6.1 for PV1 in place of PV. The main difference is that lines in a PV1 proof are Boolean combinations of equations instead of just equations. The propositional translation works similarly for PV1 and respects the Boolean connectives.

Theorem 6.1 leaves open the possibility that perhaps a weaker propositional proof system than EF or ER could be sufficient to give polynomial size proofs of the formulas $[\![t(x) = u(x)]\!]_n$. This, however, is not the case. A general propositional proof system is defined as follows. (Again, see Chapter 5 in the current volume by Krajíček for more details.)

Theorem 6.1 [CR74, CR79]

5. Cook [Coo75b] stated and proved this theorem for extended resolution, hence its name. As already stated, we prefer to work with the equivalent extended Frege proof system.

A *propositional proof system* is a polynomial time function f from $\{0,1\}^*$ onto the set of tautologies.

The extended Frege proof system can be viewed as a propositional proof system in the abstract sense of Definition 6.1 by letting $f(w) = \varphi$ if w encodes a valid extended Frege proof of the formula φ, and letting $f(w)$ equal a fixed tautology otherwise. If $f(w) = \varphi$, we call w an *f-proof* of φ.

Let CON(f) be a PV formula expressing—in a natural way—the fact that f is a consistent propositional proof system.[6] One way to formalize CON(f) is to express that there is no f-proof of a particular nontautology, say $x_0 \wedge \overline{x_0}$. Another way to formalize CON(f) is as stating $\forall w\, \forall u\, \forall v\, \mathrm{CN}_f(w,u,v)$, where $\mathrm{CN}_f(w,u,v)$ states that it is not the case that both w encodes an f-proof of a propositional formula encoded by u, and v encodes a truth assignment that falsifies the formula encoded by u. Note that CN_f can be taken to be an equation in PV. We shall write SAT(u,v) to represent the PV formula expressing the property that v encodes a satisfying assignment for u: thus $\forall v\, \neg\mathrm{SAT}(u,v)$ expresses that u encodes an unsatisfiable formula. And CN_f can be expressed by the PV1 formula $f(w) = u \rightarrow \neg\mathrm{SAT}(u,v)$. The two formalizations of CON(f) are equivalent over PV. The definition using CN_f is essentially the one used by Cook [Coo75b], although he used the terminology "f is p-verifiable". So when we say PV proves the consistency of f, CON(f), we mean that PV proves the equation $\mathrm{CN}_f(w,u,v)$.

Theorem 6.2 [Coo75b]

PV proves CON(EF).

A corollary is that EF has polynomial size proofs of the propositional translations $[\![\mathrm{CON(EF)}]\!]_n$, namely the propositional formulas expressing the partial consistency of EF. These formulas state in essence that there is no EF proof P of a falsifiable formula where proof P is encoded by a string of n bits. This corollary is initially quite surprising in light of Gödel's incompleteness theorem, but it highlights a big difference between finitary consistency and ordinary (infinitary) consistency. An analogue has also been shown for first-order theories such as Peano arithmetic (for this see Pudlák [Pud86, Pud87b]). An analogue also holds for Frege systems, as shown by Buss [Bus91].

The next theorem is the main theorem of Cook [Coo75b]; it implies that extended Frege is the weakest propositional proof system that satisfies Theorem 6.1, and the strongest propositional proof system whose consistency is PV provable.

6. We follow the common conventions, as well as Cook [Coo75b], in using the notation CON(f). However, the notation is somewhat misleading: it is not the case that there is a fixed PV-formula CON($\cdot$) that takes a Gödel number for f as a parameter. A better notation would be CON_f, since each f has its own consistency statement.

Definition 6.2 [Coo75b]

Let f and g be propositional proof systems. We say f *p-simulates* g provided there is a polynomial time algorithm A such that if w is a f-proof of φ, then $A(w)$ is a g-proof of φ. We say that f *p-verifiably p-simulates* g provided this fact is provable in PV for a suitably formalized representation of A as a polynomial time function in $\mathcal{L}$.

Theorem 6.3 Main Theorem of Cook [Coo75b]

Suppose g is a propositional proof system such that PV proves CON(g). Then EF p-verifiably p-simulates g.

Theorem 6.1 established that the extended Frege proof system can give polynomial size proofs of the propositional translations of any PV provable identity. However, it still remains open whether there are polynomial size extended Frege proofs of *all* tautologies. Cook [Coo75b] identified this question as an important step toward proving $\mathsf{NP} \neq \mathsf{coNP}$ and $\mathsf{P} \neq \mathsf{NP}$. He conjectured in particular that the partial consistency statements $[\![\mathrm{CON(PV)}]\!]_n$ do not have polynomial size EF proofs. To formulate these, we view PV itself as a *propositional* proof system by treating a PV proof of $\neg\mathrm{SAT}(\ulcorner\varphi\urcorner, v)$ as a proof of φ. Here $\ulcorner\varphi\urcorner$ is the Gödel number of φ in some natural Gödel numbering scheme. The incompleteness theorem for PV—mentioned at the end of Section 6.2—implies that PV cannot prove the existence of (necessarily uniform) extended Frege proofs of the formulas $[\![\mathrm{CON(PV)}]\!]_n$. In light of the Verifiability Thesis, this means that CON(PV) is not p-verifiable. This, however, does not rule out the possibility of nonuniform extended Frege proofs or of extended Frege proofs whose properties cannot be proved in PV.[7]

6.4 Subsequent Developments

The connection between PV and extended Frege was the first linkage of its kind between arithmetic and propositional logic, but it was definitely not the last. The hallmarks of this connection are that we have

- A formal theory, in this case PV.
- A computational complexity class, in this case polynomial time (P).
- And a propositional proof system, in this case extended Frege (EF) or equivalently ER.

These enjoy the following properties:

7. Buss [Bus86a] later made a similar proposal using the "jump" of a theory.

- The provably total functions of the theory (PV), and the functions that can be used in induction axioms, are the functions from the complexity class (P).
- The theory (PV) proves the consistency of the propositional proof system (EF). Furthermore, the propositional proof system (EF) p-simulates any propositional proof system that is provably consistent (in PV).
- The lines that appear in the propositional proofs (of the system EF) have computational complexity corresponding to a nonuniform version of the complexity class (in this case, the lines are polynomial size circuits, hence in nonuniform P).

The next result of this type was due to Dowd [Dow78, Dow79], who developed a theory PSA of polynomial space arithmetic. The corresponding proof system was quantified propositional logic. This fulfills the template above since quantified Boolean formulas (QBF's) are the nonuniform analogue of polynomial space. An improved version of translations from a bounded arithmetic theory to quantified propositional logic was given later by Krajíček and Pudlák [KP90].

Paris and Wilkie [PW81] discovered a different connection between the bounded arithmetic theory $I\Delta_0$ (which was introduced by Parikh [Par71]) and polynomial size, constant depth Frege proofs. There is also a Paris–Wilkie translation for $I\Delta_0 + \Omega_1$ and quasipolynomial size constant depth Frege proofs; the axiom Ω_1 states the totality of the function $x \mapsto 2^{\log^2 x}$ corresponding to polynomial growth rate functions. This "Paris–Wilkie translation" differs from the "Cook translation" in that Paris and Wilkie used free second-order predicates and translated predicate values to propositional variables, whereas Cook used only first-order objects and translated bits (or, dyadic digits) of first-order objects to propositional variables. (The distinction between the Cook and the Paris–Wilkie translations becomes less clear in the much more recent second-order systems of Cook and Nguyen [CN10].)

The next main development was the definition of a hierarchy of first-order and second-order theories of bounded arithmetic by Buss [Bus86a]; the former are subtheories of $I\Delta_0 + \Omega_1$. One of the first-order theories, S^1_2, is in essence conservative over PV, and so has the same connection to extended Frege proofs as PV. Other theories S^i_2, T^i_2, U^1_2, and V^1_2 have computational complexity related to the levels of the polynomial time hierarchy, to polynomial space, and to exponential time. For example, the provably total functions of U^1_2 correspond to the polynomial space computable functions.[8] The first-order theories, S^i_2 and T^i_2, have Paris–Wilkie translations to quasipolynomial size, constant depth Frege proofs. Krajíček and Pudlák

8. It seems certain that U^1_2 is essentially conservative over PSA, but there is no proof of this in the literature.

[KP90] also gave a Cook-style translation from these theories to fragments of quantified propositional logic (these are subtheories of the quantified propositional proof system used by Dowd).

There is a great deal of subsequent development of formal proof systems with Cook-style connections to propositional proof systems. It is beyond the scope of this article to describe these, but many of them are described by Cook–Nguyen [CN10] who give a second-order formulation for all these theories. Figure 6.1 lists many of these results as well, with pointers to the literature. Another set of relations between consistency statements (in formal theories) for propositional proof systems and propositional proof complexity was given by Krajíček and Pudlák [KP89]. More connections between extensions of PV and propositional proof complexity can be found in Krajíček [Kra19].

In a different direction, the theory PV was generalized to intuitionistic theories by Buss [Bus86b] and Cook and Urquhart [CU93]. The latter papergives a very

Formal Theory	Propositional Proof System	Total Functions	
PV, S^1_2, VPV	EF, G^*_1	P	[Coo75b], [Bus86a], [CN10]
T^1_2, S^2_2	G_1, G^*_2	$\leq_{1-1}$(PLS)	[KP90], [KT92], [Bus86a], [BK94]
T^2_2, S^3_2	G_2, G^*_3	$\leq_{1-1}$(CPLS)	[KP90], [KT92], [KST07], [Bus86a]
T^i_2, S^{i+1}_2	G_i, G^*_{i+1}	$\leq_{1-1}$(LLI$_i$)	[KP90], [KT92], [KNT11], [Bus86a]
PSA, U^1_2, W^1_1	QBF	PSPACE	[Dow78, Dow79], [Bus86a], [Ske04]
V^1_2	**	EXP	[Bus86a]
VNC^1	Frege (F)	ALOGTIME	[CT92], [Ara00]; [CM05], [CN10]
VL	GL*	L	[Zam97], [Per05], [CN10]
VNL	GNL*	NL	[CK04], [Per09], [CN10]

Figure 6.1 Cook translations from formal theories to propositional proof systems. PV and PSA are equational theories; S^i_2 and T^i_2 are first-order theories; U^1_2, V^1_2, VNC^1, VL, VNL, and VPV are second-order theories; W^1_1 is a third-order theory. F and EF are the Frege and extended Frege proof systems; G_i and QBF are quantified propositional proof systems. Starred (*) propositional systems are tree-like. PLS is *Polynomial local search*, [JPY88]; CPLS is *Colored PLS*, [ST11]; LLI is *Linear local improvement* [KNT11], [BB14]; and $\leq_{1-1}$ is many-one reducibility for TFNP functions [Pap94a].

elegant intuitionistic type theory called PV^{ω} corresponding to polynomial time computability.

Acknowledgments

I thank Bruce M. Kapron, Jan Krajíček, and Sanjiva Prasad for their comments on drafts of this paper.

7 Towards a Complexity Theory of Parallel Computation

Paul Beame and Pierre McKenzie

7.1 First Words

This chapter covers Steve Cook's "parallel complexity years." It adopts an historical perspective: first, the context leading to his interest in the study of parallel computation, then the steps of a theory in the making with the many actors involved, followed by the refinements and maturing of the theory. This culminates with an overview of his other work in the field and his two influential survey papers on parallel complexity, one of which is included in this volume.

7.2 The Early Years

In his seminal work characterizing computation, Alan Turing [Tur36] used the paradigm of a computer as a single intelligence, with a single focus of attention and action. However, in building electronic realizations of digital computing devices to implement this paradigm, aspects of parallelism soon arose in a natural way: Separate hardware components were needed to handle individual parts of the process of computation and, once they became available, it was natural to consider overlapping their activation rather than activating them only one at a time. This idea of parallel execution, which later became known as *instruction-level* parallelism [RF93], was already part of Turing's later proposal for Pilot ACE [CD86] and discussed in detail by Maurice Wilkes [Wil51].

On the theoretical side, parallelism on this modest scale became a natural part of the analysis of computation. For example, the proof that any recursively enumerable language whose complement is also recursively enumerable must also

be decidable naturally involves running two computations, one for the language and one for its complement, in parallel. Moreover, more pointedly, in their seminal development of complexity theory of sequential computation in the early 1960s, Juris Hartmanis and Richard Stearns [HS64, HS65] used *multi-tape* Turing machines with some (small) constant number of tapes as the standard model for measuring time and space complexity.

Parallelism on a much larger scale had already been a feature of human computation for some time—rooms full of people had been employed to perform numerical or code-breaking calculations for actuarial, scientific, or military applications. However, the idea of building a digital computer comprised of a large number of processors was not widely suggested until the late 1950s. In his 1958 survey of the modest amount of parallelism that had been used in digital computers, Stanley Gill [Gil58] argued that the benefits of using parallelism on a much larger scale would outweigh the difficulties of its implementation, an opinion which he indicated his peers did not share. Around the same time, John Cocke and Daniel Slotnick [CS58] analyzed the impact of using a highly parallel computer, organized as an array of processing elements, for numerical computations. In the early to mid-1960s, Slotnick led the design of highly parallel computers along these lines [Slo82], first on the SOLOMON project at Westinghouse [SBM63], and then that of ILLIAC IV whose fundamental design was widely known by the mid-1960s though the full design was only complete and released in 1968 [BBK+68].

In 1966, Michael Flynn produced his widely used taxonomy of parallel systems [Fly66] that distinguished the Single-Instruction-Multiple-Data (SIMD) approach, used in the SOLOMON and ILLIAC IV designs, from the Multiple-Instruction-Multiple-Data (MIMD) approach that had been used on a very small scale in the designs of two and four processor systems such as LARC [ECTS59] and had also been suggested as a basis for large-scale parallel computation [LM64].

Though the SOLOMON project was abandoned and the actual completion of the ILLIAC IV took the better part of a decade, the release of the ILLIAC IV design and the anticipation for its completion captured the imagination of many researchers. These included researchers developing algorithms to take advantage of parallel computation, along with computer architects and programming language researchers. Highly parallel algorithms for numerical [Nie64, CW67, She67], algebraic [KMW67, Pea67, Pea68], and data manipulation [Bat68] problems began to be published with regularity in the most widely read publications in the field like the *Communications of the ACM* and the *Journal of the ACM*.

However, the relatively young field of computational complexity was far from ready to tackle the question of large-scale parallelism. For example, in their

1971 survey of the theory in the *Journal of the ACM* [HH71], Juris Hartmanis and John Hopcroft only briefly mention parallelism in the context of the small-scale instruction-level parallelism discussed above.[1]

Meanwhile, in parallel programming and system design, a number of distinctly different ways of structuring and thinking about parallel and distributed computations were being developed, such as Petri nets [Pet62], fork-and-join parallelism [Con63], and vector addition systems [KM69].

7.3 The Beginnings of a Theory

The first model that began to tackle the question of the computational complexity of parallel computation was given in a paper by Vaughan Pratt, Michael Rabin, and Larry Stockmeyer[2] [PRS74], presented at the annual ACM STOC conference in the spring of 1974 prior to the completion of the ILLIAC IV. Pratt et al. considered the question of augmenting the usual model of random-access machines (RAMs) with integer registers with extra vector registers each capable of holding a vector of bits on which operations could be performed at unit cost. These unit-cost operations included bit-wise parallel Boolean operations on pairs of bit vectors as well as indexing and shift (left and right) operations that combined vector and integer registers, yielding a SIMD model of parallelism.

Their paper began by giving examples of the dramatic speed-up that such a model could yield, for example, satisfiability of CNF (Conjunctive Normal Form) formulas in linear time. This algorithm used bit-vectors of exponential length in the input size and the authors noted that, though in general it was unclear whether bit-vectors of more than exponential length could lead to an even more powerful model, they could not rule out the possibility.

Using ideas from the quadratic simulation of nondeterministic space by deterministic space shown by Cook's Ph.D. student Walter Savitch [Sav69, Sav70], they proved that, for decision problems, and with reasonable functions used as time and space bounds, their vector machines could simulate nondeterministic sequential machines that use space $S(n)$ in deterministic parallel time roughly $O(S^2(n))$, and, in turn, a nondeterministic version of their vector machines running in time $T(n)$ could similarly be simulated in deterministic space $O(T^2(n))$. This equivalence, up to a polynomial, of time on their vector machines and space on ordinary sequential machines would become a theme of research on parallel computing over the next several years.

1. The survey appeared only months prior to Cook's 1971 paper on NP-completeness.

2. The 1976 journal version of the paper [PS76] did not include Rabin as a co-author.

They asked

> Can one always obtain a "polynomial in log" time improvement by going from serial to parallel computation? If we equate vector machines with parallel computation then this question is equivalent to an open question concerning the "DTIME vs SPACE" relation for Turing Machines. Of course, there is no reason to suppose that vector machines are the most powerful possible forms of parallel computers, even to within a polynomial.

In a follow-on paper presented at the IEEE Switching and Automata Theory conference (which would become the IEEE Symposium on Foundations of Computer Science (FOCS)) later that fall, Hartmanis and Janos Simon [HS74] showed that one could eliminate the idiosyncrasy of the segregated variable types of Pratt et al.'s vector machine model and use unit-cost multiplication[3] on unbounded registers in an ordinary RAM model to obtain an equivalent model (up to a polynomial).

By this time, the completed (quarter of the) ILLIAC IV was finally in operation, the Cray-1 [Rus78] was in development, and there were research projects, such as the C.mmp [WB72], focused on developing other modes of parallel computation and alternative interconnection networks [Sto71]. Algorithmic developments included the first parallel graph algorithms [LK72] and general methods for parallelizing arithmetic expressions [Bre74].

From a complexity-theoretic point of view, the next major milestone was the independent work of Dexter Kozen [Koz76] and of Ashok Chandra and Stockmeyer [CS76] presented at the 1976 IEEE FOCS conference that produced beautiful and deeper connections between sequential space complexity, alternating Turing machines, and parallel computation. One natural mechanistic view of the operation of a nondeterministic computation involves a parallel execution over all possible branches that the algorithm may take on a given input. With this parallel view, there is no necessity that the combination rule for these parallel branches be the existence of an accepting branch among them. One could instead have the universal requirement that all of them accept. Both papers considered this parallel interpretation. Kozen, whose work also seems independent of that of Pratt et al. [PRS74], explicitly drew a direct correspondence between this kind of parallelism and the parallelism created through the fork operation and the subsequent join operation that brings the parallel executions back together again, whereas Chandra and Stockmeyer used the model to show equivalence of two seemingly different models of parallelism.

3. It is worth noting that the time-bounded RAM model of Cook and Robert Reckhow [CR73] had deliberately excluded multiplication as an allowable unit-cost operation.

An *alternating* Turing machine (ATM) allows for both of existential and universal possibilities in a single machine: some states are designated existential states and other states are designated universal states. Both of these kinds of states cause a fork and the creation of (two) child processes: For a universal state, the associated join requires both child processes to have accepted, for an existential state, at the join only one of the two child processes needs to have accepted. Chandra, Kozen, and Stockmeyer showed how complexity classes like PSPACE and levels of the polynomial-time hierarchy (recently defined by Stockmeyer [Sto75]) had direct natural characterizations using ATMs: Time on this ATM model captures both deterministic and nondeterministic sequential space up to a quadratic amount.[4] While Pratt et al. had been somewhat hesitant about the universality of the loose connection they had found, Chandra and Stockmeyer explicitly formulated a hypothesis that Leslie Goldschlager, in his 1977 Ph.D. thesis [Gol77] written under Cook's supervision termed the *Parallel Computation Thesis*:

Parallel time is polynomially equivalent to sequential space.

7.4 Development and Issues with the Theory

The outlines of a model of parallel computation seemed to be emerging nicely, though none of the models felt especially natural as a parallel computing device. A series of papers developed models designed to be the truly parallel analogues of the sequential RAM. Savitch and Michael Stimson [SS76] showed how the model of Pratt et al. [PRS74] is equivalent to a parallel RAM model, the *k*-PRAM, that had an explicit call/return, somewhat like a fork/join of processes except that the processors are ready and waiting to be activated at the beginning and each processor can activate up to *k* other processors with explicit calling parameters and return values. In this model, each processor has its own collection of registers. Like Pratt et al., Savitch and Stimson emphasized the fact that, unlike the sequential models where the question is still open, the addition of nondeterminism to the model affects the parallel time by at most a polynomial amount.

Two papers presented at the ACM STOC conference in the spring of 1978 (submitted in December 1977), one by Goldschlager [Gol78] and another by Steven Fortune and James Wyllie [FW78], suggested a different approach to parallel RAMs—one that involves synchronous processors using global shared memory to communicate. In Goldschlager's model, which he called a *SIMDAG* (SIMD and global memory), each processor has an individual processor number that allows for the

4. And alternating space is equivalent to exponentially larger deterministic time, a fact that we will revisit later.

centrally broadcast instructions to have different effects for each processor. In Fortune and Wyllie's model, which they called the *P-RAM*, new processors are activated by a fork operation and each processor has its own, possibly different, program counter in the common shared program; it also allows for local memory in addition to global shared memory.

Both models allow concurrent read access to the shared global memory, but their choices of write access differed: Goldschlager's SIMDAG gave a concurrent write tie-breaking rule in which the lowest-numbered processor writing to a location succeeds. Fortune and Wyllie's P-RAM assumed immediate halt and rejection if the execution causes any concurrent writes to be attempted. Fortune and Wyllie's name P-RAM with the same pronunciation, but without the hyphen, came to be adopted subsequently for the refinement of both models, with Fortune and Wyllie's model being the basis for the concurrent-read exclusive-write (CREW) PRAM and Goldschlager's SIMDAG becoming the concurrent-read concurrent-write (CRCW) PRAM.

Both papers emphasized the same connections between parallel time and sequential space discussed in earlier papers, though the emphasis of the two papers was quite different: Fortune and Wyllie also showed that the nondeterministic $O(\log n)$ P-RAM time is sufficient to simulate NP and described how the parallel simulations of PSPACE can be executed using only polynomial-size global memory. Goldschlager focused on the constructibility of his SIMDAGs, as well as a very general notion of parallel computation based on what he called *conglomerates*, interconnected computing units inspired by VLSI circuits, out of which he showed one could build SIMDAGs with a suitable instruction cost model, and which he argued also satisfy the Parallel Computation Thesis.

An alternative approach to parallel computation was initiated somewhat earlier in the work of Allan Borodin [Bor77] who focused on circuits as models of parallel computation. Since circuits are an inherently nonuniform model of computation—one needs a family of circuits with a separate circuit for each input size—in order to connect the model to machine-based models of computation, Borodin needed to discuss how efficiently the circuit for inputs of size n can be constructed as a function of n. Borodin's paper, "On relating time and space to size and depth," which was submitted to *SIAM Journal on Computing* in March 1976, connects circuits to space-bounded sequential computations and hence requires that n-input circuits of depth $d(n)$ be constructible by Turing machines using space $O(d(n))$. The key idea that enables the simulation of nondeterministic space $O(S(n))$ by circuit depth $O(S^2(n))$ is a *uniform* circuit of depth $O(\log^2 N)$ (and size $N^{O(1)}$) for Boolean $N \times N$ matrix powering, and hence transitive closure on directed graphs of

size N, which allows the circuit to simulate reachability in the configuration graph of the space-bounded computation.

Borodin asked a fundamental question about the relationship between *simultaneous* complexity bounds in the sequential and parallel models: He noted that in addition to the simulation of space $S(n)$ by depth $O(S^2(n))$, prior work by Nicholas (Nick) Pippenger and Michael Fischer [FP74] had given a very tight separate simulation of sequential time by circuit size (time $T(n)$ can be simulated by circuit size $O(T(n) \log T(n))$). He asked whether these simulations could be combined to give a single simulation[5] that would take a sequential algorithm that runs in time $T(n)$ using space $S(n)$ and obtain a circuit of size $T(n)^{O(1)}$ and depth $S(n)^{O(1)}$. This question ended up being critical in the later development of the theory.

Borodin also introduced a nonuniform notion of *depth-completeness* for polynomial size circuits, in analogy with log-space completeness for P. Because the best circuit bound available even for simulating deterministic log-space used $O(\log^2 n)$ depth, he chose many-one reductions that are computable by $\log^{O(1)} n$ (polylogarithmic) depth circuits. This notion of reduction did not preserve polynomial size, so in this definition of completeness it was necessary to add a separate extra constraint that the resulting function could also be shown to be computable by polynomial-size circuits.

This anomaly in the definition of reduction is the first hint of an issue with the consensus in the definitions of parallel time complexity. In all of these papers, as was the case in the earlier work on vector machine and ATM models, as the parallel time $T(n)$ or circuit depth $d(n)$ grows to $\omega(\log n)$, problems computable in this parallel time or depth are not necessarily computable in polynomial time since the amount of hardware available after $T(n)$ steps or depth in these models potentially grows exponentially with $T(n)$.

7.5 Steve's Class and Nick's Class

Space-bounded computation and its relationship to sequential computation time had long been a focus of interest for Cook[6]; even before his work on NP-completeness, in a 1969 paper he had raised the question of whether all of P can be computed in logspace [Coo69]. This question, expanded to the larger question of whether everything in P can be computed in $\log^{O(1)} n$ space, was a key motivator

5. This is Open Question 2 in the paper, though there is a typographical error in which what was obviously intended to be an S shows up as T.

6. See Chapter 8 in the current volume by N. Pippenger for more detail.

for the 1973 paper in which he had introduced the notion of log-space completeness for P and shown that a problem on path systems is log-space complete for P [Coo73b].

As Cook's first Ph.D. student, Walter Savitch had proved the simulation of nondeterministic algorithms using space S by deterministic algorithms using space $O(S^2)$ and, in particular, that transitive closure is computable using $O(\log^2 n)$ space [Sav69].

While Savitch's theorem showed that $O(\log^2 n)$ space is sufficient, the algorithm that achieves this requires $n^{\Theta(\log n)}$ time, which is only quasipolynomial rather than polynomial. Cook asked whether it is possible to find an algorithm that *simultaneously* operates in polynomial time and only uses $O(\log^2 n)$ or $\log^{O(1)} n$ space.

Work on the Parallel Computation Thesis had largely been expressed as refinements of the relationship given by Savitch's theorem. This had been part of Cook's interest in advising Goldschlager in his work on parallel computation, but Cook was not yet focusing on these questions himself.

In January 1978, Nick Pippenger began a half-year as a research visitor at the University of Toronto. During that spring semester, he taught a graduate research course on parallel communication and the theory of switching in telephone networks. Telephone switching networks were modeled as bounded-degree graphs in which certain nodes are designated as *external*, representing the potential senders/receivers, and the remaining nodes, representing the switches, are *internal*. The overall goal is to design switching networks that let one choose short node-disjoint routes through the network between as many simultaneous arbitrarily chosen sender–receiver pairs as possible among the external nodes while using as little hardware as possible for the internal nodes in the network. Examples of such networks include Clos–Benes networks [Clo53, Ben65] in which n senders can be simultaneously connected to n receivers via edge-disjoint paths in a graph with $\log n$ layers, each of n nodes, each of which pairs up two edges of one layer with two edges of the next layer in one of two ways.

In the telephone switching theory, switches had no separate computational power in themselves and were controlled via an external algorithm that set the switches in order to match up the designated sender–receiver pairs. Pippenger had begun to develop simple algorithms to set the switches and began to consider whether it would be possible to compute the positions of the switches inside the network itself in order to make the required sender–receiver connections [Pip]. To make this useful, one would want the algorithm to run quickly and not use too much network hardware. For example, it was possible to build smarter networks to do this internally computed setting of switches by using sorting networks due to Kenneth (Ken) Batcher [Bat68], which have depth of $O(\log^2 n)$ and $O(n \log^2 n)$

hardware. What kinds of computations should one allow? When he began thinking about the question, Pippenger originally focused on computations of $\log^{O(1)} n$ (polylog) time and $n \log^{O(1)} n$ (quasilinear) hardware, in keeping with the hardware limitations typically considered in the telephone switching models [Pip].

However, there were certain algorithms for networks that Pippenger wanted to use that involved operations like reachability or transitive closure, for which it seemed that quasilinear hardware would not be enough and something closer to n^3 size might be necessary. He realized that relaxing the hardware bound to any polynomial would probably be necessary to get a general enough model [Pip].

Cook and Borodin regularly attended Pippenger's class. After class, the three of them would often discuss research and the material from the class over tea in the large spartan lounge of the McLennan Physics building at the University of Toronto that was the temporary home of part of the computer science department.[7] While it was an easy place to socialize, the environment in the lounge was not a conducive place for writing out details for all of them to see. It was in this environment that Pippenger raised the questions of what computations were possible using polylog parallel time and polynomial hardware simultaneously.

To all of them, this immediately seemed a very interesting class of problems worth exploring. It also nicely cleaned up the concern from Borodin's earlier paper that polynomial-size circuits and polynomial-time are not necessarily closed under polylog depth reductions; reductions using these simultaneous resource bounds (with suitable notions of uniformity) would preserve both. Borodin's work had shown that $O(\log n)$ sequential space could be simulated by $O(\log n)$-space uniform circuits of polynomial size and $O(\log^2 n)$ depth. Borodin and Cook suggested that, more generally, the right notion of uniformity for this new class would be constructibility in $O(\log n)$ space.

Their discussions quickly led to the puzzle of transitive closure. While Cook had been focused on the simultaneous sequential complexity of the problem "Is transitive closure simultaneously computable in polynomial time and polylog space?" in the parallel case, the $O(\log^2 n)$ time/depth algorithm for transitive closure requires only polynomial hardware and hence is in the class of problems solvable by the kinds of algorithms that Pippenger had suggested.

There were now two related classes involving simultaneous resources, one sequential and one parallel. What natural problems might be in these two classes? What are the relationships between them and their associated parameters? As noted above, in his 1977 paper Borodin had already explicitly asked the dual

7. A fire the previous spring had destroyed the building that had previously housed the department.

question, in general form, of whether there was containment of the sequential class in the parallel one [Bor77]. Transitive closure was a good candidate for a separation between the two classes.

Transitive closure was not the only natural candidate. After hearing a talk at Toronto by Janos Simon on simulations of probabilistic space-bounded machines by deterministic machines with polynomially larger space bounds that was later presented at FOCS 1978 [SGH78], Borodin, Cook, and Pippenger were able to improve the simulation for probabilistic machines to match the bounds of Savitch's Theorem. Moreover, using a redundant representation of numbers, they were able to extend parallel algorithms for transitive closure to an analogous closure for stochastic matrices and hence show that probabilistic $O(\log n)$ space can be simulated by polynomial-size circuits of $O(\log^2 n)$ depth.

What about other interesting problems already known to be separately solvable in polylog space or polynomial time? For example, was context-free language (CFL) recognition in either of these simultaneous classes? (More than a decade prior, Philip Lewis, Hartmanis, and Stearns had shown that every CFL can be decided using only $O(\log^2 n)$ space [LSH65], matching the later bound for transitive closure and complementing the earlier polynomial-time algorithm of Cocke, Kasami, and Younger [Kas66, You67].) CFL recognition was a particularly natural next problem to consider for simultaneous bound beyond transitive closure since I. H. (Hal) Sudborough [Sud75] had shown that transitive closure is log-space reducible to CFL recognition and hence in the class LOGCFL that consists of such problems.

As they discussed these questions aloud, they needed a shorthand description to enable the free flow of conversation. There is some variance in their recollections of how these shorthand descriptions came to be. In Borodin's recollection, Cook began to refer to Pippenger's new simultaneous parallel complexity class as "Nick's class." Cook liked to refer to the simultaneous sequential complexity class as PLOPS for "Polynomial-time PolyLOg Space," though this was not a name that seemed desirable to pronounce[8]; over time, Pippenger, in response to Cook's shorthand for his newly defined simultaneous parallel complexity class, began to refer to the sequential one as "Steve's class." However, in Pippenger's recollection, Borodin had been the first to use both names.

Over the spring and summer of 1978, Pippenger continued to develop the theory of this new complexity class and began relating it to other properties of sequential

8. In his paper first defining the class, Cook used both PLOPS and the more mellifluous PLOSS to stand for its subclass of "Polynomial-time LOg Squared Space" [Coo79].

computation. Due to the peculiarities of the conference-focused publication ethos in theoretical computer science, he was not the first to discuss the class in print and by then it had already been named after him.

The submission deadline for the 1978 IEEE FOCS conference that fall had long passed and the next natural venue to present the work would have been the 1979 ACM STOC conference. However, Pippenger was one of the eight members of the program committee and, in keeping with a longstanding policy that has continued for decades, as a member of the program committee member he could not submit anything to the conference. As a result, his paper on the subject of his new complexity class did not appear in print until the fall of 1979 at the FOCS conference [Pip79].

Meanwhile, Cook continued to work on the problem of simultaneous complexity bounds for CFL recognition, eventually developing an algorithm for the recognition problem for the special case of *deterministic* context-free languages (DCFLs) that simultaneously ran in polynomial time and $O(\log^2 n)$ space [Coo78a]. An examination of the algorithm showed that it was also in Nick's class; moreover, it required only $O(\log^2 n)$ circuit depth and polynomial size, avoiding the general loss in parameters in going from sequential space to parallel depth (unlike the simulation of $O(\log n)$ space by circuits). Cook's paper at the 1979 STOC conference [Coo79] included a short paragraph mentioning Nick's Class and its definition, using the notation NC, together with its containment of the problem of DCFL recognition, which became the first description of the complexity class in the published literature.

Pippenger's work on these new complexity classes eventually appeared in his 1979 FOCS paper, "On simultaneous resource bounds." There, Pippenger pointed out the weakness of simulations between sequential and parallel models, including those previously described by Borodin, that

> bound one resource in the simulating realm as a function of the corresponding resource in the simulated realm, but...allow the other resource in the simulating realm to grow beyond any interesting bound, even if there is a bound on the other corresponding resource in the simulated domain.

His paper focused on the question that had been raised by Borodin of the relationship between the two complexity classes, the simultaneous sequential one defined by time and space bounds and the simultaneous parallel one defined by $O(\log n)$-space uniform circuit size and depth. Could one prove a simulation in either direction? He did indeed prove that there were precise characterizations of each of these

simultaneous classes by simultaneous classes involving the other model, but not of a form that answered either of these questions.

In particular, Pippenger showed that (1) uniform polynomial size and polylogarithmic depth circuits were equivalent to Turing machines running in polynomial time and polylogarithmic *reversals* (change in direction of head movement) and (2) Turing machines running in polynomial time and polylogarithmic space were equivalent to uniform polynomial size circuits of polylogarithmic *width*. However, his paper makes no mention of these specific bounds and their associated complexity classes and only discusses the simulations in terms of arbitrary time, space, size, and depth bounds. The only reference to Cook's DCFL paper is to the notion of $O(\log n)$-space uniformity for circuits. In fact, it is only at the very end of the paper that Pippenger mentions that the circuit models he had been considering are models of parallel computation, and then only in the context of a discussion of subsequent work. To discuss that subsequent work, we first step back to the spring of 1979.

Cook's paper on DCFLs was not the only paper at the 1979 STOC conference that considered questions concerning CFLs and models with simultaneous resource bounds. The other was by Walter L. (Larry) Ruzzo [Ruz79b], who was motivated by the CFL recognition algorithms in the ATM model. As noted earlier, in addition to suggesting ATM time as a measure for parallel computation, Chandra, Kozen, and Stockmeyer had shown that deterministic polynomial-time corresponds to languages decided by ATMs using $O(\log n)$ space. Ruzzo observed that CFL recognition could be implemented easily as an $O(\log n)$ space ATM algorithm but that this implementation did not seem to require the full unrestricted power of $O(\log n)$-space ATMs; it required only a size $O(n \log n)$ *accepting tree*, a tree of configurations representing the part of the computation on an input required to prove that the input is accepted. (For a nondeterministic TM, this tree is simply a path corresponding to an acception computation, but for an ATM the universal branches require branching. In general, such an accepting tree can have many repeated configurations and the result for ATM space $O(\log n)$ could be as large as $2^{n^{O(1)}}$.)

Based on this, Ruzzo considered a new resource, the (accepting) *tree-size* of an ATM and focused on complexity classes associated with ATMs that have simultaneous space and tree-size bounds.[9] Ruzzo also showed that with the additional tree-size bound, ATM space can be simulated using small parallel time; in particular,

9. In particular, Ruzzo related this restriction to a model called *auxiliary pushdown automata* (AuxPDAs) introduced by Cook in 1969 [Coo69, Coo71a] and used by Sudborough [Sud78] to characterize LOGCFL, showing that it exactly corresponds to languages recognized by ATMs with $O(\log n)$ space and polynomial tree-size.

ATMs with $O(\log n)$ space and polynomial tree-size can be simulated by ATMs using only $O(\log^2 n)$ time and hence $O(\log^2 n)$ time vector machines.

Ruzzo learned of Pippenger's results and had discussions with Borodin, Cook, and Pippenger during the STOC conference. During these conversions, Cook speculated that, while DCFL was in NC, CFL recognition would not be. Shortly afterward, Ruzzo realized that, by a simple modification, he could improve his simulation of $O(\log n)$-space polynomial tree-size ATMs by ATMs with $O(\log^2 n)$ time while still keeping the $O(\log n)$ space bound. Moreover, he realized that NC could be characterized by ATMs that simultaneously had $O(\log n)$ space and $\log^{O(1)} n$ time. Together, these showed that general CFL recognition is actually in NC, contradicting Cook's speculation.

With its multiple alternative characterizations, Ruzzo's 1979 FOCS paper [Ruz79a] containing these results provided another argument for the robustness of NC using ATMs bounded in both space and time. In particular, Ruzzo showed that for polynomial-size circuits of depth $\geq \log^2 n$, Borodin and Cook's $O(\log n)$-space constructibility definition yields the same notion as time-bounded $O(\log n)$-space ATMs, but that at smaller depths that are not known to simulate $O(\log n)$ space, this is unclear. On the other hand, with a stronger uniformity notion,[10] circuit depth for polynomial-size circuits and ATM time for $O(\log n)$-space machines yield precisely the same complexity classes for all depths. Ruzzo's paper also defined the notation NC^k to refer to the subclass of NC consisting of those problems solvable by polynomial size circuits of $O(\log^k n)$ depth.[11]

While these results tied together the class NC nicely in a mathematical sense, there are drawbacks in the models of computing devices themselves in expressing the resources of parallel hardware and parallel time. ATMs with bounded accepting tree size are largely in the realm of a thought experiment. Combinational circuits are closer to modelling real devices, but they mix the two resources since circuit elements are each associated with only one time step. PRAMs/SIMDAGs with their shared memory and simultaneous reads and writes to the same location have a potentially unreasonable power to have fully reconfigurable communication between processors on a per-step basis.

Cook and his Ph.D. student Patrick Dymond, whom he co-advised together with Borodin and Charles Rackoff, took up the question of more precisely modeling these two resources. Their work gave further evidence that NC is fair in representing efficiently parallelizable problems.

10. Termed U_{E^*} uniformity.

11. The terminology of SC for "Steve's Class" had not yet made an appearance in print. Ruzzo's FOCS paper still referred to it as PLOPS.

In order to avoid the mixing of parallel time and hardware in combinational circuits, they introduced a notion of synchronous sequential circuits with random access to the inputs, which they termed "aggregates,"[12] that reuses the same gates with fixed connectivity at every step.[13] Aggregates have the advantage of allowing one to consider sublinear hardware and the possibility of having the product of parallel hardware and parallel time be as small as the sequential time, a property that would later be termed a *work-optimal* algorithm (see, e.g., JáJá [JáJ92]). Dymond and Cook showed that aggregate hardware has a natural correspondence with the width of (synchronous[14]) combinational circuits.

Alternatively, could one make the PRAM/SIMDAG model more reasonable? For this, Dymond and Cook developed a model with communication between parallel processing elements that is reconfigurable but in a much more limited way than in the PRAM/SIMDAG. The general idea is that each processor can activate a new processor or get information from one of only a constant number of other neighbor processors per step, and can only change this list of neighbor processors by adding a newly activated processor or by replacing one by a processor on one of its neighbors' lists of neighbor processors. In their 1980 STOC paper, Dymond and Cook called such machines Hardware Modification Machines (HMMs), as they were an extension of the Storage Modification Machines (SMMs) introduced by Arnold Schönhage in 1970 to model flexible storage and input access in sequential computation [Sch70],[15] but in the much later journal publication of this work, Cook and Dymond [CD93] used the more natural name Parallel Pointer Machines (PPMs). Using what has later been termed "pointer jumping"[16] and prefix computation [LF80], Dymond and Cook showed that their simpler HMMs/PPMs still had all the good properties of the PRAM/SIMDAG models with respect to efficient simulation of sequential space. In particular,

$$\text{HMM/PPM } \textit{time } S \subseteq \text{Turing machine } \textit{space } S^2 \subseteq \text{HMM/PPM } \textit{time } S^2.$$

12. A natural finite version of Goldschlager's notion of conglomerates.

13. The random access to the inputs is achieved by having certain groups of circuit gates associated with input indices and a corresponding returned input bit that is available $\log n$ steps later, which accounts for the delay of the fan-in tree that would be needed to implement it.

14. Those with all paths to a gate from the inputs having the same length.

15. Cook had developed a notion similar to SMMs in his 1966 Ph.D. thesis [Coo66a].

16. Pointer jumping is the operation of taking an out-degree 1 directed graph represented by a function p on vertices that defines the unique out-edge from each vertex v as $(v, p(v))$, and producing the new graph for which $p(v)$ is replaced by $p(p(v))$.

Finally, Dymond and Cook showed that one can characterize *both* NC and SC using either uniform aggregates or HMMs/PPMs, with the former characterized by polynomial hardware and polylogarithmic parallel time and the latter characterized by polynomial parallel time and polylogarithmic hardware.

7.6 Cook's Surveys of Parallel Computation

Over the next few years, Cook concentrated on parallel computation and on mapping out directions for the development of its complexity theory. Two survey papers are major milestones in that development. Cook's first survey paper

"Towards a complexity theory of synchronous parallel computation"

was the outcome of a lecture delivered in early 1980 at a *Logic and Algorithms Symposium* in honor of mathematician Ernst Specker. Cook's theory of *synchronous*[17] parallel computing intertwined nicely with the study of sequential computation, as witnessed by the parallel computation thesis and by the tantalizing issues raised by the works of Borodin, Cook, Pippenger, and Ruzzo on simultaneous resource bounds. Cook's survey brought together many of the results for these models that had been shown over the preceding half dozen years. It gave many of the details of the new models of aggregates and mentioned HMMs/PPMs and their associated characterizations of simultaneous complexity classes NC and SC. The properties of the aggregate and HMM/PPM models gave further justification in working with NC.

Though the focus of this survey was on the different models of parallel computation and the connections between them, the primary impact of the survey was as a convergence of ideas, particularly on the importance of questions of simultaneous resource bounds in general, and NC and the question of NC versus SC more specifically. Given that these subjects had attracted the attention of one of the leading researchers in all of theoretical computer science, the survey provided a spark for a growing group of researchers to turn to the study of parallel computation and NC algorithms, a subject that was to become one of the dominant directions in theoretical computer science in the ensuing decade.

Complexity-theoretic and algorithmic aspects of parallel computation were central to Cook's research during this time. He supervised Romas Aleliunas' Ph.D. research on randomized fast parallel routing in bounded-degree networks that

17. Excluding computation by processes that differed widely in speed or localization and thus required a significant emphasis on routing and clocks and the study of race conditions that was more in the realm of "distributed computing."

would let such architectures simulate NC algorithms efficiently [Ale82]. With Cynthia Dwork and Rudiger Reischuk, he produced the first lower bounds for the PRAM model, proving that the CREW PRAM requires $\Theta(\log n)$ steps even to compute simple functions such as the OR of n bits [CD82, CDR86], and hence PRAMs do not always provide asymptotic speed-up relative to Boolean circuit depth.

Borodin, Cook, and Pippenger [BCP83] finally published their probabilistic analogue of Savitch's Theorem and their parallel algorithm for stochastic closure from 1978 which showed that probabilistic $O(\log n)$ space is contained in NC^2. In doing so, they simplified their algorithm and generalized their results to a broad class of objects, which they termed "well-endowed" rings, ones for which addition and multiplication can be particularly efficiently computed (in NC^0 and NC^1, respectively).

Cook also worked with each of us, as his Ph.D. students, on developing algorithms for parallel computation. With McKenzie, Cook wondered whether permutation group problems recently shown solvable in polynomial time could be placed in NC. They developed NC^3 algorithms for problems such as testing membership in an Abelian group specified by generating permutations [MC87]. Permutation groups provided rare examples of natural computational problems[18] in NC but apparently outside NC^2. Cook and McKenzie further identified, as a by-product of their work on permutation groups, a list of problems [CM87] complete for the class $\mathsf{L} = \mathsf{DSPACE}(\log n)$, thus located well inside NC^2 but likely outside NC^1.

The space complexity of the basic arithmetic operations had interested Cook for some time; integer addition and multiplication were easily shown to be computable in $O(\log n)$ space, but division seemed much harder. One motivation was to be able to discuss a representation-independent notion of log-space computable integer functions as a refinement of Cobham's representation-independent notion of polynomial time complexity [Cob65]. In his Ph.D. thesis, Cook had shown how to use Newton iteration to compute integer division in $O(\log^2 n)$ space. From the new perspective of parallel computation, this became an NC^2 algorithm. Cook's graduate student, H. James (Jim) Hoover, showed an alternative direct approach based on reducing the problem to an NC^2 algorithm for efficiently computing the n-th power of an n-bit integer [Hoo79]. Inspired by a faster algorithm for division by John Reif [Rei83] based on a self-reduction using the fast Fourier transform, Cook, together with Beame and Hoover, first developed an $O(\log n \log^* n)$-depth NC algorithm for division based on the Chinese Remainder Theorem (CRT) and

18. Much effort over the period 1983–1987 culminated in an intricate NC algorithm by László Babai, Eugene Luks, and Ákos Seress [BLS87] that relied on the massive classification of finite simple groups to test membership in general permutation groups.

self-reduction; then Beame, Cook, and Hoover applied ideas from Cook's work with McKenzie to eliminate the self-reduction component and achieve $O(\log n)$ depth circuits for integer division [BCH86], matching the circuit depth of the other arithmetic operations.[19]

For a keynote talk at the 1983 Foundations of Computing Theory conference, Cook produced his second major survey of parallel computation entitled "The classification of problems which have fast parallel algorithms" and modestly revised in journal form as

"A taxonomy of problems with fast parallel algorithms."

By the time of this survey, parallel algorithms had become a burgeoning field, expressed as circuits, PRAM algorithms, or algorithms for networks of processors. The number of problems for which good NC algorithms were known had grown tremendously. A survey of parallel algorithms by Uzi Vishkin [Vis83] from the same year catalogued a wide range of problems and problem areas for which very fast parallel algorithms had been found, which PRAM variants (EREW, CREW, or CRCW) were required to achieve them, and introduced a "Super-PRAM" model with even more powerful instructions.

Cook's survey focused on developing a larger picture among the results and on general methods for understanding the relationships between the parallel complexity of problems. This survey introduced and popularized some key terminology and raised important open questions; it has become one of Cook's most widely cited papers.

In addition to providing a clear presentation of the key circuit definitions for NC and NC^k, Cook's survey introduced several new organizing concepts and definitions. It introduced a particularly clean notion of uniform circuit reduction, NC^1-reducibility, in which to solve a problem A one can employ $O(\log n)$ depth circuits with the usual binary gates plus oracle gates that compute B at unit cost in circuit size and a cost of logarithmic depth in their number of inputs. Such circuits automatically have polynomial size. These reductions have the nice property that they preserve the levels of the NC hierarchy. Cook also defined several other complexity classes of functions using closure under NC^1 reductions.

This survey also introduced the terminology AC^k for functions computable by polynomial-size circuits of $O(\log^k n)$ depth that can use *unbounded fan-in* AND and

19. These circuits were somewhat uniform in that they were polynomial-time constructible but they were not known to be $O(\log n)$-space constructible; hence, they did not yet yield a representation-independent notion of log-space computable integer functions. That was implied by the later work of Chiu, Davida, and Litow [CDL01], which built on the CRT approach and was refined by Hesse, Allender, and Barrington [HAB02].

OR gates in addition to negations.[20] The study of nonuniform versions of such circuits of constant-depth (the case $k = 0$) had been popularized by the results of Merrick Furst, James Saxe, and Michael Sipser [FSS81] and by Chandra Stockmeyer, and Vishkin [CSV82], but there was no agreed-on terminology to describe the complexity classes that they define.[21] The latter's simulations, as well as uniform versions of those simulations by Ruzzo and Martin Tompa in unpublished work, showed that the AC^k complexity is equivalent to the SIMDAG (uniform CRCW PRAM) model with polynomially many processors running in $O(\log^k n)$ time.

Much of this survey is devoted to understanding the complexity of problems lying between NC^1 and NC^2, which is where many of the most interesting parallel algorithms seemed to sit. NC^1 lies in L, deterministic log-space, for which the best parallel simulations are no better than NC^2 algorithms, while NC^2 also contains NL, nondeterministic log-space, and its functional closure under NC^1 reductions, which Cook denoted NC^* and showed contains the minimum spanning tree problem. Moreover, he pointed out that a generic *parallel greedy* algorithm could be used to compute a minimum weight basis for an arbitrary matroid in NC^* given an oracle for matroid rank yielding the result for minimum spanning trees.[22]

Moreover, Cook observed that most natural problems known to be in NC^2 were NC^1-reducible to one of two natural problems: CFL recognition or integer determinant, each of which suffices to capture all of NL. By a trivial simulation, the AC^k class hierarchy interleaves the NC^k hierarchy, and AC^1 contains NL^* via the natural circuit for Boolean matrix powering. By an extension of the results of Ruzzo mentioned earlier, AC^1—and hence NC^2—includes the complexity class LOGCFL, as well as the potentially larger set of problems CFL^* that are NC^1 reducible to CFL recognition.[23]

20. The A in AC was intended to stand for *alternating* since such circuits naturally correspond to alternating bounded quantifiers.

21. At the time, Miklós Ajtai's work on these circuits was not yet widely known [Ajt83].

22. Carla Savage's 1977 Ph.D. thesis [Sav77] gave an algorithm for minimum spanning trees, along with a number of other graph problems, using $O(\log^2 n)$ parallel time and polynomial numbers of processors, which naturally translates to an NC^3 algorithm rather than an NC^2 algorithm because the model is a PRAM variant.

23. The survey predates the surprising results of Neil Immerman [Imm88] and Róbert Szelepcsényi [Sze88] showing that NL is closed under complement, and suggests that CFL^* is larger than LOGCFL since the latter seemed unlikely to contain the complement of NL. A later and beautiful characterization of LOGCFL due to H. Venkateswaran [Ven91], shows that the AC^1 algorithms for LOGCFL can be tightened to an equivalence of LOGCFL with SAC^1, the set functions computable by uniform $O(\log n)$ depth *semi-unbounded* fan-in circuits in which the fan-in of OR gates is unbounded and that of AND gates is bounded. Using inductive counting ideas similar to those used to show the closure of NL under complement, Cook together with Borodin, Dymond, Ruzzo,

Cook identified the importance of the class DET of functions NC^1 reducible to integer determinant and showed that it had several alternative characterizations and, via Cook's results with Borodin and Pippenger on the computation of stochastic closures [BCP83], included simulating $O(\log n)$ space-bounded probabilistic algorithms.

One of the major impacts of this survey was to point out the importance of the question of whether NC=FP; that is, whether every polynomial-time computable function[24] can be efficiently parallelized to yield an efficient polylogarithmic time parallel solution. This included highlighting results on problems that are NC^1-complete for FP, and hence unlikely to have such algorithms—a property that came to be termed *inherently sequential*. These included some of Goldschlager's results showing that restricted circuit value problems are complete for FP and especially his result with Ralph Shaw and John Staples that the maximum flow problem with large capacities is also complete for FP [GSS82]. Cook also gave a particularly surprising and simple example of such a hard problem: computing the lexicographically first maximal clique in an undirected graph, a problem computable by the most trivial of sequential greedy algorithms.

7.7 Last Words

In his 1981 survey paper, Cook wrote:

> We close this section with two little results about aggregates in the style "if horses can whistle then pigs can fly." This style (but not those results) comes from the paper of Karp and Lipton [KL]. [Coo81a]

That excerpt (but not those horses) is testimony to Cook's unpretentious approach to research. In earnest, Cook spent a half-dozen years studying parallel computation. But approaching the area with his characteristic rigor and method, he played a significant role in firming up a fledgling complexity theory of parallelism. Beyond the technical contributions due to him and reported in the present chapter, Cook raised a number of questions and directions for further study that helped guide the work of others. This led to many subsequent unexpected discoveries, such as the stream of complexity upper bounds described in footnote[23] and, to some extent, to Barrington's proof that simultaneous polynomial time and constant space captures NC^1 [Bar89]. The young at heart need not despair, though,

and Tompa subsequently showed that LOGCFL is indeed closed under complement and hence SAC^1 can also be defined in terms of circuits with bounded fan-in OR gates and unbounded fan-in AND gates [BCD+89a].

24. Since NC is defined as a set of functions, it is natural to relate it to the class of functions FP rather than the class P of decision problems.

as Cook also raised issues that remain unresolved 40 years later: Is randomness necessary for some problems to have fast parallel algorithms? Is integer determinant complete for NC^2? Are integer greatest common divisor or modular exponentiation either in (R)NC or complete for FP? Is $\mathsf{NC}=\mathsf{NC}^k$ for some integer k? And of course: Is NC=FP?

8 Computation with Limited Space

Nicholas Pippenger

8.1 Time and Space Bounds

Stephen Cook's paper "Pebbles and branching programs for tree evaluation" (with Pierre McKenzie, Dustin Wehr, Mark Braverman, and Rahul Santhanam) [CMW+12], reproduced in this volume, touches upon many themes that recapitulate Cook's research interests over his career. Two of these, "pebbles" and "branching programs," are clear from the title. But broader themes also recur in this paper. One of these concerns the general region of the map of complexity classes that is dealt with: the internal (or "fine structure") of polynomial time, P, and also small amounts of space, which we shall take to be polylogarithmic space $\mathsf{PL} = \bigcup_{k\geq 1} \mathrm{SPACE}\left((\log n)^k\right)$. In this chapter, we shall take up these themes in turn, giving the background to Cook's work, and outlining subsequent developments.

Although Cook is best known for inquiring after the relationship between P (polynomial time) and its superset NP (nondeterministic polynomial time), he devoted much of his attention to what lies between L (logarithmic space) and its superset P. In this survey, we shall meet many intermediate complexity classes. We'll also need to mention PL (polylogarithmic space). It is not known if either of P and PL is included in the other; but as we'll see below, it *is* known that they are not equal!

Our story begins with Walter Savitch, who did his Ph.D. under Cook at Berkeley. Savitch [Sav69, Sav70] relates L to NL (nondeterministic logarithmic space) just as Cook related P to NP: find a language complete for the larger class (with respect to a reducibility based on the smaller class); the two classes are then equal if and only if the complete language belongs to the smaller class. Savitch also showed

that a nondeterministic space-bounded machine can be simulated by deterministic machine whose space bound is the *square* of that of the simulated machine, so that $\mathsf{NL} \subseteq \text{SPACE}\left((\log n)^2\right)$. (And when PSPACE (polynomial space) was defined, there was no need to define NPSPACE as a separate complexity class, for the square of a polynomial is a polynomial.)

We have mentioned "reducibility," and this will be a good time to more fully describe some of the various reducibilities used in complexity theory. The notion of reducibility comes to us from recursive function theory, which has, among others $\leq_T$ (Turing reducibility: $A \leq_T B$ means "A is computable, given an oracle for B") and the stronger $\leq_m$ (many-one reducibility: "there is a computable function that transforms questions about membership in A to questions about membership in B having the same answer"). Turing reducibility is generally regarded as the weakest (and therefore most broadly applicable) notion of reducibility that makes sense when discussing computability; the advantage of using a stronger reducibility, such as many-one reducibility, is that it yields a finer (and thus more detailed) classification of languages. In complexity theory, we use the same types of reducibility but qualify "computable" by imposing resource bounds. Cook's work on P versus NP [Coo71b], reproduced in this volume, used $\leq_T^{\mathsf{P}}$, combining computability in polynomial time with Turing reducibility. Richard Karp's subsequent work [Kar72] used the stronger $\leq_m^{\mathsf{P}}$, combining computability in polynomial time with many-one reducibility. To discuss the relationship between L and P, a still stronger reducibility is needed. The relation $\leq_m^{\mathsf{L}}$, combining computability in logarithmic space with many-one reducibility, was used implicitly by Cook [Coo73b, Coo74] to exhibit a problem (Solvable Path System) complete for P. He presented this as evidence that this problem was not in L, or even PL. This result enabled Ronald Book [Boo76] to observe that, as stated above, $\mathsf{P} \neq \mathsf{PL}$ because the former class has a complete problem with respect to $\leq_m^{\mathsf{L}}$ while a complete language for the latter would violate the hierarchy theorem of Lewis, Hartmanis, and Stearns [SHL65]. Neil Jones and William Laaser [JL76] then used $\leq_m^{\mathsf{L}}$ to show other problems complete for P. Savitch [Sav70] used $\leq_m^{\mathsf{L}}$ to show the problem Threadable Mazes is complete for NL; Jones, Y. Edmund Lien, and Laaser [JLL76] then used it to show other problems complete for NL.

The next instalment of our story concerns Cook's work on *characterizing* P. The class P was first defined by Alan [Cob65] and Jack [Edm65a]. Edmond's definition was in terms of steps for a machine model, which is clearly a precursor of the current definition. Cobham, however, characterized P in terms of its closure under certain operations on languages. This characterization involved operations capable of generating strings of various polynomial lengths, and thus had the notion "polynomial" built into it in the same way as Edmond's. Cook has been involved

with two characterizations of P that do not require any mention of "polynomials" or "time."

The first of these characterizations of P was based on "auxiliary pushdown machines." Pushdown machines come to us from the world of context-free languages. Nondeterministic pushdown machines were used by Noam Chomsky [Cho62] and Robert Evey [Eve63] to characterize the class CFL of context-free languages, which was originally defined by Chomsky [Cho56] in terms of grammars. Later, the class DCFL of deterministic context-free languages was defined by Seymour Ginsburg and Sheila Greibach [GG66], simply by restricting the pushdown machines recognizing them to be deterministic. (The deterministic context-free languages can also be characterized by imposing restrictions on their grammars [see Knu65].) Cook's first "resource-free" characterization of P involved space-bounded auxiliary pushdown machines (where "auxiliary" means that the space needed to maintain the pushdown store is not included in the space bound). Cook [Coo71a], reproduced in this volume, showed that auxiliary pushdown machines (either deterministic or nondeterministic) running in logarithmic space recognize exactly the languages in P. (More generally, he showed that such machines running in space $s(n)$ recognize exactly the languages in $\bigcup_{c>1} \text{TIME}\left(c^{s(n)}\right)$. See Chapter 4 in the current volume on NP-completeness by Christos Papadimitriou for further discussion of this result.) It might be thought that we have just traded one resource bound, "polynomial time," for another, "logarithmic space," that is just as objectionable. But the point here is that logarithmic space can easily be defined without mentioning either "logarithms" or "space" merely by allowing a fixed number of "heads" (or even just movable "markers") on the input tape.

A later "resource-free" characterization of P was given by Stephen Bellantoni and Cook. It characterizes the languages recognizable in polynomial time among those definable by "primitive recursion." The latter are usually defined for integer functions of integer arguments [see Kle52], but can also be defined with binary words replacing integers (as was done by Cobham [Cob65]). Bellantoni and Cook [BC92a, BC92b] showed that by imposing a purely syntactic condition on the recursion schemes used to define functions, the functions that can be defined are all and only those computable in polynomial time.

The sets CFL and DCFL are "families of languages," meaning that they are closed under various operations studied in formal language theory. In particular, they are both "cylinders," meaning that they are closed under inverse homomorphic images and intersections with regular languages. In fact, CFL is a "principal cylinder," meaning that there is a single language $L_1 \in \text{CFL}$ such that every other language $L \in \text{CFL}$ is the intersection of a regular language with an inverse homomorphic image of L_1 (that is, there is regular language R and a homomorphism $h : \Sigma \to \Sigma_1$

from the alphabet Σ of L to the alphabet Σ_1 of L_1 such that $L = R \cap h^{-1}(L_1)$). Such a "hardest context-free language" was first described by Greibach [Gre73]. (Actually, one must take two such languages, one generating languages containing the empty string and another for languages not containing the empty string.) For complexity theory, however, we usually want complexity classes to be closed under some reducibility. Thus, we shall consider the complexity class LOGDCFL, comprising the languages $\leq_m^{\mathsf{L}}$-reducible to a context-free language. Any hardest context-free language is then also a complete language for LOGCFL (with respect to $\leq_m^{\mathsf{L}}$-reducibility). We then have

$$\mathsf{L} \subseteq \mathsf{LOGCFL} \subseteq \mathsf{P},$$

because every language in L is reducible to the context-free language $\{\varepsilon\}$ comprising just the empty string and because every context-free language is in P (as was shown by John Cocke [CS70], Tadao Kasami [Kas66], and Daniel Younger [You67] independently), and P is closed under $\leq_m^{\mathsf{L}}$-reducibility.

For deterministic context-free languages, the situation is different. Greibach [Gre74] (see also Jean-Michel Autebert [Aut79]) has shown that DCFL is *not* a principal cylinder, meaning that there is no single language $L_1 \in \mathsf{DCFL}$ such that every other language $L \in \mathsf{DCFL}$ is $R \cap h^{-1}(L_1)$ for some regular language R and some homomorphism h. But Ivan Sudborough [Sud78] has shown that the complexity class LOGCFL, comprising the languages $\leq_m^{\mathsf{L}}$-reducible to a deterministic context-free language, has a complete language (with respect to $\leq_m^{\mathsf{L}}$-reducibility) that is a deterministic context-free language (and not merely reducible to one). Thus, we have

$$\mathsf{L} \subseteq \mathsf{LOGDCFL} \subseteq \mathsf{LOGCFL} \subseteq \mathsf{P}.$$

In Cook's characterization of P using auxiliary pushdown machines, the auxiliary pushdown machine is not subject to a time bound (only to a space bound). Sudborough [Sud77] has shown that if, in addition to the logarithmic space bound, we impose a polynomial time bound on a deterministic (respectively, nondeterministic) auxiliary pushdown machine, the class of languages recognized is exactly LOGDCFL (respectively, LOGCFL).

8.2 Pebbling

We now come to Cook's contributions to "pebbling." Let G be a *binary acyclic directed graph* (that is, an acyclic directed graph in which every vertex has in-degree two, except for the "sources," which have in-degree zero). We consider the following activity, in which "pebbles" are placed on and removed from the vertices of G

according to the following rules. (1) A pebble may be placed on a vertex if all the immediate predecessors of that vertex currently have pebbles. (Note that this rule allows a pebble to be placed on a source at any time because a source has no immediate predecessors.) (2) A pebble may be removed from a vertex at any time. The goal of this activity is to start with no pebbles on the graph and perform a sequence of placements and removals in which every vertex receives a pebble at some time, ending with no pebbles on the graph. (Note that it would be enough to demand that every "sink" (vertex having out-degree zero) receive a pebble at some time because every other vertex lies on some path to a sink, and thus must be pebbled before that sink can be pebbled.) Every graph can be pebbled by a trivial strategy that consists of arranging its vertices in sequence such that the predecessors of a vertex precede that vertex, pebbling the vertices in this order, then removing all the pebbles. We shall be interested, however, in minimizing the "space" used by a pebbling strategy (the maximum number of pebbles on the graph at any time), even if this entails an increase in the "time" (the total number of placements of pebbles). The trivial strategy described above, for example, uses space n and time n for an n-vertex graph.

The significance of the terms "space" and "time" can be seen by considering the graph as representing a straight-line program in which the sources represent the input values of the computation and in which other vertices represent values computed by applying a dyadic operator to the two values represented by the immediate predecessors, with the sinks representing the output values of the computation. The space used by a pebbling strategy then corresponds to the maximum number of values that must be kept in local storage (think of registers) simultaneously (we do not count the storage required for the input values until they are fetched into local storage), while the time used corresponds to the number of applications of dyadic operations, plus the number of fetches of input values into local storage. (Another interpretation arises by considering the presentation of a proof at a blackboard. The vertices represent assertions, the sources represent axioms, while other vertices represent the proof of an assertion by applying a rule of inference to the two assertions represented by its immediate predecessors.)

The earliest pebbling result is due to Michael Paterson and Carl Hewitt [PH70], who showed that pebbling a balanced binary tree with depth k and $n = 2^{k+1} - 1$ vertices requires exactly $k + 2$ pebbles (unless $k = 0$ and $n = 1$, when one pebble suffices). They used this result to show that the power of recursive programs (with access to a pushdown store implementing recursion) exceeds that of programs with only fixed static storage. For the upper bound, a straightforward recursive strategy (for $k \geq 1$, get a pebble on the root of the left subtree, get a pebble on the root of the right subtree, place a pebble on the root of the tree, then clear the

pebbles from the roots of the subtrees) gets a pebble on the root of the tree while using space $k+2$. For the lower bound, they gave the following argument. Say a path from an input to the root is *open* if all its vertices are unpebbled, otherwise *closed*. Any strategy that gets a pebble on the root starts with all paths open and ends with all paths closed. Consider then the last configuration in which there is an open path P. The move from this configuration must close P by pebbling the input of P (for each of the $k+1$ other vertices on P has its immediate predecessor on P unpebbled). Each of these $k+1$ other vertices on P has another immediate predecessor that is not on P, and each of the $k+1$ disjoint subtrees rooted in these $k+1$ other immediate predecessors must contain at least one pebble (else there would still be an open path), and these $k+1$ pebbles, plus the one just placed, give a total of $k+2$.

A balanced binary tree with n vertices requires space $\Theta(\log n)$. Cook [Coo73b, Coo74] introduced another family of graphs, called *pyramids*, that require more space. A pyramid of height k has $n=(k+1)(k+2)/2$ vertices, regarded as the integer lattice points $(i,j) \in \mathbf{Z} \times \mathbf{Z}$ such that $i \geq 0, j \geq 0$, and $i+j \leq k$. The vertex (i,j) has as immediate successors the vertices $(i-1,j)$ (unless $i=0$) and $(i,j-1)$ (unless $j=0$). It is straightforward to pebble this graph with $k+2$ pebbles, pebbling first the sources (the vertices with $i+j=k$), then the vertices with $i+j=k-1$, and so forth (removing pebbles when they are no longer needed). And a modification of the open-path/closed-path argument given above for trees shows that $k+2$ pebbles are necessary as well as sufficient. But now, because n grows more slowly with k (quadratically, rather than exponentially), we have graphs that require space $\Theta(n^{1/2})$ rather than $\Theta(\log n)$.

At this point, the question arises: what graphs with n vertices require the most space? In particular, are there binary graphs that require space $\Theta(n)$? That the answer to the latter question is "no" was shown by John Hopcroft, Wolfgang Paul, and Leslie Valiant, who showed that space $O(n/\log n)$ is always sufficient and used this result to show that $\text{TIME}(f(n)) \subseteq \text{SPACE}\left(f(n)/\log f(n)\right)$ for multitape Turing machines. (This result has been extended (with a slightly larger space bound) to Turing machines with multidimensional tapes by Paul and Rüdiger Reischuk [PR79, PR81] and (with the original $\text{SPACE}\left(f(n)/\log f(n)\right)$ space bound) to pointer machines by Joseph Halpern, Michael Loui, Albert Meyer, and Daniel Weise [HLMW86]. For Turing machines with a single one-dimensional tape, the smaller space bound $\text{SPACE}\left(f(n)^{1/2}\right)$ has been shown by Paterson [Pat72], even for nondeterministic time-bounded machines and deterministic space-bounded machines.)

The work of Hopcroft, Paul and Valiant [HPV75, HPV77] provides an upper bound of $O(n/\log n)$ to the space required by any n-vertex graph; but are there any

graphs for which $\Omega(n/\log n)$ space is actually necessary? That the answer is "yes" was proved by Paul, Tarjan, and James Celoni [PTC76a, PTC76b, PTC77]. Savings in space usually come at a cost in time, and Lengauer and Tarjan [LT79, LT82] have shown that the time t required to achieve space $s = \Omega(n/\log n)$ is at most

$$t = n \exp\exp O\left(\frac{n}{s}\right),$$

and that there are graphs for which

$$t = n \exp\exp \Omega\left(\frac{n}{s}\right)$$

is actually necessary. Loui [Lou80] has given simplified proofs of both the upper bound for space of Hopcroft, Paul, and Valiant and the upper bound for time of Lengauer and Tarjan.

All these results on pebbling raise the question as to how hard it is to determine the space required by a graph. For trees, the space required can be determined in linear time, using a recurrence first derived by Ershov [Ers58]. (This result has been extended to trees in which the in-degrees of vertices may be greater than two [see LT80].) But for general graphs, it can be very difficult to determine the space required: John Gilbert, Lengauer, and Tarjan [GLT79, GLT80] have shown that this problem is PSPACE-complete.

In 1976, Cook and Ravi Sethi [CS74, CS76] introduced "white pebbles" to pebbling. The pebbles employed previously were now called "black pebbles," and the new white pebbles were placed and removed using rules dual to those of black pebbles, as follows. (1) A white pebble may be placed on a vertex at any time. (2) A white pebble may be removed from a vertex if all the immediate predecessors of that vertex currently have pebbles. (The pebbles that justify the placement of a black pebble or the removal of a white pebble may be black, white, or a combination of the two.) As before, the goal of this activity is to start with no pebbles on the graph and perform a sequence of placements and removals in which every vertex receives a pebble (black or white) at some time, ending with no pebbles (black or white) on the graph.

The use of white pebbles corresponds in a way to nondeterminism. In the interpretation regarding straight-line programs, placing a white pebbles corresponds to guessing the result of an operation, while removing a white pebble corresponds to verifying the correctness of that guess. (In the interpretation regarding presentations of proofs, placing a white pebble corresponds to making an assertion and promising to prove later, while removing a white pebble corresponds to fulfilling that promise.)

The duality that is evident in the rules for black and white pebbles has as its consequence the following observation: if, for a strategy for pebbling a graph, we run it backwards (exchanging placements with removals), and if in addition we exchange black pebbles with white pebbles, we obtain another strategy for pebbling the graph. In particular, the space required using black pebbles alone is the same as the space required using white pebbles alone. But a strategy employing both black and white pebbles may use less space than one using only a single color of pebble. In particular, Cook and Sethi showed that a pyramid of height k could be pebbled with space about $k/2$ when both black and white pebbles are used.

This upper bound shows that white pebbles can be used to reduce space requirements, but they also derived a lower bound of $\Omega(k^{1/2}) = \Omega(n^{1/4})$ for pyramids. They presented this lower bound as evidence that the Solvable Path System problem cannot be solved in polylogarithmic space, even by nondeterministic machines. It is noteworthy that the lower bound they derived for black and white pebbles grows as square root of the $\Omega(k) = \Omega(n^{1/2})$ bound they derived for black pebbles alone, corresponding to the relationship established by Savitch: a lower bound of $f(n)$ for deterministic space-bounded machines implies a lower bound of $f(N)^{1/2}$ for nondeterministic space-bounded machines.

The results of Cook and Sethi raise many questions: how many black and white pebbles are needed for pyramids? How many for other graphs? In particular, what is the largest gap between black and white pebbles and black pebbles alone? The first of these questions was answered by Maria Klawe [Kla83, Kla85], who showed that space about $k/2$ is required to pebble a pyramid of height k using black and white pebbles. For trees, Lengauer and Tarjan [LT80] have shown that white pebbles can reduce space requirements by at most a factor of two, even for trees with unbounded in-degree, and they have shown that the bound $\Omega(n/\log n)$ of Paul, Tarjan, and Celoni, and that their bound

$$t = n \exp\exp \Omega\left(\frac{n}{s}\right)$$

continue to hold, even when white pebbles are allowed.

As for the largest gap, Friedhelm Meyer auf der Heide [Mey79, Mey81], showed that it can be no larger than the squaring that occurs in Savitch's result: any graph that can be pebbled with s black and white pebbles can be pebbled with $O(s^2)$ black pebbles alone. Obtaining results in the other direction proved much more challenging. Robert Wilber [Wil85, Wil88] first proved that the gap could be larger than any constant factor, and later found a graph for which s black and white pebbles suffice, but for which $\Omega(s \log s/\log\log s)$ pebbles are needed if they can only be black. Finally, a matching bound was obtained by Bala Kalyanasundaram and

Georg Schnitger [KS88, KS91], who found a graph for which $O(s)$ black and white pebbles suffice, but for which $\Omega(s^2)$ pebbles are needed if they can only be black.

We again have the problem of how hard it is to determine the space requirements, using both black and white pebbles, of a given graph. For trees, this can be done in time $O(n \log n)$ (even when vertices can have arbitrary in-degree), as was shown by Yannakakis [Yan83, Yan85]. For general graphs, the problem is again PSPACE-complete, as was shown by Philipp Hertel and Toniann Pitassi [HP07, HP10] (but unlike the case of black pebbles alone, this result requires graphs with unbounded in-degree).

The general idea of "pebbling," but with different rules, has continued to recur in the theory of computation. To give just one example, Yuval Filmus, Pitassi, Robert Robere, and Cook [FPRC13b] introduced "reversible pebbling," in which both the placement and removal of a pebble require the immediate predecessors to have pebbles.

We come now to what has come to be called "Steve's Class" (denoted SC). This class was defined by Cook in the spring of 1978 during a discussion of the notion of "simultaneous resource bounds." As noted above, the membership problem for a context free language can solved in polynomial time, and it can also be solved in log-squared space (as was shown by Lewis, Stearns, and Hartmanis [LSH65, Har67]). Thus, it is in $\mathsf{P} \cap \mathsf{L}^2$. But we are referring here to two different algorithms: a polynomial-time algorithm that uses more than polylogarithmic space and a log-squared-space algorithm that uses more than polynomial time. But is there a single algorithm that uses polynomial time and polylogarithmic space? We still do not know the answer to this question, but Cook showed that every *deterministic* context-free language has such an algorithm. Cook [Coo79] defined the complexity class (now known as "Steve's Class" and denoted SC) comprising the problems that can be solved by an algorithm that uses both polynomial time and polylogarithmic space. We also use SC^k when the algorithm uses polynomial time and $\log^k$-space (and we have $\mathsf{SC}^1 = \mathsf{L}$). (See Chapter 7 in the current volume on parallel computation by Beame and McKenzie, for further discussion of the results in this and the next paragraph.) Cook did not refer to pebbling in his paper, but it was soon recognized (by Burchard von Braunmühl and Rutger Verbeek [vBV80], and independently by Kurt Mehlhorn [Meh80]) that it could be formulated in terms of pebbling graphs (called "mountain ranges") that describe the way data is accessed by a deterministic pushdown machine. The final result was a joint paper by von Braunmühl, Cook, Mehlhorn, and Verbeek [vBCMV83].

8.3 Circuits

In that same spring, the current author considered simultaneous resource bounds in order to characterize problems solvable in polynomial time in a "highly parallel" way. The natural way to do this was to consider not time and space for machines but rather "size" and "depth" for circuits, for circuits of small size have small cost, and circuits of very small depth introduce very small delay between the acceptance of the inputs and the production of the outputs. Since a circuit solves the instances of a problem of one particular size, what solves all instances of a problem is an infinite sequence of circuits (one for instances of each size), and so to make sequences of circuits comparable with machines, we must impose some "uniformity" constraint on the sequences of circuits (usually taking the form of a requirement that the circuit for instances of size n can be "easily" computed (in some sense) from n). Assuming this to have been done in some appropriate way, Savage [Sav72] (in one direction) and Pippenger [Pip77] (in the other) established the equivalence of polynomial time for machines and polynomial size for circuits, and Allan Borodin [Bor77] established the equivalence of polylogarithmic space for machines and polylogarithmic depth for circuits. Thus, it might seem that SC would be the desired class, but the arguments relating time and size on one hand, and space and depth on the other, do not appear to work together for simultaneous resource bounds. Thus Pippenger [Pip79] defined a complexity class (now known as "Nick's Class" and denoted NC) comprising the problems that can be solved by a single uniform sequence of circuits having both polynomial size and polylogarithmic depth. We also use NC^k when the circuits have polynomial size and $\log^k$-depth (and Borodin's results show that $\mathsf{NC}^1 \subseteq \mathsf{L} \subseteq \mathsf{SC}^2$). In view of the results of Savage, Pippenger, and Borodin, it may seem strange that NC and SC are apparently different, but Patrick Dymond and Cook [DC89] exhibited a sort of duality between these two classes by introducing what they called "aggregates."

At this point, we should say more about the various notions of uniformity that have been used (that is, about the various ways in which the phrase "some appropriate way," appearing above, may be interpreted). To establish the equivalence of polynomial time for machines and polynomial size for circuits, it is enough to insist that the circuits be "polynomial-time uniform," meaning that a description of the circuit can be computed in polynomial time from a tally of the input (a word over a single-letter alphabet that is the same length as the input). To establish the equivalence of space and depth calls for a more delicate uniformity, and "logarithmic-space uniformity" is commonly used. To illustrate the difference, consider the problem of division of integers represented in binary (to obtain a quotient and remainder). For the simpler problems of addition and multiplication, it

has long been known that they can be performed both by circuits of depth $O(\log n)$ and by machines in logarithmic space. For many years it was an open question as to whether similar results could be obtained for division, until Paul Beame, Cook, and H. James Hoover [BCH84, BCH86] showed in 1986 that division could indeed be performed by circuits of depth $O(\log n)$. But the circuits they described were only polynomial-time uniform, so they could not obtain the conclusion that division can be done in logarithmic space. Only in 2001 did Andrew Chiu, George Davida, and Bruce Litow [CDL01] construct circuits of depth $O(\log n)$ that are logarithmic-space uniform, thereby showing that the division problem is itself solvable in logarithmic space.

But the story of division does not stop here. William Hesse et al. [HAB02, HAB14] showed that division is in a complexity class believed to be even smaller than NC^1, namely the class TC^0 comprising those functions that can be computed by polynomial size, bounded depth circuits (or formulas, it makes no difference) of majority gates (the gates can have any number of inputs, with the size of a circuit being the sum over gates of their number of inputs). Since the majority function is in NC^1, we have $\mathsf{TC}^0 \subseteq \mathsf{NC}^1$. (Multiplication is also in TC^0, and addition is in the even smaller class AC^0, which is like TC^0 but with AND- and OR-gates instead of majority gates.) Could we go farther, and place division in some complexity class that might be even smaller than TC^0? The answer is "no," for Hesse also showed that division is complete for TC^0. Of course, proving completeness for TC^0 calls for a reducibility more delicate than log-space reducibility. The reducibility Hesse used is called $\mathsf{DLOGTIME}$; we will not go into its definition here but merely observe that with Hesse's result, division has found its home (just as Cook showed that NP is the home of Satisfiability, and that P is the home of Solvable Path System).

One cannot help but notice that the various notions of uniformity just discussed are closely related to the various notions of reducibility discussed earlier. Just as one may define uniformities more delicate than logarithmic space, one may define more delicate reducibilities, and these will allow one to explore the fine structure of L and to find problems that are complete for L. For example, Cook and McKenzie [CM87] give a list of problems complete for L, using NC^1-reducibility (which is intermediate between $\mathsf{DLOGTIME}$ and logarithmic space reducibilities). Prominent on this list is the determination of connectedness for undirected graphs consisting of one or more cycles. The restriction of the graphs to collections of cycles arose because, at the time their paper was written, it was not known that connectedness for general undirected graphs is in L. When Omer Reingold [Rei05, Rei08] proved that USTCONN (point-to-point connectedness in general undirected graphs) is in

L, it showed that USTCONN is a natural complete problem for L, in perfect analogy with the completeness of STCONN (point-to-point connectedness in general directed graphs) for NL.

8.4 Branching Programs

Thus far, we have considered two kinds of models for computation: machines and circuits. But there is a third kind of model that plays a role in the paper under discussion: the branching program. Branching programs were introduced by Chester Lee [Lee59], under the name of "binary-decision programs." A branching program is an acyclic directed graph with a single source, in which every vertex has out-degree either two (in which case it represents a query of an input variable, whose value determines which outgoing edge is to be followed) or zero (in which case it announces the output value). For a branching program, "time" is defined as the length of the longest path from the source to a sink (and thus measures the maximum number of queries in a computation), and "space" is defined as the logarithm (to base two) of the number of vertices (and thus represents the number of bits needed to keep track of the state of the computation, where the bits representing the values of the input variables are not counted). As for circuits, branching programs compute Boolean functions, so recognizing a language calls for a *sequence* of branching programs, one for each size of instances. Thus, to establish a correspondence between complexity classes defined by machines and complexity classes defined by branching programs, one must introduce a notion of uniformity for the sequences of branching programs.

That branching programs present a dramatically new perspective on computational complexity is shown by a result of David Barrington. Consider branching programs that are "leveled," in the sense that all paths from the source to a sink have the same length. For such a program, we define the "width" to be the maximum number of vertices at any level. Any Boolean function can be computed by a branching program of width three (consider conjunctive or disjunctive normal form), so interesting classes of functions can only be defined by imposing additional resource constraints. Barringto [Bar86, Bar89] showed that the functions computable in polynomial time by branching programs of bounded width are exactly the functions in NC^1.

Cook's first contribution to the theory of branching programs concerns time–space tradeoffs for the problem of sorting n elements. For this problem, the output is as large as the input, and as always when we discuss small amounts of space, we do not want to count the space used to store the input or the output in our space

bound. Thus, for this problem, we assume the input is read from read-only storage and that the output is written (as the execution of the program progresses) to write-only storage (by announcing the ranks of the elements).

The story starts with a result of Borodin, Fischer, David Kirkpatrick, Nancy Lynch, and Martin Tompa [BFK+79, BFK+81], who proved a lower bound of $st = \Omega(n^2)$ for a modification of the branching program model in which the only operations that can be performed to determine the order of the n elements are pairwise comparisons. This result is not far from the best possible: there are many algorithms that sort with $O(n \log n)$ comparisons and use space $O(n \log n)$ to keep track of a permutation of the n elements. On the other hand, one can get by with $O(\log n)$ space by making n passes over the input, with the k-th pass finding the k-th smallest element; only one element need be remembered from pass to pass; but this algorithm uses time $O(n)$ comparisons per pass, for a total of $O(n^2)$ time.

This result suffers, however, from its restriction to programs that use only pairwise comparisons to determine the order of the input elements. There are "digital" sorting algorithms that directly access the bits of the binary representation of the elements to be sorted, and in some situations such algorithms can provably outperform algorithms that use only pairwise comparisons.

Borodin and Cook [BC82] (included in this volume) improved this result by showing that if the elements to be sorted are integers in the range $[1, n^2]$, and if the program can, as a single operation, branch n^2 ways on the value of any element, a lower bound of $st = \Omega(n^2 / \log n)$ still holds. The model used in this result is very general: it can clearly simulate the operation of any sorting algorithm for a serial computer, even one with random access to storage. In particular, it can simulate any algorithm that performs pairwise comparisons, with a contribution of 2 to time for each comparison, and an overall contribution of $O(\log n)$ to space.

These results raise the question as to whether similar results could be proved for a decision problem (a problem having a yes-or-no answer). Borodin, Faith Fich, Meyer auf der Heide, Eli Upfal, and Avi Wigderson [BFM+86, BFM+87] adapted the result of Borodin, Fischer, Kirkpatrick, Lynch, and Tompa [BFK+79, BFK+81] to the problem of "element uniqueness" (the problem of determining whether all n elements are distinct), using the two-way comparison model, and obtaining a lower bound of $st = \Omega\left(n^{3/2} (\log n)^{1/2}\right)$. Beame [Bea89, Bea91] then showed that the same result held for the general n^2-way branching model, obtaining $st = \Omega(n^2)$.

In this survey, we have not been able to discuss all the results inspired by Cook's work on polynomial time and small amounts of space, nor all recent results on pebbling and branching programs. We have, however, exhibited some of the diversity of such work, and we expect many more such results to be forthcoming in the future.

PART IV

SELECTED PAPERS

The Complexity of Theorem-Proving Procedures

Stephen A. Cook

Summary

It is shown that any recognition problem solved by a polynomial time-bounded nondeterministic Turing machine can be "reduced" to the problem of determining whether a given propositional formula is a tautology. Here "reduced" means, roughly speaking, that the first problem can be solved deterministically in polynomial time provided an oracle is available for solving the second. From this notion of reducible, polynomial degrees of difficulty are defined, and it is shown that the problem of determining tautologyhood has the same polynomial degree as the problem of determining whether the first of two given graphs is isomorphic to a subgraph of the second. Other examples are discussed. A method of measuring the complexity of proof procedures for the predicate calculus is introduced and discussed.

Throughout this paper, a *set of strings* means a set of strings on some fixed, large, finite alphabet Σ. This alphabet is large enough to include symbols for all sets described here. All Turing machines are deterministic recognition devices, unless the contrary is explicitly stated.

1 Tautologies and Polynomial Re-Reducibility

Let us fix a formalism for the propositional calculus in which formulas are written as strings on Σ. Since we will require infinitely many proposition symbols (atoms), each such symbol will consist of a member of Σ followed by a number in binary

Originally published in STOC '71: Proceedings of the third annual ACM symposium on Theory of computing May 1971 Pages 151–158.
Original DOI: 10.1145/800157.805047

notation to distinguish that symbol. Thus a formula of length n can only have about $n/\log n$ distinct function and predicate symbols. The logical connectives are & (and), $\vee$ (or), and $\neg$ (not).

The set of tautologies (denoted by {tautologies}) is a certain recursive set of strings on this alphabet, and we are interested in the problem of finding a good lower bound on its possible recognition times. We provide no such lower bound here, but Theorem 1 will give evidence that {tautologies} is a difficult set to recognize, since many apparently difficult problems can be reduced to determining tautologyhood. By *reduced* we mean, roughly speaking, that if tautologyhood could be decided instantly (by an "oracle") then these problems could be decided in polynomial time. In order to make this notion precise, we introduce query machines, which are like Turing machines with oracles in [KR64].

A *query machine* is a multitape Turing machine with a distinguished tape called the *query tape*, and three distinguished states called the *query state*, *yes state*, and *no state*, respectively. If M is a query machine and T is a set of strings, then a *T-computation* of M is a computation of M in which initially M is in the initial state and has an input string w on its input tape, and each time M assumes the query state there is a string u on the query tape, and the next state M assumes is the yes state if $u \in T$ and the no state if $u \notin T$. We think of an "oracle", which knows T, placing M in the yes state or no state.

Definition A set S of strings is *P-reducible* (P for polynomial) to a set T of strings iff there is some query machine M and a polynomial $Q(n)$ such that for each input string w, the T-computation of M with input w halts within $Q(|w|)$ steps ($|w|$ is the length of w), and ends in an accepting state iff $w \in S$.

It is not hard to see that P-reducibility is a transitive relation. Thus the relation E on sets of strings, given by $(S, T) \in E$ iff each of S and T is P-reducible to the other, is an equivalence relation. The equivalence class containing a set S will be denoted by deg(S) (the polynomial degree of difficulty of S).

Definition We will denote deg({0}) by $\mathscr{L}_*$, where 0 denotes the zero function.

Thus $\mathscr{L}_*$ is the class of sets recognizable in polynomial time. $\mathscr{L}_*$ was discussed in [Coo71a], p. 5, and is the string analog of Cobham's class of functions [Cob65].

We now define the following special sets of strings.

1) *The subgraph problem* is the problem given two finite undirected graphs, determine whether the first is isomorphic to a subgraph of the second. A graph G can be represented by a string $\overline{G}$ on the alphabet $\{0, 1, ,^*\}$ by listing

the successive rows of its adjacency matrix, separated by 's. We let {subgraph pairs} denote the set of strings $\overline{G}_1$ ** $\overline{G}_2$ such that G_1 is isomorphic to a subgraph of G_2.

2) *The graph isomorphism problem* will be represented by the set, denoted by {isomorphic graphpairs}, of all strings $\overline{G}_1$ ** $\overline{G}_2$ such that G_1 is isomorphic to G_2.

3) The set {Primes} is the set of all binary notations for prime numbers.

4) The set {DNF tautologies} is the set of strings representing tautologies in disjunctive normal form.

5) The set D_3 consists of those tautologies in disjunctive normal form in which each disjunct has at most three conjuncts (each of which is an atom or negation of an atom).

Theorem 1 If a set S of strings is accepted by some nondeterministic Turing machine within polynomial time, then S is P-reducible to {DNF tautologies}.

Corollary Each of the sets in definitions 1) - 5) is P-reducible to {DNF tautologies}.

This is because each set, or its complement, is accepted in polynomial time by some nondeterministic Turing machine.

Proof of the theorem. Suppose a nondeterministic Turing machine M accepts a set S of strings within time $Q(n)$, where $Q(n)$ is a polynomial. Given an input w for M, we will construct a proposition formula $A(w)$ in conjunctive normal form such that $A(w)$ is satisfiable iff M accepts w. Thus $A(w)$ is easily put in disjunctive normal form (using De Morgan's laws), and $A(w)$ is a tautology if and only if $w \notin S$. Since the whole construction can be carried out in time bounded by a polynomial in $|w|$ (the length of w), the theorem will be proved.

We may as well assume the Turing machine M has only one tape, which is infinite to the right but has a left-most square. Let us number the squares from left to right $1, 2, \ldots$. Let us fix an input w to M of length n, and suppose $w \in S$. Then there is a computation of M with input w that ends in an accepting state within $T = Q(n)$ steps. The formula $A(w)$ will be built from many different proposition symbols, whose intended meanings, listed below, refer to such a computation.

Suppose the tape alphabet for M is $\{\sigma_1, \ldots, \sigma_\ell\}$, and the set of states is $\{q_1, \cdots, q_s\}$. Notice that since the computation has at most $T = Q(n)$ steps, no tape square beyond number T is scanned.

Proposition symbols $P^i_{s,t}$ *for* $1 \leq i \leq \ell$, $1 \leq s$, $t \leq T$. $P^i_{s,t}$ is true iff tape square number s at step t contains the symbol σ_i.

Q_t^i for $1 \le i \le r, 1 \le t \le T$. Q_t^i is true iff at step t the machine is in state q_1.

$S_{s,t}$ for $1 \le s,\ t \le T$ is true iff at time t square number s is scanned by the tape head.

The formula $A(w)$ is a conjunction $B \,\&\, C \,\&\, D \,\&\, E \,\&\, F \,\&\, G \,\&\, H \,\&\, I$ formed as follows. Notice $A(w)$ is in conjunctive normal form. B will assert that at each step t, one and only one square is scanned. B is a conjunction $B_1 \,\&\, B_2 \,\&\, \cdots \,\&\, B_T$, where B_t asserts that at time t one and only one square is scanned:

$$B_t = (S_{1,t} \vee S_{2,t} \vee \cdots \vee S_{T,t}) \,\&\, [\underset{1 \le i < j \le T}{\&} (\neg S_{i,t} \vee \neg S_{j,t})]$$

For $1 \le s \le T$ and $1 \le t \le T_j$ $C_{s,t}$ asserts that at square s and time t there is one and only one symbol. C is the conjunction of all the $C_{s,t}$.

D asserts that for each t there is one and only one state.

E asserts the initial conditions are satisfied:

$$E = Q_1^o \ \&\ S_{1,1} \ \&\ P_{1,1}^{i_1} \ \&\ P_{2,1}^{i_2} \ \&\ \cdots \ \&\ P_{n,1}^{i_n} \,\&\, P_{n+1,1}^{1} \ \&\ \cdots \ \&\ P_{T,1}^{1}$$

where $w = \sigma_{i_1} \cdots \sigma_{i_n}$, q_o is the initial state and σ_1 is the blank symbol.

F, G, and H assert that for each time t the values of the P's, Q's and S's are updated properly. For example, G is the conjuction over all t, i, j of $G_{i,j}^t$, where $G_{i,j}^t$ asserts that if at time t the machine is in state q_i scanning symbol σ_j, then at time $t+1$ the machine is in state q_k, where q_k is the state given by the transition function for M.

$$G_{i,j}^t = \overset{T}{\underset{s=1}{\&}} (\neg Q_t^i \vee \neg S_{s,t} \vee \neg P_{s,t}^j \vee Q_{t+1}^k)$$

Finally, the formula I asserts that the machine reaches an accepting state at some time. The machine M should be modified so that it continues to compute in some trival fashion after reaching an accepting state, so that $A(w)$ will be satisfied.

It is now straightforward to verify that $A(w)$ has all the properties asserted in the first paragraph of the proof.

Theorem 2 The following sets are P-reducible to each other in pairs (and hence each has the same polynomial degree of difficulty): {tautologies}, {DNF tautologies}, D_3, {subgraph pairs}.

Remark We have not been able to add either {primes} or {isomorphic graph pairs} to the above list. To show {tautologies} is P-reducible to {primes} would seem to require some deep results in number theory, while showing {tautologies} is P-reducible to {isomorphic graph pairs} would probably upset a conjecture of Corneil's [CG70] from which he deduces that the graph isomorphism problem can be solved in polynomial time.

Incidently, it not hard to see from the Davis-Putnam procedure [DP60] that the set D_2 consisting of all DNF tautologies with at most two conjuncts per disjunct, is in $\mathscr{L}^*$. Hence D_2 cannot be added to the list in Theorem 2 (unless all sets in the list are in $\mathscr{L}^*$).

Proof of Theorem 2. By the corollary to Theorem 1, each of the sets is P-reducible to {DNF tautologies}. Since obviously {DNF tautologies} is P-reducible to {tautologies}, it remains to show {DNF tautologies} is P-reducible to D_3 and D_3 is P-reducible to {subgraph pairs}.

To show {DNF tautologies} is P-reducible to D_3, let A be a proposition formula in disjunctive normal form. Say $A = B_1 \vee B_2 \vee \cdots \vee B_k$, where $B_1 = R_1 \,\&\, \cdots \,\&\, R_s$, and each R_i is an atom or negation of an atom, and $s > 3$. Then A is a tautology if and only if A' is a tautology where

$$A' = P \,\&\, R_3 \,\&\, \cdots \,\&\, R_s \vee \neg P \,\&\, R_1 \,\&\, R_2 \vee B_2 \vee \cdots \vee B_k,$$

where P is a new atom. Since we have reduced the number of conjuncts in B_1, this process may be repeated until eventually a formula is found with at most three conjuncts per disjunct. Clearly the entire process is bounded in time by a polynomial in the length of A.

It remains to show D_3 is P-reducible to {subgraph pairs}. Suppose A is a formula in disjunctive normal form with three conjuncts per disjunct. Thus $A = C_1 \vee \cdots \vee C_k$, where $C_i = R_{i1} \,\&\, R_{i2} \,\&\, R_{i3}$, and each R_{ij} is an atom or a negation of an atom. Now let G_1 be the complete graph with vertices $\{v_1, v_2, \ldots, v_k\}$, and let G_2 be the graph with vertices $\{u_{ij}\}$, $1 \leq i \leq k$, $1 \leq j \leq 3$, such that u_{ij} is connected by an edge to u_{rs} if and only if $i \neq r$ and the two literals (R_{ij}, R_{rs}) do not form an opposite pair (that is they are neither of the form $(P, \neg P)$ nor of the form $(\neg P, P)$). Thus there is a falsifying truth assignment to the formula A iff there is a graph homomorphism $\phi : G_1 \to G_2$ such that for each i, $\phi(v_i) = u_{ij}$ for some j. (The homomorphism tells for each i which of R_{i1}, R_{i2}, R_{i3} should be falsified, and the selective lack of edges in G_2 guarantees that the resulting truth assignment is consistently specified).

In order to guarantee that a one-one homomorphism $\phi : G_1 \to G_2$ has the property that for each i, $\phi(v_i) = u_{ij}$ for some j, we modify G_1 and G_2 as follows. We select graphs $H_1, H_2, \ldots, H_k$ which are sufficiently distinct from each other that if G_1' is formed from G_1 by attaching H_i to v_i, $1 \leq i \leq k$, and G_2' is formed from G_2 by attaching H_i to each of u_{i1} and u_{i2} and u_{i3}, $1 \leq i \leq k$, then every one-one homomorphism $\phi : G_1' \to G_2'$ has the property just stated. It is not hard to see such a construction can be carried out in polynomial time. Then G_1' can be embedded in G_2' if and only if $A \notin D_3$. This completes the proof of Theorem 2.

2 Discussion

Theorem 1 and its corollary give strong evidence that it is not easy to determine whether a given proposition formula is a tautology, even if the formula is in normal disjunctive form. Theorems 1 and 2 together suggest that it is fruitless to search for a polynomial decision procedure for the subgraph problem, since success would bring polynomial decision procedures to many other apparently intractible problems. Of course the same remark applies to any combinatorial problem to which {tautologies} is P-reducible.

Furthermore, the theorems suggest that {tautologies} is a good candidate for an interesting set not in $\mathscr{L}^*$, and I feel it is worth spending considerable effort trying to prove this conjecture. Such a proof would be a major breakthrough in complexity theory.

In view of the apparent complexity of {DNF tautologies}, it is interesting to examine the Davis-Putnam procedure [DP60]. This procedure was designed to determine whether a given formula in conjunctive normal form is satisfiable, but of course the "dual" procedure determines whether a given formula in disjunctive normal form is a tautology. I have not yet been able to find a series of examples showing the procedure (treated sympathetically to avoid certain pitfalls) must require more than polynomial time. Nor have I found an interesting upper bound for the time required.

If we let strings represent natural numbers, (or k-tuples of natural numbers) using m-adic or other suitable notation, then the notions in the preceeding sections can be made to apply to sets of numbers (or k-place relations on numbers). It is not hard to see that the set of relations accepted in polynomial time by some nondeterministic Turing machine is precisely the set $\mathscr{L}^+$ of relations of the form

$$(\exists\, y \leq g_k(\bar{x}))\ R(\bar{x}, y) \tag{1}$$

where $g_k(\bar{x}) = 2^{(\ell(\max \bar{x}))^k}$, $\ell(z)$ is the dyadic length of z, and $R(\bar{x}, y)$ is an $\mathscr{L}^*$ relation, ($\mathscr{L}^+$ is the class of extended positive rudimentary relations of Bennett [Ben62]). If we remove the bound on the quantifier in formula (1), the class $\mathscr{L}^+$ would become the class of recursively enumerable sets. Thus if $\mathscr{L}^+$ is the analog of the class of r.e. sets, then determining tautologyhood is the analog of the halting problem; since, according to Theorem 1, {tautologies} has the complete $\mathscr{L}^+$ degree just as the halting problem has the complete r.e. degree. Unfortunately, the diagonal argument which shows the halting problem is not recursive apparently cannot be adapted to show {tautologies} is not in $\mathscr{L}^*$.

3 The Predicate Calculus

Formulas in the predicate calculus are represented by strings in a manner similar to the propositional calculus. In addition to the symbols for the latter, we need the quantifier symbols $\forall$ and $\exists$, and symbols for forming an infinite list of individual variables, and infinite lists of function and predicate symbols of each order (of course the underlying alphabet Σ is still finite).

Suppose Q is a procedure which operates on the above formulas and which terminates on a given input formula A iff A is unsatisfiable. Since there is no decision procedure for satisfiability in the predicate calculus, it follows that there is no recursive function T such that if A is unsatisfiable, then Q will terminate within $T(n)$ steps, where n is the length of A. How then does one appraise the efficiency of the procedure?

We will take the following approach. Most automatic theorem provers depend on the Herbrand theorem, which states briefly that a formula A is unsatisfiable if and only if some conjunction of substitution instances of the functional form $\text{fn}(A)$ of A is truth functionally inconsistent. Suppose we order the terms in the Herbrand universe of $\text{fn}(A)$ according to rank, and then order in a natural way the substitution instances of $\text{fn}(A)$ from the Herbrand universe. The ordering should be such that in general substitution instances which use terms with greater rank follow substitution instances which use terms of lesser rank. Let $A_1, A_2, \ldots$ be these substitution instances in order.

Definition If A is unsatisfiable, then $\phi(A)$ is the least k such that $A_1 \;\&\; A_2 \;\&\; \ldots \;\&\; A_k$ is truth-functionally inconsistent. If A is satisfiable, then $\phi(A)$ is undefined.

Now let Q be the procedure which, given A, computes the sequence $A_1, A_2, \ldots$ and for each i, tests whether $A_1 \;\&\; \ldots \;\&\; A_i$ is truth-functionally consistent. If the answer is ever no, the procedure terminates successfully. Then clearly there is a recursive $T(k)$ such that for all k and all formulas A, if the length of $A \leq k$ and $\phi(A) \leq k$, then Q will terminate within $T(k)$ steps. We suggest that the function $T(k)$ is a measure of the efficiency of Q.

For convenience, all procedures in this section will be realized on single tape Turing machines, which we shall call simply *machines*.

Definition Given a machine M_Q and recursive function $T_Q(k)$, we will say M_Q is of *type* Q *and runs within time* $T_Q(k)$ provided that when M_Q starts with a predicate formula A written on its tape, then M_Q halts if and only if A is unsatisfiable, and for all k, if $\phi(A) \leq k$ and $|A| \leq \log_2 k$, then M_Q halts within $T_Q(k)$ steps. In this case we will also say that $T_Q(k)$ is of type Q. Here $|A|$ is the length of A.

The reason for the condition $|A| \leq \log_2 k$ instead of $|A| \leq k$, is that with the latter condition, finding a lower bound for $T_Q(k)$ would be nearly equivalent to finding a lower bound for the decision problem for the propositional calculus. In particular, Theorem 3A would become obvious and trivial.

Theorem 3 A) For any $T_Q(k)$ of type Q,

$$\frac{T_Q(k)}{\sqrt{k}/(\log\ k)^2} \text{ is unbounded.} \tag{2}$$

B) There is a $T_Q(k)$ of type Q such that

$$T_Q(k) \leq k\, 2^{k(\log\ k)^2}$$

Outline of proof. A). Given any machine M, one can construct a predicate formula $A(M)$ which is satisfiable if and only if M never halts when starting on a blank tape. This is done along the lines described in Wang [Wan62] in the proof which reduces the halting problem to the decision problem for the predicate calculus. Further, if M halts in s steps, then $\phi(A(M)) \leq s^2$. Thus, if, contrary to (2), $T_Q(k) = 0(\sqrt{k}/\log^2 k)$, then a modification of M_Q could verify in only

$$0(\sqrt{s^2}/\log^2 s^2) = 0(s/\log^2 s)$$

steps that M halted in s steps (provided $m \leq \log s^2$, where m is the length of $A(M)$). A diagonal argument (see [HU69] p. 153) shows that this is impossible in general.

B) The machine M_Q operates in time T_Q by following the procedure outlined at the beginning of this section. Note that the formula $A_1\ \&\ A_2\ \&\ \ldots\ \&\ A_k$ has length $0(k \log^2 k)$, since we can assume $|A| \leq \log k$.

Theorem 4 If the set S of strings is accepted by a nondeterministic machine within time $T(n) = 2^n$, and if $T_Q(k)$ is an honest (i.e. real-time countable) function of type Q, then there is a constant K so S can be recognized by a *deterministic* machine within time $T_Q(K8^n)$.

Proof. Suppose M_1 is a nondeterministic machine which accepts S in time 2^n. Let M_2 be a nondeterministic machine which simulates M_1 for exactly 2^n steps and then halts, unless M_1 accepts the input, in which case M_2 computes forever. Thus for all strings w, if $w \in S$ then there is a computation for which M_2 with input w fails to halt, and if $w \notin S$, then M_2 with input w halts within 4^n steps for *all* computations. Now given w of length n, we may construct a formula $A(w)$ of length $0(n)$ such that $A(w)$ is satisfiable if and only if M_1 accepts w. ($A(w)$ is constructed in a way similar

to $A(M)$ in the proof of 1A). Further, if M_2 halts within 4^n steps for all possible computations, then $\phi(A(w)) \leq K(4^n)^2 = K8^n$. Thus, a deterministic machine M can be constructed to determine whether $w \in S$ by presenting M_Q with input $A(w)$. If no result appears within $T_Q(K8^n)$ steps, then $w \in S$, and otherwise $w \notin S$.

4 More Discussion

There is a large gap between the lower bound of $\sqrt{k}/(\log k)^2$ for time functions $T_Q(k)$ given in Theorem 3A and a possible

$$T_Q(k) = k2^{k(\log k)^2}$$

given in 3B. However, there are reasons for the gap. For example, if we could improve the result in 3B and find a $T_Q(k)$ bounded by a polynomial in k, then by Theorem 4 we could simulate a nondeterministic 2^n time bounded machine deterministically in time $p(2^n)$ for some polynomial p. This is contrary to experience which indicates deterministic simulation of a nondeterministic $T(n)$ time bounded machine requires time $k^{T(n)}$ in general.

On the other hand, if we could push up the lower bound given in Theorem 3A and show

$$\frac{T_Q(k)}{2^k}$$

is unbounded, then we could conclude $\{\text{Tautologies}\} \notin \mathscr{L}^*$, since otherwise the general Herbrand proof procedure would provide a $T_Q(k)$ smaller than 2^k. Thus such an improvement in 3A would require a major breakthrough in complexity theory.

The field of mechanical theorem proving badly needs a basis for comparing and evaluating the dozens of procedures which appear in the literature. Performance of a procedure on examples by computer is a good criterion, but not sufficient (unless the procedure proves useful in some practical way). A theoretical complexity criterion is needed which will bring out fundamental limitations and suggest new goals to pursue. The criterion suggested here (the function $T_Q(k)$) is probably too crude. For example, it might be better to make $T_Q(k)$ a function of several variables, of which one is $\phi(A)$, and another might be the minimum number of substitution instances of fn(A) needed to form a contradiction (note that in general not all of $A_1, A_2, \cdots, A_{\phi(A)}$ are needed.)

$T_Q(k)$ may be a crude measure, but it does provide a basis for discussion, and, I hope, will stimulate progress toward finding better complexity measures for theorem provers.

References

[Ben62] J. H. Bennett. 1962. *On Spectra*. Ph.D. thesis. Princeton University.

[Cob65] A. Cobham. 1965. The intrinsic computational difficulty of functions. In *Proceedings of the 1964 International Congress for Logic, Methodology, and Philosophy of Science*. North-Holland, Amsterdam, 24–30.

[Coo71] S. A. Cook. 1971. Characterizations of pushdown machines in terms of time-bounded computers. *J. ACM* 18, 1, 4–18. DOI: https://doi.org/10.1145/321623.321625.

[CG70] D. G. Corneil and C. C. Gotlieb. January. 1970. An efficient algorithm for graph isomorphism. *J. Assoc. Comput. Machinery* 17, 1, 51–64. DOI: https://doi.org/10.1145/321556.321562.

[DP60] M. Davis and H. Putnam. 1960. A computing procedure for quantification theory. *J. Assoc. Comput. Machinery* 7, 201–215. DOI: https://doi.org/10.1145/321033.321034.

[HU69] J. E. Hopcroft and J. D. Ullman. 1969. *Formal Languages and their Relation to Automata*. Addison-Wesley, 262.

[KR64] D. L. Kreider and R. W. Ritchie. 1964. Predictably computable functionals and definitions by recursion. *Zeitschrift für math. Logik und Grundlagen der Math*. 10, 65–80.

[Wan62] H. Wang. 1962. Dominoes and the AEA case of the decision problem. In *Proceedings of the Symposium on Mathematical Theory of Automata at Polytechnic Institute of Brooklyn*. Polytechnic Press, 23–55.

10 Characterizations of Pushdown Machines in Terms of Time-Bounded Computers

Stephen A. Cook

Abstract

A class of machines called *auxiliary pushdown machines* is introduced. Several types of pushdown automata, including stack automata, are characterized in terms of these machines. The computing power of each class of machines in question is characterized in terms of time-bounded Turing machines, and corollaries are derived which answer some open questions in the field.

Key words and Phrases

Turing machines, multitape Turing machines, time-bounded computers, abstract computer models, pushdown automata, multihead pushdown automata, stack automata, writing pushdown acceptors, auxiliary pushdown machines, computational complexity

---

Department of Mathematics. The research reported here was supported in part by Office of Naval Research Contract Nonr 3656 (23) and by the National Science Foundation Contract GJ474. Most of the results in this paper were announced in [Coo69].

Originally published in Journal of the Association for Computing Machinery, Vol. 18, No. 1, January 1971, pp. 4–18. https://doi.org/10.1145/321623.321625

CR Categories

5.22, 5.23

1 Introduction

In this paper we have two main purposes: first, to provide a unified treatment of several types of two-way pushdown automata and stack automata, and, second, to give an interesting characterization of the computing power of each of these machines in terms of deterministic time-bounded Turing machines.

In Section 2 we introduce the notion of *time-bounded computer.* This is simply a general computer model used to distract attention from picky arguments about Turing machine tapes and heads. In Section 3 we introduce the notion of *auxiliary pushdown machine*, which is like a tape-bounded Turing machine but has a pushdown store attached whose storage does not count in the tape bound. The main theorem, in Section 4, characterizes the computing power of auxiliary pushdown machines for tape bounds $L(n) \geq \log_2 n$ in terms of time-bounded computers, and states that for any tape bound $L(n) \geq \log_2 n$ the deterministic and nondeterministic versions have the same computing power. Here n is the length of the input string.

Theorem 2 in Section 4 states that the computing power of two-way multihead pushdown machines, writing pushdown acceptors, and two-way stack automata can be characterized in terms of auxiliary pushdown machines, and hence in terms of time-bounded computers. In particular, a deterministic two-way stack automaton is equivalent to a deterministic n^{cn} time-bounded Turing machine, and a nondeterministic two-way stack automaton is equivalent to a deterministic 2^{cn^2} time-bounded Turing machine. A number of corollaries are derived from Theorems 1 and 2, some of which answer open questions in the literature.

2 Time-Bounded Computers

Turing machines of various kinds are the most common abstract computer model used in the theory of computational complexity. Yet they are often criticized as models of real computers, since they do not behave much like random-access machines. In fact, some of the results in the literature showing Turing machines require a great deal of time to recognize simple sets of strings, surely depend on exploiting the inefficient storage arrangement of a single head on a single tape, and do not reflect any fundamental property of computation. Many other results, however, hold as well for any reasonable computer model as they do for Turing machines.

The results in this paper are of the latter kind. In order to emphasize this point we shall quote our results in terms of "time-bounded computers" instead of Turing machines. Informally, a time-bounded computer is any device equipped to accept a set A of strings within a certain time bound $T(n)$, provided some Turing machine accepts A within time $(T(n))^k$ for some k. Our formal definition will not characterize the possible such devices, but just the notion of acceptance within a time bound.

Definition Let $T(n)$ be a function from positive integers to positive integers, and let A be a set of strings over a finite alphabet Γ. Then we say *A is accepted by a time-bounded computer within time $T(n)$* provided some simple deterministic Turing machine M (see Section 3) accepts A within time $(T(n))^k$, for some constant k.

Examples of time-bounded computers are simple Turing machines, multitape Turing machines, iterative arrays of finite-state machines [Col66], Shepherdson-Sturgis machines [SS63], and abstract random access machines in the sense of Earley [Ear70, p. 102]. The proofs in each case are straightforward; see, for example, [HS65] for the case of multitape Turing machines.

The motivation for this notion of time-bounded computer rests on a feeling that (1) any reasonable computer model should be at least as strong as a simple Turing machine [i.e. to demonstrate conclusively that a task can be performed within time $T(n)$ it is more than sufficient to show that a simple Turing machine can do so], (2) as stated above a large class of computer models is included in the notion of time-bounded computer—in particular, a plausible formal notion of random-access machine—and (3) at the present state of the art it is difficult to find a convincing more restricted definition of computer.

In Section 5 we refer to the class $\mathfrak{L}_*$ of sets of strings, first defined by Cobham [Cob65] in terms of numerical functions instead of sets of strings.

Definition A set A of strings is in the class $\mathfrak{L}_*$ if and only if A is accepted by a time-bounded computer within time $P(n)$, for some polynomial $P(n)$.

The class $\mathfrak{L}_*$ is the smallest class which can be characterized (in a reasonable way) in terms of time-bounded computers. As pointed out by Cobham [Cob65], the class is unchanged if the word "computer" in the definition is replaced by any of "random-access machine," "Turing machine," "Shepherdson-Sturgis machine," or "iterative array of finite-state machines." It will turn out (see Section 5) that $\mathfrak{L}_*$ has an interesting characterization in terms of multihead pushdown machines.

3 Other Machine Models

By a *simple Turing machine* we mean the familiar device consisting of a finite-state control attached to a single read/write head moving along a single two-way infinite

tape. In one step, the machine can assume a new state, print one of a finite set of symbols on the tape square currently scanned, and shift its head either left or right one square, or leave it stationary. The action of the machine in a step depends on the current state and tape symbol being scanned. By a *multitape Turing machine* we mean a finite-state control attached to both a two-way read-only *input head* moving along an *input tape*, and finitely many read/write *work* heads each moving along a distinct two-way infinite *work tape.* In one step, the machine can assume a new state, print one of a finite set of symbols (excluding the blank symbol) on each of its work tapes (but not its input tape), and shift some of its heads left or right one square in any combination. The action taken depends on the current state of the machine and the symbols scanned by each of its heads.

Before any computation a finite input string is placed on the input tape delimited by blanks at each end. The finite-state control is designed so that the input head will never leave the segment consisting of this string and the two blanks during any computation.

An *auxiliary pushdown machine* (auxiliary PDM) is a multitape Turing machine which has an extra tape called the *pushdown tape*, which operates in a special fashion. The pushdown alphabet has a distinguished symbol s_0 which appears initially on the pushdown tape. The machine is designed so that the pushdown head never shifts left of s_0 or changes s_0. Further, the pushdown head can never shift left when scanning any tape symbol unless it first erases (i.e. overwrites a blank on) that symbol, and it can never shift right from a square unless it first prints a nonblank symbol on that square. Initially the pushdown head is scanning the symbol s_0 and all other squares on the pushdown tape are blank. Thus throughout the computation all squares on the pushdown tape are blank except for the segment bounded on the left by s_0 and on the right by the pushdown head, which is nonblank.

The last type of machine we describe here is the two-way stack automaton, introduced in [GGH67]. For our purposes, a two-way stack automaton, or briefly *stack automaton*, consists of a finite-state control attached to an *input tape* and a *stack tape*. The input tape is just like that for multitape-Turing machines: it is of the two-way read-only type, with input delimited by blanks. The stack tape is like the pushdown tape for an auxiliary PDM, except the stack head is allowed to read the information on the stack between the symbol s_0 and the rightmost nonblank symbol. At the beginning of any computation, the stack tape is blank except for one square containing the symbol s_0, and the stack head scans this square. The input tape is initially blank, except for the input string w, and the input head scans the leftmost symbol of w. In one step, the machine will, depending on its internal state and the symbols scanned by the input head and stack head, assume a new state, shift its input head one or zero squares left or right (but not outside the blanks

delimiting the input), and shift the stack head one or zero squares left or right, but not left of the symbol s_0. In addition, the stack head may print a symbol before shifting, provided either it is scanning the *leftmost* blank symbol to the right of s_0 on the stack tape or if on the preceding step the stack head printed a blank and shifted left, and in addition provided the scanned symbol is not s_0 and provided the stack head does not print a blank and shift right. These provisions are implemented by the design of the finite-state control, and by use of an enlarged symbol alphabet for the stack tape.

We shall sometimes allow auxiliary PDM's and stack automata to be nondeterministic, although Turing machines are here always assumed to be deterministic, unless the contrary is explicitly stated.

Each of the above types of machines has certain distinguished states called *accepting* states, and a single distinguished state q_0 called the initial state. Let M be one of the above machines (deterministic or not), and let Γ be a (finite) subset of the set of symbols which M can read on its input tape (here we will refer to the single tape of a simple Turing machine as the input tape), and suppose Γ does not contain the blank symbol. Then we say M *accepts* a string w on Γ provided that some computation of M terminates in an accepting state, where initially M is in the state q_0, and all tapes are blank except the input tape contains the string w with the input head scanning the leftmost symbol of w (and the pushdown or stack tape contains the symbol s_0). For the case of auxiliary PDM's we shall require in addition that the pushdown head scan the symbol so (i.e. the pushdown list is empty) at the end of the accepting computation.

Suppose $T(n)$ and $L(n)$ are functions on the positive integers, and suppose A is a set of strings on Γ. Then we say a machine M *accepts* the set A *within time* $T(n)$ provided, for each $w \in A$, M accepts w in some computation consisting of $T(|w|)$ or fewer steps, where $|w|$ is the length of w, and provided M accepts no strings w not in A. We say a multitape Turing machine or auxiliary PDM *accepts* A *within storage* $L(n)$ provided, for each $w \in A$, M accepts w in some computation in which no work tape (excluding the input tape and pushdown tape) scans more than $L(|w|)$ distinct squares, and provided M accepts no strings w not in A.

4 The Main Theorem

Theorem 1 **Main Theorem**

The following three conditions are equivalent for any set A of strings on an alphabet Γ and for any function $L(n) \geq \log_2 n$ on the positive integers.

(a) *A is accepted by some deterministic auxiliary PDM within storage $L(n)$.*

(b) *A is accepted by some nondeterministic auxiliary PDM within storage* $L(n)$.

(c) *A is accepted by some time-bounded computer within time* $T(n) = 2^{cL(n)}$ *for some constant c.*

Proof. (a) $\Rightarrow$ (b). Obvious.

(b) $\Rightarrow$ (c). The argument is a generalization of one appearing in [AHU68]. Suppose M_1 is a nondeterministic auxiliary PDM which accepts the set A within storage $L(n)$. We will construct a deterministic multitape Turing machine M_2 which accepts A within time $T(n) = 2^{cL(n)}$ for some constant c.

Fix a string w on the input alphabet Γ, let $n = |w|$ (the length of w), and suppose M_1 has k work tapes in addition to the input tape and pushdown tape. Then a *configuration* of M_1 with input w is a string $P = pqu_1{\downarrow}v_1 * u_2{\downarrow}v_2 * \cdots * u_k{\downarrow}v_k * s$, where p is the dyadic notation for an integer between 0 and $n+1$, q is a state of $M_1, u_1, \ldots, u_k$ and $v_1, \ldots, v_k$ are strings on the work tape alphabets, s is a symbol on the pushdown alphabet, and $\downarrow$ and $*$ are new symbols. We say M_1 with input w is in configuration P provided M_1 has the string w delimited by blanks written on its input tape, the input head is scanning symbol number p from the left (counting the blank immediately to the left of w as symbol number 0), M_1 is in state q, the k work tapes have the strings $u_1v_1, u_2v_2, \cdots, u_kv_k$ written on them with work head i scanning the first symbol of v_i ($i = 1, 2, \ldots, k$), and the symbol s is currently scanned by the pushdown head (i.e. on top of the pushdown list). Thus a configuration completely specifies an instantaneous description of M_1, except the input string w and the contents of the pushdown tape (other than the currently scanned symbol) are left unspecified.

We say the pair (P, Q) of configurations of M_1 with input w is *realizable* provided there is some partial computation of M_1 with input w such that at the beginning of the partial computation M_1 is in configuration P with the pushdown head scanning some square c, and at the end of the partial computation M_1 is in configuration Q with the pushdown head scanning the same square c (although c need not have the same symbol written on it) and throughout the partial computation the pushdown head never moves to the left of c. If we let P_0 correspond to the initial configuration, so that $P_0 = 1q_0{\downarrow}\beta * \cdots * {\downarrow}\beta s_0$ (where β represents the blank symbol), then M_1 accepts the input w if and only if there is some configuration Q_a, whose state symbol is an accepting state, such that the pair (P_0, Q_a) is realizable. We shall call such a pair (P_0, Q_a) an *accepting pair*.

We shall design the Turing machine M_2 to build a list of realizable pairs (P, Q) on one of its work tapes. If ever M_2 finds an accepting pair (P_0, Q_a), it will halt and accept the input string w. Otherwise M_2 computes forever.

Since the machine M_1 is supposed to operate within storage $L(n)$, M_2 need only consider configurations $P = pqu_1\downarrow v_1 * \cdots * u_k\downarrow v_k * s$ whose work tape strings u_iv_i do not exceed $L(n)$ in length, where $n = |w|$. However, the function L may be difficult or impossible to compute, so rather than calculate the value $L(n)$, M_2 uses a parameter l, stored on one of its work tapes, to guess at $L(n)$. Initially $l = 1$. In general, after M_2 has found all realizable pairs (P, Q) whose work tape storage does not exceed l, and if no accepting pair (P_0, Q_a) has been found, then l is increased by one. The process continues indefinitely.

The list of realizable pairs (P, Q) formed by M_2 is initially empty. In general, after all realizable pairs have been found for a particular value of l, l is increased by one and the list is increased as follows. First, all pairs (P, P) of configurations whose work storage is equal to l are added to the list. Next, for every two pairs $(P_1, Q_1), (P_2, Q_2)$ appearing on the list, each pair (P_3, Q_3) is added to the list such that (1) the pairs $(P_1, Q_1), (P_2, Q_2)$ "yield" the pair (P_3, Q_3) in the sense defined below, (2) configurations P_3, Q_3 each have work storage at most l, and (3) the pair (P_3, Q_3) does not already appear on the list. This last step is repeated until either an accepting pair (P_0, Q_a) is found, in which case M_2 accepts w, or no new realizable pairs can be found, in which case l is increased by one and the process starts all over again.

Definition The two pairs (P_1, Q_1) and (P_2, Q_2) of configurations are said to *yield* the pair (P_3, Q_3) provided $P_1 = P_3$, and either (i) $Q_1 = P_2$ and M_1 can go from configuration Q_2 to Q_3 in one step without shifting its pushdown head, or (ii) M_1 can go from Q_1 to P_2 in one step by printing some symbol s on its pushdown tape and shifting its pushdown head right, M_1 can go from Q_2 to Q_3 in one step by shifting its pushdown head left, and when M_1 is in configuration Q_3 its pushdown head scans this symbol s.

Clearly if (P_1, Q_1) and (P_2, Q_2) are realizable pairs which yield (P_3, Q_3), then (P_3, Q_3) is realizable. Conversely, the following lemma shows that the procedure outlined above for M_2 will eventually produce all realizable pairs.

Lemma 1 *Every realizable pair (P, Q) of configurations of work tape storage l or less can be obtained from pairs of the form (P, P) by successively applying the yield relation, where all configurations appearing have work tape storage l or less.*

Proof. Suppose the pair (P, Q) is realizable and the work storage of the configurations P, Q does not exceed l. Then there is some partial computation of M_1 with input w whose initial configuration is P, whose final configuration is Q, and such that the pushdown head satisfies the restrictions in the definition of "realizable." Let $P = P_1, P_2, \ldots, P_t = Q$ be the configurations of the successive steps in the computation. Then the work storage for none of the P_i can exceed l, since the storage cannot shrink during a computation. We shall prove by induction on t that (P, Q)

can be obtained as stated in the lemma. If $t = 1$, then (P, Q) is one of the initial pairs (P, P). Suppose $t > 1$. There are two cases:

(i) Suppose the pushdown head remains stationary in the step P_{t-1}, P_t. Then the pair (P_1, P_{t-1}) is realizable, and by the induction hypothesis it can be obtained as stated in the lemma. But (P_1, P_1) and (P_1, P_{t-1}) yield (P_1, P_t). Therefore $(P_1, P_t) = (P, Q)$ can be obtained as stated in the lemma.

(ii) Suppose the pushdown head shifts left in the step P_{t-1}, P_t (it cannot shift right). Let P_i be the first configuration in the computation such that the pushdown head shifts right in the step P_i, P_{i+1}. Then both the pairs (P_1, P_i) and (P_{i+1}, P_{t-1}) are realizable, and by the induction hypothesis they can be obtained as stated in the lemma. But the pairs (P_1, P_i) and (P_{i+1}, P_{t-1}) yield (P_1, P_t). Therefore $(P_1, P_t) = (P, Q)$ can be obtained as stated in the lemma.

It remains to estimate the time required by M_2 to carry out the simulation. Clearly there is some constant c_1 (independent of w and $n = |w|$) such that for each value of $l \geq \log_2 n$ [recall $L(n) \geq \log_2 n$], M_2 requires at most $c_1 l + c_1$ squares to represent any pair (P, Q) of configurations on its work tape, provided the work storage indicated in P, Q does not exceed l. Thus there is some constant γ such that the total number of possible configuration pairs of work storage not exceeding l is bounded by $\gamma^{c_1 l + c_1}$. Hence the number of realizable pairs of configurations appearing in the list on M_2's work tape never exceeds $\gamma^{c_1 l + c_1}$. Since the time required to add one pair to the list will not exceed a constant times, say, the fourth power of this bound, and since if M_1 accepts w [within storage $L(n)$] then M_2 will accept w for some $l \leq L(n)$, it is clear that there is some constant c such that if M_1 accepts w, then M_2 will accept w within $2^{cL(n)}$ steps.

(c) $\Rightarrow$ (a). Suppose some time-bounded computer accepts the set A within time $T(n) = 2^{cL(n)}$. Then, by definition of acceptance by a time-bounded computer, there is a simple (deterministic) Turing machine S and a constant d such that S accepts A within time $2^{dL(n)}$. We first modify S to form a simple Turing machine M_1 which accepts A within time $2^{c_1 L(n)}$ for some constant c_1, such that the head of M_1 operates in a regular predictable pattern as follows. Suppose initially M_1 has the input w of length n on its tape, delimited by blanks, with the head scanning the leftmost symbol of w. Throughout its computation, M_1 prints only nonblank symbols. First the head shifts right for n successive steps until reaching the first blank square, immediately to the right of w. Then the head prints a nonblank symbol and shifts left for $n + 1$ successive steps until reaching the first blank square, immediately to the left of w. The head prints a nonblank symbol, shifts right for $n + 2$ steps, then left for $n + 3$ steps, and so on. Meanwhile M_1 is simulating S by carrying out one or more print operations of S's computation for each sweep of M_1's head (except possibly the first). On the first sweep right, M_1 simulates S as long as S continues to shift right at the rate of one shift per step. Then M_1 marks that square, and continues

the simulation when its head sweeps back left. The details of the simulation are left to the reader. The important thing is that M_1 requires at most one sweep, and hence at most $T(n) = 2^{dL(n)}$ steps, for each step of S, and hence if S accepts w, then M_1 accepts w within $[T(n)]^2 = 2^{2dL(n)}$ steps.

We shall now construct a deterministic auxiliary PDM M_2 which indirectly simulates M_1. Let us fix an input string w of length n, and consider triples $\langle t, q, s\rangle$, where t is a nonnegative integer, q is a state of M_1, and s is a symbol in the tape alphabet of M_1. Such a triple is said to be *realizable* provided that, at step number t of the computation of M_1 with input w, M_1 is in state q scanning the symbol s. The simulating machine M_2, with input w, will operate by making a list of coded forms of realizable triples $\langle t, q, s\rangle$ on its pushdown tape, always searching for one such that q is an accepting state. Note that M_1 accepts w if and only if there is some realizable triple $\langle t, q, s\rangle$ such that q is an accepting state.

Given a realizable triple $\langle t, q, s\rangle$, the immediate task of M_2 is to calculate the realizable triple $\langle t+1, q', s'\rangle$, which describes M_1 at step $t+1$ of its computation. The state q' is easily determined by applying M_1's state transition function to the pair (q, s). However, in order to determine s', usually M_2 must have access to a realizable triple $\langle t_1, q_1, s_1\rangle$ describing M_1 the last time it scanned the square it scans at step $t+1$. We make the relationship among these three triples precise as follows:

Definition The triples $\langle t, q, s\rangle$ and $\langle t_1, q_1, s_1\rangle$ are said to *yield* the triple $\langle t+1, q', s'\rangle$ provided that when M_1 is in state q scanning the symbol s, it next assumes the state q', and either (i) M_1 scans a square for the first time at time $t+1$, in which case s' is the symbol originally occupied by that square (either blank or a symbol of w), or (ii) t_1 is the greatest integer less than $t+1$ such that M_1 scans the same square at steps t_1 and $t+1$, in which case s' is the symbol printed by M_1 when in state q_1 scanning the symbol s_1.

It should be clear that if this yield relation holds, and if both the triples $\langle t, q, s\rangle$ and $\langle t_1, q_1, s_1\rangle$ are realizable, then $\langle t+1, q', s'\rangle$ is realizable. Furthermore, the question of whether or not the three triples satisfy the yield relation can be easily answered by M_1, because of the predictable pattern described by M_1's head. This point will be discussed later.

We can describe M_2's operation informally as follows. We assume M_2 has enough work tape space available to store, say, at least four triples. Thus M_2 has no trouble in calculating realizable triples $\langle t, q, s\rangle$ for $t \leq n$, i.e. triples describing the first sweep of M_1's head. Now let us assume as an informal induction hypothesis that for some fixed t, and every $\tau \leq t$, there will be a point in M_2's computation at which the realizable triple $\langle \tau, q_\tau, s_\tau\rangle$, which describes M_1 at time τ, will appear alone on M_2's pushdown tape. Suppose M_2 has just written the realizable triple $\langle t, q, s\rangle$ on

its pushdown tape, which is otherwise empty. Let $\langle t_1, q_1, s_1\rangle$ be a realizable triple such that $\langle t, q, s\rangle$ and $\langle t_1, q_1, s_1\rangle$ yield $\langle t+1, q', s'\rangle$. M_2 will proceed by repeating its entire computation to date, except now $\langle t, q, s\rangle$ will be treated as the bottom (left end) of the pushdown tape, and $\langle t, q, s\rangle$ will remain undisturbed. By our induction hypothesis, the triple $\langle t_1, q_1, s_1\rangle$ will appear immediately to the right of $\langle t, q, s\rangle$ on the pushdown tape at some point in the computation. M_2 will constantly check (using work tape storage) to see whether the rightmost two triples on its pushdown tape can be combined to yield a third triple, and if this is possible, M will replace the two triples by the third triple. Thus, at some point in the computation, the realizable triple $\langle t+1, q', s'\rangle$ will appear alone on the pushdown tape, and we have carried the induction one step farther. Of course we must supply a program for M_2 before the argument can be made precise. This is done below.

We note that in the course of computing the triple $\langle t, q, s\rangle$ and placing it at the left end of its pushdown tape, M_2 will develop a list of triples on its pushdown tape which, at a maximum, contains as many triples plus one as the head of M_1 makes sweeps up to step t.

We now give the precise algorithm which M_2 follows. We assume that throughout its computation M_2 has a list of triples on its pushdown tape. (The value of t in $\langle t, q, s\rangle$ is stored in binary notation.) Initially this list is empty. The *top* of the list means the rightmost entry on the list. Of course throughout the computation M_2 has access to the input string w.

4.1 Algorithm for M_2

P1 (ADD FIRST TRIPLE). Add the triple $\langle 0, q_0, \bar{s}\rangle$ to the top of the pushdown list, where q_0 is the initial state of M_1 and $\bar{s}$ is the leftmost symbol of w.

P2 (ACCEPT?). If the top triple is $\langle t, q, s\rangle$, where q is an accepting state of M_1, then ACCEPT the input w. Otherwise, go to P3.

P3 (APPLY YIELD RELATION). If the list has at least two triples, and x and y are the top two triples, and x and y yield a triple z, then remove x and y from the list and add z, and go to P2. Otherwise go to P1.

To prove the algorithm works (i.e. causes M_2 to accept the input w if and only if M_1 accepts w), one need only formalize the informal argument given above. Notice that initially P1 is executed twice before P3 succeeds, so that two copies of $\langle 0, q_0, \bar{s}\rangle$ appear on the pushdown list.

It remains to estimate the auxiliary storage (i.e. work tape storage) required by M_2. We shall confine our remarks to step P3, since it is clearly the most difficult.

The key to executing step P3 is computing the function $\pi(n, t)$, whose value is the position of the head of M_1 at step number t of a computation with an input string of

length n. The position is an integer assigned to the square scanned at step t. Integers are assigned to squares on the tape by assigning 0 to the square containing the leftmost symbol of the input w (in its position at the start of the computation), consecutive positive integers to squares to the right of square 0, and consecutive negative integers to squares to the left of square 0. Assuming the argument t is available on a work tape in binary notation, and n is the length of the input string, M_2 computes $\pi(n, t)$ by simulating on a work tape the head motion of M_1 for t steps, keeping track of M_1's head position during the simulation by a parameter p written in binary notation. Thus the number of work tape squares required to compute $\pi(n, t)$ is at most $d_1 \max(\log_2 n, \log_2 t)$ for some small constant d_1. If M_1 accepts w then it does so within $2^{c_1 L(n)}$ steps. Hence values of t considered by M_2 will never exceed $2^{c_1 L(n)}$, so that the storage used is at most $d_1 \max(\log_2 n, c_1 L(n))$ and, hence, at most $d_2 L(n)$ for some constant d_2, since $L(n) \geq \log_2 n$. If M_1 fails to accept w, we are not concerned with the storage required by M_2.

To execute step P3, M_2 copies the top two triples, if they exist, from the pushdown tape (where they will be destroyed) to a work tape. If the pushdown tape contains only one triple, M_2 restores that triple to the pushdown tape and passes control to step P1. Otherwise, M_2 determines whether or not the top two triples yield a triple z. This is easily done by computing $\pi(n, t+1)$, and $\pi(n, \tau)$ for successive values of $\tau \leq t$, and by referring to the transition function for M_1 and referring to the input w. If the triple z can be found, it is added to the top of the pushdown list and control is passed to step P2. Otherwise, the top two triples are restored to the pushdown tape and control is passed to step P1.

As mentioned above, if M_1 accepts w, then the value of t in the triples $\langle t, q, s\rangle$ considered by M_2 never exceeds $2^{c_1 L(n)}$, and hence the space required to store a triple is bounded by $d_3 L(n) + d_3$ for some constant d_3. From this it is clear from the above discussion that the auxiliary storage required by M_2, in case M_1 (and hence M_2) accepts w, is at most $d_4 L(n) + d_5$, for some constants d_4, d_5. By using the standard techniques of increasing the work tape alphabet and finite-state control for M_2, the constants d_4 and d_5 can be reduced to 1 and 0. Thus in fact M_2 can be made to accept the set A within storage $L(n)$.

5 Applications of the Main Theorem

The theorem below gives characterizations of the computing power of several types of pushdown machines in terms of time-bounded computers. In addition to the pushdown devices described in Section 3, the theorem mentions two others: the writing pushdown acceptors of Mager [Mag69] and the two-way multihead pushdown automata discussed in [HI68]. A *writing pushdown acceptor* was defined in [Mag69] to be a nondeterministic linearly bounded automaton equipped with a

Table 1

Machine type	$L(n)$ for auxiliary PDM	$T(n)$ for deterministic Turing machine
1. Two-way multihead pushdown automaton	$\log n$	n^c, constant c
2. Deterministic two-way multihead pushdown automaton	$\log n$	n^c, constant c
3. Writing pushdown acceptor	n	2^{cn}, constant c
4. Deterministic writing pushdown acceptor	n	2^{cn}, constant c
5. Deterministic two-way stack automaton	$n \log n$	n^{cn}, constant c
6. Nondeterministic two-way stack automaton	n^2	2^{cn^2}, constant c

pushdown store, but this is easily seen to be equivalent in computing power to a nondeterministic auxiliary PDM with storage bounded by $L(n) = n$. A multihead two-way pushdown automaton consists of a nondeterministic finite-state control attached to a pushdown store, and to several two-way read-only input heads operating on an input tape with endmarkers.

Theorem 2 *A set A of strings is accepted by a machine of type i ($1 \leq i \leq 6$) in the first column of Table 1 if and only if A is accepted by some auxiliary PDM (deterministic or not) within storage $L(n)$ given in row i of the second column, and again if and only if A is accepted by a time-bounded computer within time $T(n)$ given in row i of the third column.*

Proof. The equivalences indicated between columns 2 and 3 of the table follow directly from Theorem 1. Of the equivalences between columns 1 and 2, those in rows 3 and 4 follow easily from Mager's definition of writing pushdown acceptor in the light of Theorem 1, which guarantees that the deterministic and nondeterministic versions of auxiliary PDM's bounded by storage $L(n) = n$ to have the same computing power. The equivalence between two-way multihead pushdown automata and $(\log n)$-bounded auxiliary PDM's follows from unpublished but fairly well-known proof techniques by Alan Cobham and others showing that two-way multihead finite-state machines are equivalent to $(\log n)$ tape-bounded multitape Turing machines.

The equivalences between stack automata and auxiliary PDM's remain to be demonstrated. These will follow immediately from the next three lemmas.

Lemma 2 *If a set A of strings can be recognized by a deterministic* [*nondeterministic*] *stack automaton, then A can be recognized by an auxiliary PDM within storage* $n \log n$ [*storage* n^2].

Proof. The argument is similar to the one used by Hopcroft and Ullman [HU67] to simulate nonerasing stack automata by tape-bounded Turing machines. Suppose S is an (erasing) stack automaton (deterministic or not) and let w be its input. At any point in the computation of S with input w the nonblank portion of the stack tape will be a string y. Following [HU67], we associate with each w and y a transition matrix $M_{w,y}$ defined as follows.

Let n be the length of the input w, and let N be the set of integers between 0 and $n+1$. Then $M_{w,y}$ is a binary relation on $(Q \times N) \cup \{A\}$. We say $M_{w,y}$ holds between (p,i) and (q,j) provided there is some partial computation of S in which initially S is in state p with the input head scanning symbol number i of the input w (counting symbol 0 as the blank to the left of w and symbol $n+1$ as the blank to the right of w) with y the nonblank portion of the stack tape and the stack head scanning the rightmost symbol of y, and finally the input head is scanning the position j of w and the stack head is scanning the blank immediately to the right of y. Further, throughout the partial computation the string y must remain unchanged, and except for the last step, the stack head must never leave the string y.

We say $M_{w,y}$ holds between (p,i) and A if there is some partial computation of S which satisfies the same initial conditions as above, but ends in S accepting w before the stack head leaves y.

We construct an auxiliary PDM X to simulate S as follows. X will be deterministic if S is deterministic, and sometimes nondeterministic if S is nondeterministic. Suppose the stack automaton S is in state q with input head scanning symbol i of the input w, and let $y = Y_1 Y_2 \cdots Y_k$ be the nonblank portion of the stack tape. If the stack head is scanning the leftmost blank on the stack tape (to the right of y), we say S is in a *regular configuration*. This regular configuration is represented in X by the pair $\langle q,i \rangle$ on a work tape and by the information $Y_1 M_{w,Y_1}, Y_2, M_{w,Y_1Y_2}, \ldots, Y_k, M_{w,y}$ on the pushdown tape (where the transition matrices $M_{w,Y_1 \ldots Y_j}$ are specified in some suitable notation).

Given a regular configuration occurring in a computation of S, X proceeds by finding the next regular configuration in the computation; or, in case S is nondeterministic, X will nondeterministically choose a possible next regular configuration for S.

Before S next assumes a regular configuration, the nonblank portion y of the stack tape may either (1) remain unchanged; or be changed in any of the following ways: (2) a new (nonblank) symbol Y_{k+1} is added to the right of y, resulting in

yY_{k+1}, (3) the right symbol of y is altered to a new nonblank symbol Y'_k, resulting in $Y_1Y_2\cdots Y_{k-1}Y'_k$, or (4) the right symbol of y is erased, resulting in $Y_1Y_2\cdots Y_{k-1}$.

In case (2), X must calculate the transition matrix $M_{w,yY_{k+1}}$ and add it to the right of its pushdown tape. X can easily calculate the new transition matrix from the symbol Y_{k+1} and the transition matrix $M_{w,y}$, which appears on the pushdown tape. The method is described in detail in [HU67]. In case (3), X must calculate $M_{w,Y_1\cdots Y'_k}$. This new transition matrix is calculated from Y'_k and $M_{w,y_1\cdots Y_{k-1}}$ (which appears on the pushdown tape immediately to the left of Y_k, $M_{w,y}$) using exactly the same method as for case (2). Finally, in case (4), X need only delete the pair Y_k, $M_{w,y}$ from its pushdown tape and update the pair $\langle q, i\rangle$ on one of its work tapes.

In any of the cases (1), (2), or (3), S may shift its stack head to the left of the rightmost nonblank symbol on its stack tape at some point before reaching the next regular configuration. Then, by the restrictions placed on the stack head in the definition in Section 3, the stack head cannot further change any symbols on the stack tape before reaching a regular configuration. Hence X can find the next state input position pair $\langle q, i\rangle$ (or the possible next pairs in the nondeterministic case) of S by referring to the transition matrix on the right of its pushdown tape. Of course X can also tell whether S can accept the input before the next regular configuration, and if S can accept, then X accepts.

The auxiliary (i.e. work tape) storage required by X for the simulation is a constant multiple of the number of squares required to store the largest transition matrix. In case S is deterministic, for each transition matrix $M_{w,y}$ and each pair (p, i) there is at most one possible pair (q, i) standing in the relation $M_{w,y}$ to (p, i) Hence the storage required to write down all pairs $\langle(p, i), (q, j)\rangle$ or $\langle(p, i), A\rangle$ satisfying $M_{w,y}$ is at most $cn \log n$ for some constant c, where n is the length of the input w. (The integers i and j are stored in binary notation.) In case S is nondeterministic, the possibilities for $M_{w,y}$ are increased. However, $M_{w,y}$ is always a subset of $(Q \times N) \times ((Q \times N) \cup \{A\})$, a set of at most c_1n^2 elements, where the constant c_1 depends only on the cardinality of the state set Q. Since a set of c_1n elements has $2^{c_1n^2}$ subsets, an arbitrary subset, and hence any transition matrix $M_{w,y}$, can be specified by using c_1n^2 tape squares, using any efficient notation. Details can be found in [HU67]. Also, the method for calculating the transition matrix $M_{w,yY}$ from $M_{w,y}$ and Y using at most the storage $c_2n \log n$ (if S is deterministic) or c_2n^2 (if S is nondeterministic) is described in [HU67]. The constants c, c_1, and c can be reduced to unity by standard methods.

Lemma 3 *If a set A of strings is accepted by some deterministic auxiliary PDM within storage $L(n) = n^2$, A is accepted by some nondeterministic stack automaton.*

Proof. The argument uses the same techniques as are used in [HU67] to show a nondeterministic nonerasing stack automaton can simulate a nondeterministic n^2 tape-bounded Turing machine.

Let X be a deterministic auxiliary PDM which accepts A within storage $L(n) = n^2$. Let us fix an input w for X. We will abbreviate the notation for configuration given in Section 4 in the proof (b) $\Rightarrow$ (c), and write $P = \langle q, i, u, Y\rangle$ instead of $P = iqu_1{\downarrow}v_1 * u_2{\downarrow}v_2 * \ldots * u_k{\downarrow}v_k * Y$, where u stands for $u_1{\downarrow}v_1 * \ldots * u_k{\downarrow}v_k$. Thus X is in the configuration $\langle q, i, u, Y\rangle$ provided X is in state q, with input head scanning symbol number i of w, with work tapes described by u, and pushdown head scanning the symbol Y.

We now show how a nondeterministic stack automaton S simulates X. Suppose at some point in its computation X is in configuration $\langle q, i, u, Y_k\rangle$, with $y = Y_1Y_2 \ldots Y_k$ equal to the contents of its pushdown tape (i.e. the string of symbols between s_0 on the left and the pushdown head on the right, including both ends). For each symbol Y_j on the pushdown tape, let $\phi(j)$ be the configuration of X the last time in the computation to date that the square on which Y_j is printed was scanned. Thus, regardless of the values of $Y_1, Y_2, \cdots, Y_{j-1}$, if $j < k$, then we know that when X is in the configuration $\phi(j)$, it will print Y_j on the pushdown tape and compute with its pushdown head to the right of this Y_j until at some point it is in the configuration $\langle q, i, u, Y_k\rangle$, with the symbols $Y_{j+1} \cdots Y_k$ written to the right of Y_j on the pushdown tape. Further, $\phi(k) = \langle q, i, u, Y_k\rangle$.

The stack tape of S encodes the computation of X to date as follows. The stack tape is divided into an upper channel and a lower channel. On the upper channel is written the sequence of configurations $\phi(1), \phi(2), \cdots, \phi(k)$, with a certain amount of "garbage" between adjacent entries. The garbage consists of obsolete configurations which are labeled with $*$'s and ignored by the machine. On the lower channel below each $\phi(j)$ is a configuration $\psi(j)$ which represents a guess by S as to the configuration of X the next time X scans square j of its pushdown tape.

We now describe how S updates its stack tape for each of the possible steps that X can take. Notice that S can determine what action X will next take by examining (without destroying) the top configuration $\phi(k)$ in its stack. The index i in the triple $\langle q, i, u, Y_k\rangle$ is stored in unary notation (i strokes) so that S can determine which symbol of the input string X is scanning.

(1) Suppose X prints a symbol on its pushdown tape and shifts its pushdown head right (and does various operations with its work heads and input head). Then S will compute the new configuration $\phi(k+1) = \langle q_1, i_1, u_1, \beta\rangle$ and write it on the upper channel of its stack tape to the right of $\phi(k)$. The method used by S to produce $\phi(k+1)$ to the right of $\phi(k)$ is exactly the one described in [HU67] that enables

a nondeterministic nonerasing stack automaton to write the next instantaneous description of an n^2 tape-bounded Turing machine next to the old one. While S is producing $\phi(k+1)$, it simultaneously writes on the lower channel below $\phi(k+1)$ an arbitrary configuration $\psi(k+1)$. $\psi(k+1)$ is chosen nondeterministically, and represents a guess as to the configuration of X the next time X scans square $k+1$ on its pushdown tape.

(2) Suppose X prints a symbol Y'_k on its pushdown tape but does not shift its pushdown head. Then S prints a $*$ to the right of $\phi(k)$ to label that configuration as obsolete (garbage), and then proceeds to write the new configuration

$$\phi'(k) = \langle q', i', u', Y'_k \rangle$$

to the right of $\phi(k)*$, using the method of the preceding paragraph. Simultaneously S guesses at the configuration $\psi'(k)$ and writes it on the lower channel below $\phi'(k)$.

(3) Suppose X prints a blank on its pushdown tape and shifts its pushdown head left. Then S searches left on its stack tape (without erasing) until it finds the rightmost configuration $\phi(k-1)$ which is to the left of $\phi(k)$ and which is not labeled obsolete with a $*$. On the lower channel below $\phi(k-1)$ is the configuration

$$\psi(k-1) = \langle q_2, i_2, u_2, Y'' \rangle$$

which represents the guess made earlier by S as to the present configuration of X. The stack automaton S now checks whether $\psi(k-1)$ is correct, by first checking whether Y'' is the symbol printed by X on its pushdown head when in the configuration $\phi(k-1)$. The remaining entries q_2, i_2, u_2 of $\psi(k-1)$ are checked against the rightmost configuration $\phi(k)$ on the stack to see if they are correct. The method of checking is exactly the reverse of the method mentioned in case (1) for producing $\phi(k+1)$ next to $\phi(k)$, and as each symbol (starting with the right) of $\psi(k-1)$ is checked, the corresponding symbol of $\phi(k)$ will be erased. If $\psi(k-1)$ is incorrect, the computation of S is terminated unsuccessfully (recall S is nondeterministic). If $\psi(k-1)$ is correct, then all garbage to the right of $\psi(k-1)$ is erased, the pair $\frac{\phi(k-1)}{\psi(k-1)}$ is labeled obsolete with a $*$, $\psi(k-1)$ is copied as $\phi'(k-1)$ on the upper channel immediately to the right of $\phi(k-1)$, and a new guess $\psi'(k-1)$ is written on the lower channel below $\phi'(k-1)$.

(4) If X accepts the input w, then S accepts the input w.

Lemma 4 *If a set A of strings is accepted by some deterministic auxiliary PDM within storage $L(n) = n \log n$, then A is accepted by a deterministic stack automaton.*

Proof. The argument is the same as the previous proof, with two exceptions. First, when the simulating deterministic stack automaton S produces the successor

$\phi(k+1)$ to a configuration $\phi(k)$ on its stack tape, it cannot use the same method as the nondeterministic stack automaton in the previous proof. The method used is that described in [HU67] for a deterministic nonerasing stack automaton to update an instantaneous description of an $n \log n$ storage bounded Turing machine. Since the auxiliary PDM being simulated has its work tapes bounded by $n \log n$, the new method works. Similarly, the new method is used by S for the other copy and ckeck operations on the configurations of X.

The second difference in the present argument is that the deterministic stack automaton S cannot guess nondeterministically at the configurations $\psi(k)$. Instead, S guesses systematically. We assume the possible configurations of the auxiliary PDM have a lexicographical ordering. Initially S sets $\psi(k)$ equal to the first configuration in the order, and writes this configuration on the lower channel below $\phi(k)$. In general, suppose S is updating its stack tape to correspond to a left shift of the pushdown head of x [see case (3) in the preceding proof]. If the latest guess $\psi(k-1)$ of the new configuration is incorrect, S erases all information to the right of the pair $\begin{smallmatrix}\phi(k-1)\\ \psi(k-1)\end{smallmatrix}$, labels that pair obsolete with a $*$, and copies over the pair on the space immediately to the right of the $*$, except $\psi(k-1)$ is replaced by the next configuration in the lexicographical order. Then the simulation of X is repeated, starting with the configuration $\phi(k-1)$. Sooner or later the right choice for $\psi(k-1)$ will be found, and the simulation can continue.

This completes the proof of Lemma 4 and of Theorem 2. As an immediate consequence of Theorem 2 and the definition of Cobham's class $\mathfrak{L}_*$ (see Section 2) we have the following.

Corollary 1 *A set A is in $\mathfrak{L}_*$ if and only if A is accepted by some two-way multihead pushdown automaton.*

As a second consequence, we can answer some open questions about the effect of determinacy.

Corollary 2 *In the case of both writing pushdown acceptors and two-way multihead pushdown automata, the deterministic and nondeterministic versions have the same computing power.*

For the next corollary, we need a result proved in [HS66].

Lemma 5 **Hennie-Stearns**
If $T_1(n)$ is a real-time countable function, then there is a set A of strings which is accepted by some deterministic multitape Turing machine within time $T_1(n)$, but by no such machine within time $T_2(n)$ for any function T_2 satisfying

$$\liminf_{n\to\infty} \frac{T_2(n)\log T_2(n)}{T_1(n)} = 0.$$

The next result answers an open question in [GGH67].

Corollary 3 *The class of sets accepted by deterministic two-way stack automata is properly included in the class accepted by nondeterministic such machines. Similarly, the classes defined by the machines in lines* 1, 3, *and* 5 *of Table 1 form an increasing sequence with proper inclusions.*

Proof. This follows easily from Lemma 5 and Theorem 2. For example, if we set $T_1(n) = 2^{n^2}$, and $T_2(n) = n^{n \log n}$, then $T_2(n)$ eventually dominates n^{cn} for every constant c, and hence every set A accepted by some deterministic stack automaton is accepted by some Turing machine within time $T_2(n)$. But then by Lemma 5, there is some set A which is accepted by a Turing machine within time $T_1(n)$, and hence by a nondeterministic stack automaton, but A is accepted by no Turing machine within time $T_2(n)$ and hence by no deterministic stack automaton.

The next corollary was first proved in [KB67], by a direct but complex argument.

Corollary 4 **Knuth-Bigelow**
The class of context-sensitive languages is properly included in the class of sets accepted by deterministic two-way stack automata.

Proof. All context-sensitive languages are accepted by nondeterministic Turing machines which operate with storage bounded by a linear function of the length of the input. But these machines are clearly special cases of writing pushdown acceptors (row 3 of Table 1), so all context-sensitive languages are accepted by writing pushdown acceptors. Corollary 4 now follows from Corollary 3.

The next two corollaries state for nondeterministic Turing machines facts which are obvious for deterministic Turing machines.

Corollary 5 *If a set A of strings is accepted by some nondeterministic Turing machine within storage* $L(n) \geq \log_2 n$, *then A is accepted by a (deterministic) time-bounded computer within time* $T(n) = 2^{cL(n)}$ *for some constant c.*

This follows directly from Theorem 1, by dropping the pushdown tape from the auxiliary PDM. In particular, we have for the case $L(n) = n$:

Corollary 6 *Every context-sensitive language is accepted by some deterministic Turing machine within time* 2^{cn}, *for some constant c.*

Corollary 7 *The class of sets accepted by each machine type of Table 1 is closed under union, intersection, and taking complements.*

This is clear from the characterizations given in the third column of the table. Since each of the functions there is real-time countable (i.e. suitably easy to

compute), a Turing machine accepting a set in any of the classes can be made to halt on all inputs. The corollary follows easily.

6 Conclusion and Open Questions

The major open question concerning these results is whether the pushdown tape really adds to the computing power of auxiliary PDM's. That is, we cannot show that Theorem 1 becomes false if "auxiliary PDM" is replaced by "Turing machine" in parts (a) and (b). This modified Theorem 1 would assert the converse of Corollary 5; that is, a set is accepted by a time-bounded computer within time $2^{cL(n)}$ if and only if it is accepted by a Turing machine within storage $L(n)$. Such a general equivalence between time and storage appears unlikely, but we have no proof.

A second group of questions which remain open concerns the two-way pushdown automaton described in [GHI67]. This device is easily characterized as an auxiliary PDM operating with zero auxiliary storage (i.e. no work tapes). Since the auxiliary storage is less than $\log_2 n$, Theorem 1 does not apply, and such questions as whether the nondeterministic version is more powerful than the deterministic version, and whether the class of sets accepted by either version is closed under complements, remain unanswered.

A final long-standing problem in the field of computational complexity is to prove that some interesting set (or function) must take a long time to compute on a reasonable general computer model. More specifically, no one can find any set not in Cobham's class $\mathfrak{L}_*$, except artificial examples through use of diagonal arguments. It is, however, easy to find plausible candidates, such as the set of binary notations for the primes. It seems to me that, because of the simple nature of multihead pushdown machines, Corollary 1 in Section 5 might provide a first step toward finding something interesting not in $\mathfrak{L}_*$.

Acknowledgment

The author wishes to thank Walter Savitch for many fruitful conversations concerning the results.

References

[AHU68] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. September. 1968. Time and tape complexity of pushdown automaton languages. *Inform. Contr.* 13, 3, 186–206. DOI: https://doi.org/10.1016/S0019-9958(68)91087-5.

[Cob65] A. Cobham. 1965. The intrinsic computational difficulty of functions. In *Proceedings of the 1964 International Congress for Logic, Methodology, and Philosophy of Science*. North-Holland, Amsterdam, 24–30.

[Col66] S. N. Cole. October. 1966. Real-time computation by n-dimensional iterative arrays of finite-state machines. In *IEEE Conf. Record of 1966 Seventh Annual Symposium on Switching and Automata Theory*, 53–77.

[Coo69] S. A. Cook. May 5–7. 1969. Variations on pushdown machines (detailed abstract). In *Proceedings of the 1st Annual ACM Symposium on Theory of Computing*. Marina del Rey, CA. ACM, 229–231. DOI: https://doi.org/10.1145/800169.805437.

[Ear70] J. Earley. February. 1970. An efficient context-free parsing algorithm. *Comm. ACM* 13, 2, 94–102.

[GGH67] S. Ginsburg, S. A. Greibach, and M. A. Harrison. January. 1967. Stack automata and compiling. *J. ACM* 14, 1, 172–201. DOI: https://doi.org/10.1145/321371.321385.

[GHI67] J. Gray, M. A. Harrison, and O. H. Ibarra. July, August. 1967. Two-way pushdown automata. *Inform. Contr.* 11, 1, 2, 30–70. DOI: https://doi.org/10.1016/S0019-9958(67)90369-5.

[HI68] M. A. Harrison and O. H. Ibarra. November. 1968. Multitape and multihead pushdown automata. *Inform. Contr.* 13, 5, 433–470. DOI: https://doi.org/10.1016/S0019-9958(68)90901-7.

[HS65] J. Hartmanis and R. E. Stearns. 1965. On the computational complexity of algorithms. *Trans. Am. Math. Soc.* 117, 285–306. DOI: https://doi.org/10.2307/1994208.

[HS66] F. C. Hennie and R. E. Stearns. October. 1966. Two-tape simulation of multitape Turing machines. *J. ACM* 13, 4, 533–546. DOI: https://doi.org/10.1145/321356.321362.

[HU67] J. E. Hopcroft and J. D. Ullman. August. 1967. Nonerasing stack automata. *J. Comput. Syst. Sci.* 1, 2, 166–186. DOI: https://doi.org/10.1016/S0022-0000(67)80013-8.

[KB67] D. E. Knuth and R. H. Bigelow. October. 1967. Programming languages for automata. *J. ACM* 14, 4, 615–635. DOI: https://doi.org/10.1145/321420.321421.

[Mag69] G. Mager. 1969. Writing pushdown acceptors. *J. Comput. Syst. Sci.* 3, 3, 276–319. DOI: https://doi.org/10.1016/S0022-0000(69)80017-6.

[SS63] J. C. Shepherdson and H. E. Sturgis. April. 1963. Computability of recursive functions. *J. ACM* 10, 2, 217–255. DOI: https://doi.org/10.2307/2271277.

The Relative Efficiency of Propositional Proof Systems

Stephen A. Cook and Robert A. Reckhow

1 Introduction

We are interested in studying the length of the shortest proof of a propositional tautology in various proof systems as a function of the length of the tautology. The smallest upper bound known for this function is exponential, no matter what the proof system. A question we would like to answer (but have not been able to) is whether this function has a polynomial bound for some proof system. (This question is motivated below.) Our results here are relative results.

In Sections 2 and 3 we indicate that all standard Hilbert type systems (or Frege systems, as we call them) and natural deduction systems are equivalent, up to application of a polynomial, as far as minimum proof length goes. In Section 4 we introduce *extended Frege* systems, which allow introduction of abbreviations for formulas. Since these abbreviations can be iterated, they eliminate the need for a possible exponential growth in formula length in a-proof, as is illustrated by an example (the pigeon-hole principle). In fact, Theorem 4.6 (which is a variation of a theorem of Statman) states that with a penalty of at most a linear increase in the number of lines of a proof in an extended Frege system, no line in the proof need be more than a constant times the length of the formula proved. The most difficult

Originally published in Cook, S., & Reckhow, R. (1979). The relative efficiency of propositional proof systems. The Journal of Symbolic Logic Volume 44, Number 1, March 1979.
1979, Association for Symbolic Logic. https://www.cambridge.org/core/journals/journal-of-symbolic-logic/article/abs/div-classtitlethe-relative-efficiency-of-propositional-proof-systemsdiv/218048250981F835B4B2A4080205A0BA

result is Theorem 4.5, which states that all extended Frege systems, regardless of which set of connectives they use, are about equivalent, as far as minimum proof length goes. Finally, in Section 5 we discuss the substitution rule, and show that Frege systems with this rule can simulate extended Frege systems.

Some of our results here appeared earlier in the conference proceedings [CR74], and Reckhow's Ph. D. thesis [Rec76]. (These two papers also establish and report nonpolynomial lower bounds on some proof systems more restricted than the ones mentioned above.)

To motivate the study of propositional proof systems, let us briefly review some of the theory of $\mathscr{P}$ and $\mathscr{NP}$ (see [Coo71b, Kar72], and Chapter 10 of [AHU74]). By convention, $\mathscr{P}$ denotes the class of sets of strings recognizable by a deterministic Turing machine in time bounded by a polynomial in the length of the input. $\mathscr{NP}$ is the same for nondeterministic Turing machines. If we let TAUT denote the set of tautologies over any fixed adequate set of connectives, then the main theorem in [Coo71b] implies that $\mathscr{P} = \mathscr{NP}$ if and only if TAUT is in $\mathscr{P}$. Now $\mathscr{P} = \mathscr{NP}$ not only would imply the existence of relatively fast algorithms for many interesting and apparently unfeasible combinatorial algorithms in $\mathscr{NP}$ (see [Kar72]), it would also have an interesting philosophical consequence for mathematicians. If $\mathscr{P} = \mathscr{NP}$, then there is a polynomial p and an algorithm $\mathscr{A}$ with the following property. Given any proposition S of set theory and any integer n, $\mathscr{A}$ determines within only $p(n)$ steps whether S has a proof of length n or less in (say) Zermelo-Fraenkel set theory. To see that the existence of $\mathscr{A}$ follows from $\mathscr{P} = \mathscr{NP}$, observe that the problem solved by $\mathscr{A}$ is in $\mathscr{NP}$. In fact, a nondeterministic Turing machine can write any string of length n on its tape and then verify that the string is a proof of the given proposition. For any reasonable logical theory, this verification can be performed within time bounded by some polynomial in n.

Hence the importance of showing $\mathscr{P} \neq \mathscr{NP}$ (or $\mathscr{P} = \mathscr{NP}$?). A related important question is whether $\mathscr{NP}$ is closed under complementation, i.e. $\Sigma^* - L$ is in $\mathscr{NP}$ whenever L is in $\mathscr{NP}$. (Here we use the notation Σ^* for the set of all finite strings over the finite alphabet Σ under consideration, and the assumption $L \subseteq \Sigma^*$. This notation will be used throughout.) If $\mathscr{NP}$ is not closed under complementation, then of course $\mathscr{P} \neq \mathscr{NP}$. On the other hand, if $\mathscr{NP}$ is closed under complementation, this would have interesting consequences for each of the combinatorial problems in [Kar72]. Hence the following result is important.

Proposition 1.1 *$\mathscr{NP}$ is closed under complementation if and only if* TAUT *is in $\mathscr{NP}$.*

Notation 1.2 $\mathscr{L}$ is the set of functions $f\colon \Sigma_1^* \to \Sigma_2^*$, Σ_1, Σ_2 any finite alphabets, such that f can be computed by a deterministic Turing machine in time bounded by a polynomial in the length of the input.

Proof of Proposition 1.1. The complement of the set of tautologies is in $\mathcal{NP}$, since to verify that a formula is not a tautology one can guess at a truth assignment and verify that it falsifies the formula. Conversely, suppose the set of tautologies is in $\mathcal{NP}$. By the proof of the main theorem in [Coo71b], every set L in $\mathcal{NP}$ is reducible to the complement of the tautologies in the sense that there is a function f in $\mathcal{L}$ such that for all strings x, $x \in L$ iff $f(x)$ is not a tautology. Hence a nondeterministic procedure for accepting the complement of L is: on input x, compute $f(x)$, and accept x if $f(x)$ is a tautology, using the nondeterministic procedure for tautologies assumed above. Hence the complement of L is in $\mathcal{NP}$. ∎

The question of whether TAUT is in $\mathcal{NP}$ is equivalent to whether there is a propositional proof system in which every tautology has a short proof, provided "proof system" and "short" are properly defined.

Definitions 1.3 If $L \subseteq \Sigma^*$, a *proof system* for L is a function $f\colon \Sigma_1^* \to L$ for some alphabet Σ_1 and f in $\mathcal{L}$ such that f is onto. We say that the proof system is *polynomially bounded* iff there is a polynomial $p(n)$ such that for all $y \in L$ there is $x \in \Sigma_1^*$ such that $y = f(x)$ and $|x| \leq p(|y|)$, where $|z|$ denotes the length of a string z.

If $y = f(x)$, then we will say that x is a *proof* of y, and x is a *short* proof of y if in addition $|x| = p(|y|)$. Thus a proof system f is polynomially bounded iff there is a bounding polynomial $p(n)$ with respect to which every $y \in L$ has a short proof.

Proposition 1.4 *A set L is in $\mathcal{NP}$ iff $L = \emptyset$ or L has a polynomially bounded proof system.*

The analogous statement for recursive function theory is that L is recursively enumerable iff $L = \emptyset$ or L is the range of a recursive function. The proof of the present proposition is straightforward. If $L \in \mathcal{NP}$, then some nondeterministic Turing machine M accepts L in polynomial time. If $L \neq \emptyset$, we define f such that if x codes a computation of M which accepts y, then $f(x) = y$. If x does not code an accepting computation, then $f(x) = y_0$ for some fixed $y_0 \in L$. Then f is clearly a polynomially bounded proof system for L. Conversely, if f is a polynomially bounded proof system for L, then a fast nondeterministic algorithm for accepting L is, on input y, guess a short proof x of y and verify $f(x) = y$. ∎

Putting Propositions 1.1 and 1.4 together we see that $\mathcal{NP}$ is closed under complementation if and only if TAUT has a polynomially bounded proof system, in the general sense of Definition 1.3. It is easy to see (and is argued below) that any conventional proof system for tautologies can naturally be made to fit the definition of proof system in Definition 1.3. Although it is doubtful that every general proof system for TAUT is natural, nevertheless this general framework helps explain the motivating question of this paper: Are any conventional propositional proof systems polynomially bounded?

We cannot answer that question directly (except negatively for certain restricted systems: see [CR74] and [Rec76], and also [Tse70]), but at least we can put different proof systems into equivalence classes such that the answer is the same for equivalent systems. We conjecture that the answer is always no.

Definition 1.5 If f_1: $\Sigma_1^* \to L$ and f_2: $\Sigma_2^* \to L$ are proof systems for L, then f_2 *p-simulates* f_1 provided there is a function g: $\Sigma_1^* \to \Sigma_2^*$ such that g is in $\mathscr{L}$, and $f_2(g(x)) = f_1(x)$ for all x.

Thus g translates a proof x of y in the system f_1 into a proof $g(x)$ of y in f_2. It is easy to see, using the fact that $\mathscr{L}$ is closed under composition, that p-simulation is a transitive reflexive relation, so that its symmetric closure is an equivalence relation.

Proposition 1.6 *If a proof system f_2 for L p-simulates a polynomially bounded proof system f_1 for L, then f_2 is also polynomially bounded.*

This is an immediate consequence of the definitions of "proof system" and "polynomially bounded", and the fact that every function in $\mathscr{L}$ is bounded in length by a polynomial in the length of its argument. ∎

We close this section by establishing some notation and terminology specific for propositional proof systems which will be used in the rest of this paper. The letter κ will always stand for an adequate set of propositional connectives which are binary, unary, or nullary (have two, one, or zero arguments). *Adequate* here means that every truth function can be expressed by formulas built up from members of κ. A *formula* refers to a propositional formula built up in the usual way from atoms (propositional variables) and connectives from some set κ, using infix notation. (We speak of a formula *over* κ if its connectives are from κ.) If $A_1, \ldots, A_n, B$ are formulas, then we write $A_1, \ldots, A_n \models B$ if B is a logical consequence of $A_1, \ldots, A_n$ (i.e. every truth assignment satisfying $A_1 \ldots, A_n$ satisfies B). Each of our propositional proof systems will be defined relative to some connective set κ, and will be capable of proving all tautologies over κ by proofs using formulas over κ. A *derivation* (from zero or more lines called *hypotheses*) in such a system is a finite sequence of *lines,* ending in the line proved. A line is always a formula, except in the case of natural deduction systems (Section 3). Each line must either be a hypothesis, or *follow* from earlier lines by a rule of inference. (In case the rule itself has no hypothesis, the rule is an *axiom scheme.*) If the derivation has no hypothesis, it is called a *proof.*

Thus to specify a propositional proof system for our purposes, it is only necessary to specify κ, the definition of a line, and a finite set of rules of inference. To make this notion of proof system be an instance of our abstract Definition 1.3, we note first of all that formulas can be naturally regarded as strings over a finite alphabet. The only problem is that an atom itself must be regarded as a string (say the letter P followed by a string over $\{0, 1\}$) in order that there be an unlimited supply of

atoms. Then a proof π in the propositional system which is, say, a sequence of formulas, can naturally be regarded as a string over a finite alphabet which includes the comma as a separator symbol, as well as the symbols necessary to specify the formulas. The function f which abstractly specifies the system would be given by $f(\pi) = A$ if π proves A, and $f(\pi) = A_0$ for some fixed tautology A_0 if π is a string not corresponding to a proof in the system.

The notation $A_1, \ldots, A_n \vdash^{\pi}_{\mathscr{S}} B$ means that π is a derivation of B from hypotheses $A_1, \ldots, A_n$ in the proof system $\mathscr{S}$. (The notation $\vdash_{\mathscr{S}}$ means that there is some derivation π in the system $\mathscr{S}$.) We use the following notation for various length measures:

$l(A)$ is the number of occurrences of atoms and nullary connectives in a formula (or sequence) A.

$\lambda(\pi)$ is the number of lines in a derivation π.

$\rho(\pi) = \max_i l(A_i)$, if π is $(A_1, \ldots, A_n)$.

$|\pi|$ or $|A|$ is the length of π or A as a string.

2 Frege Systems

In the most usual propositional proof systems the rules of inference are formula schemes, and an instance of the scheme is obtained by applying a substitution to the scheme. We shall call such systems *Frege systems*, after Frege [Fre67].

Throughout this section we assume that all formulas are over some fixed adequate connective set κ. The following terms are defined relative to κ.

Definitions 2.1 If $D_1, \ldots, D_k$ are formulas and $P_1, \ldots, P_k$ are distinct atoms, then $\sigma = (D_1, \ldots, D_k)/(P_1, \ldots, P_k)$ is a *substitution*, and σA is the formula which results by simultaneously replacing P_i by $D_i, i = 1, \ldots, k$, in formula A. A *Frege rule* is a system of formulas $(C_1, \ldots, C_n)/D$, where $C_1, \ldots, C_n \models D$. If $n = 0$, the rule is an *axiom scheme.* For any substitution σ we say that σD *follows from* $\sigma C_1, \ldots, \sigma C_n$ by the rule$(C_1, \ldots, C_n)/D$. An *inference system* $\mathscr{F}$ is a finite set of Frege rules. The notions of *derivation* and the symbol $\vdash$ for $\mathscr{F}$ are defined as in the end of Section 1, where now a *line* in a derivation is a formula. By our condition on the definition of Frege rule, it is clear that if $A_1, \ldots, A_n \vdash_{\mathscr{F}} B$ then $A_1, \ldots, A_n \models B$.

Definitions 2.2 An inference system $\mathscr{F}$ is *implicationally complete* if $A_1, \ldots, A_n \vdash_{\mathscr{F}} B$ whenever $A_1, \ldots A_n \models B$. A *Frege system* is an implicationally complete inference system.

In fact, Frege's original system in [Fre67] does not fit the above definition, because it has axioms instead of axiom schemes, and tacitly includes the substitution rule (see Section 5). According to Church [Chu56, p. 158], the idea of axiom schemes used to replace the substitution rule is due to von Neumann [vNeu27].

If we modify Frege's system to be a Frege system, the result has connectives $\kappa = \{\neg, \supset\}$, and the rule

$$\frac{A, A \supset B}{B}$$

and the six axiom schemes

$$A \supset (B \supset A), \qquad (C \supset (B \supset A)) \supset ((C \supset B) \supset (C \supset A)),$$
$$(D \supset (B \supset A)) \supset (B \supset (D \supset A)), \qquad (B \supset A) \supset (\neg A \supset \neg B),$$
$$\neg\neg A \supset A, \qquad A \supset \neg\neg A.$$

Theorem 2.3 *For any two Frege systems $\mathscr{F}_1$ and $\mathscr{F}_2$ over κ there is a function f in $\mathscr{L}$ and constant c such that for all formulas $A_1, \ldots, A_n$, B and derivations π, if $A_1, \ldots, A_n \vdash^{\pi}_{\mathscr{F}_1} B$ then $A_1, \ldots, A_n \vdash^{f(\pi)}_{\mathscr{F}_2} B$, and $\lambda(f(\pi)) \leq c\lambda(\pi)$ and $\rho(f(\pi)) \leq c\rho(\pi)$. (See the end of Section 1 for notation.)*

Corollary 2.4 *Any two Frege systems over κ p-simulate each other. Hence one Frege system over κ is polynomially bounded iff all Frege systems over κ are.*

The corollary is an immediate consequence of the theorem and Proposition 1.6. Reckhow [Rec76] proves a generalization of the corollary to cover the case of Frege systems with different connective sets simulating each other, even when some of the connectives have arity greater than two. His proof is much more complicated than our proof of Theorem 2.3 given below, largely because of the difficulty of simulating systems using the connectives $\equiv$ and $\not\equiv$ by systems without these connectives. Fortunately, Corollary Theorem 4.6 below, concerning extended Frege systems, makes Corollary 2.4 and Reckhow's generalization less important than they might appear at first, since extended Frege systems seem to be more natural than Frege systems when measuring proof lengths.

The lemma below is used in the proof of Theorem 2.3. (The notation $\sigma(\pi)$ means $\sigma A_1, \ldots, \sigma A_k$, if π is a derivation $A_1, \ldots, A_k$.)

Lemma 2.5 *If π is a derivation of A from $B_1, \ldots, B_k$ in a Frege system $\mathscr{F}$, then $\sigma(\pi)$ is a derivation of σA from $\sigma B_1, \ldots, \sigma B_k$ in $\mathscr{F}$, for any substitution σ.*

The proof is an easy induction on the length of π. ∎

To prove Theorem 2.3, assume $\mathscr{F}_1$ and $\mathscr{F}_2$ are Frege systems over κ. For each rule $R = (C_1, \ldots C_m)/D$ in $\mathscr{F}_1$, let π_R be a derivation of D from $C_1, \ldots, C_m$ in $\mathscr{F}_2$. Now suppose π is a derivation of B from $A_1, \ldots, A_n$ in $\mathscr{F}_1$, and suppose $\pi = B_1, \ldots, B_k$. To construct the $\mathscr{F}_2$-derivation $f(\pi)$ from π, if B_i follows from earlier B_j's by the $\mathscr{F}_1$-rule R_i and substitution σ_i, simply replace B_i by the derivation $\sigma_i(\pi_{R_i})$ (with hypotheses deleted). According to Lemma 2.5, $\sigma_i(\pi_{R_i})$ is a derivation of B_i from

the same earlier B_j's. Clearly $\lambda(f(\pi)) \leq c_1\lambda(\pi)$, where c_1 is the number of lines in the longest derivation π_R, as R ranges over the finite set of rules of $\mathscr{F}_1$. Finally, $\rho(f(\pi)) \leq c_2\rho(\pi)$, where c_2 is an upper bound on $l(A)$ as A ranges over all formulas in all the derivations π_R, R a rule of $\mathscr{F}_1$. ∎

3 Natural Deduction Systems

The purpose of this section is to indicate the sense in which natural deduction systems are equivalent to Frege systems. Rather than presenting a specific natural deduction system, such as one appearing in Prawitz [Pra65], we shall introduce a general definition analogous to our general notion of Frege system. To make the classical proposition system of Prawitz fit our definition, it is necessary to allow Prawitz's notion of proof to be a more general directed acyclic graph, rather than a tree. That is, once a formula is derived from a set of assumptions, we do not require that it be derived again if it is used twice. Alternatively, we could stick to Prawitz's tree proofs, provided that if a formula occurred several times in a proof with the same assumptions, it be counted only once in measuring the length of the proof. In fact, we shall present our natural deduction proofs as sequences of lines, and each line will have the form $A_1, \ldots, A_n \to A$, where $A_1, \ldots A_n$ are assumptions which imply A. Thus our proofs require repeating the assumptions for a formula with each step, which makes them a little longer and harder to write down, but easier to analyze. For convenience, we allow only the right-most formula A_n to be discharged. Reckhow [Rec76] gives a more general treatment of natural deduction systems, as well as Gentzen's sequent systems.

Part of the appeal of a natural deduction system is that it allows the "deduction theorem" to be used as a rule. According to the deduction theorem, from a derivation π in a Frege system $\mathscr{F}$ showing $A_1, \ldots, A_m \vdash B$ we can construct a derivation π' in $\mathscr{F}$ showing $A_1, \ldots, A_{m-1} \vdash A_m \supset B$. The trouble is that π' may be twice as long as π, so that if a natural deduction derivation has m nested uses of this deduction rule and they are eliminated sequentially to obtain a Frege derivation, the result might be longer by a factor of 2^m than the original derivation. Fortunately, they can be eliminated simultaneously, as shown by the construction fr($\mathscr{N}$) below.

The following definitions are relative to a given adequate connective set κ.

Notation 3.1 Even if $\neg$ or $\vee$ is not in κ, formulas $N(P)$ and $O(P, Q)$ over κ can always be found such that $N(P)$ and $O(P, Q)$ are equivalent to $\neg P$ and $P \vee Q$, respectively, and such that P and Q each has at most one occurrence in each of $N(P)$ and $O(P, Q)$. A fixed "dummy" atom P_0 may occur several times, however. For example, if κ is $\{\ddagger, \supset\}$ then $N(P)$ could be $(P \ddagger (P_0 \supset P_0))$ and $O(P, Q)$ could be $((P \ddagger (P_0 \supset P_0)) \supset Q)$. (See Section 5. 3.1.1 of [Rec76] for an argument showing how this can be done in

general.) Thus we will take $\neg A$ or $A \vee B$ to mean $N(A)$ or $O(A, B)$, respectively, if $\neg$ or $\vee$ is not in κ. We use $\bigvee(A_1, \ldots, A_m)$ to stand for $(\cdots(A_1 \vee A_2)\cdots \vee A_m)$ (association to the left), and $\bigvee'(A_1, \ldots, A_m)$ to stand for $(A_1 \vee \cdots(A_{m-1} \vee A_m)\cdots)$ (association to the right).

Definitions 3.2 A *naturnal deduction line* (or just *line*) is a pair $\Gamma \rightarrow A$, where Γ is any finite sequence of formulas, and A is a formula. If Γ is empty, the line is written simply $\rightarrow A$. Associated with a line $L = (A_1, \ldots, A_m) \rightarrow A$ are two equivalent formulas $L^* = \bigvee(\neg A_1, \ldots, \neg A_m, A)$ and $L^\# = \bigvee'(\neg A_1, \ldots, \neg A_m, A)$. (If $m = 0$, the $L^* = L^\# = A$.) The line L takes on the same truth value under a truth assignment as formulas L^* and $L^\#$, so that the concepts of validity, logical consequence, etc. are well defined for lines. If Δ is a sequence $B_1, \ldots, B_n$ of formulas and L is the line $(A_1, \ldots, A_m) \rightarrow A$, then ΔL is the line $(B_1, \ldots, B_n, A_1, \ldots, A_m) \rightarrow A$. If Λ is a set of lines, L is a line, Δ is a sequence of formulas, and σ is a substitution, then $\Lambda \models L$ implies that $\Delta\sigma(\Lambda) \models \Delta\sigma(L)$, where the operations Δ and σ are extended to sets of lines in the natural way. If Λ is a finite set of lines and L is a line such that $\Lambda \models L$, then the system $R = \Lambda/L$ is a *natural deduction rule.* Line L' *follows* from Λ' by rule R provided for some substitution σ and sequence Δ, $\Lambda' = \Delta\sigma(\Lambda)$, and $L' = \Delta\sigma(L)$. A *natural deduction system* is a finite set of natural deduction rules which is implicationally complete (implicationally complete being defined in a manner analogous to that for Frege systems). A formula A is represented in a natural deduction system $\mathcal{N}$ by the line $\rightarrow A$. This convention allows us to speak of proofs of formulas and derivations of a formula from formulas in $\mathcal{N}$, and thus write for example $A_1, \ldots, A_n \vdash^\pi_{\mathcal{N}} B$ instead of $\rightarrow A_1, \ldots, \rightarrow A_n \vdash^\pi_{\mathcal{N}} \rightarrow B$.

If $L = (A_1, \ldots, A_k) \rightarrow A$ is a line, then $l(L) = l(A_1) + \cdots + l(A_k) + l(A)$. If π is a derivation, then $\lambda(\pi)$ is the number of lines in π, and $\rho(\pi)$ is the maximum of $l(L)$, for all L in π.

An example of a natural deduction rule, which embodies the deduction theorem, is $R_1 = (P \rightarrow Q)/(\rightarrow \neg P \vee Q)$. This rule together with its converse $R_2 = (\rightarrow \neg P \vee Q)/(P \rightarrow Q)$ can turn any Frege system $\mathcal{F}$ into a natural deduction sytem $\mathrm{nd}(\mathcal{F})$, provided we reinterpret every rule $R = (C_1, \ldots, C_n)/D$ of the Frege system to be $R' = (\rightarrow C_1, \ldots, \rightarrow C_n)/ \rightarrow D$. In fact, if $\Lambda \models L$, then to deduce L from Λ in $\mathrm{nd}(\mathcal{F})$, we first observe that every hypothesis M in Λ can be changed to $\rightarrow M^\#$ by repeated use of the rule R_1. By the implicational completeness of $\mathcal{F}$, we can derive $\rightarrow L^\#$ in $\mathrm{nd}(\mathcal{F})$ from these lines $\rightarrow M^\#$. Now L can be derived from $\rightarrow L^\#$ by repeated use of the rule R_2.

Notice that every derivation in $\mathcal{F}$, of say B from $A_1, \ldots, A_n$, can be turned into a derivation of B from $A_1, \ldots, A_n$ in $\mathrm{nd}(\mathcal{F})$ simply by adding the symbol $\rightarrow$ to the left of every formula in the derivation.

Conversely, every natural deduction system $\mathcal{N}$ can be turned into a Frege system fr($\mathcal{N}$), where the rules of fr($\mathcal{N}$) consist of the two rules R' and R'' for every rule R of $\mathcal{N}$. To explain R' and R'' we need to recall the notation M^* for $\bigvee(\neg A_1, \ldots, \neg A_m, A)$ and introduce the notation $(PM)^*$ for $\bigvee(P, \neg A_1, \ldots, \neg A_m, A)$, where M is a line $(A_1, \ldots, A_m) \to A$ and P is an atom. If $R = \Lambda/L$, then $R' = \Lambda^*/L^*$ and $R'' = (P\Lambda)^*/(PL)^*$, where P is some atom not occurring in Λ or L, and we have extended the * notation to sets Λ of lines in the obvious manner. It is easy to see that the rules R' and R'' are sound if R is sound.

Now if $\pi = L_1, \ldots, L_n$ is any derivation in $\mathcal{N}$, then we claim that $\pi^* = L_1^*, \ldots, L_n^*$ is a derivation in fr($\mathcal{N}$). For suppose L_i follows from earlier L_j's by the the rule $R = \Lambda/L$ in $\mathcal{N}$. Then for some substitution σ and sequence Δ, L_i is $\Delta\sigma(L)$ and the earlier L_j*'s* comprise the set $\Delta\sigma(\Lambda)$. If Δ is empty, the L_i^* follows from earlier L_j^*'s by the Frege rule $R' = \Lambda^*/L^*$ by σ, since for any line M, $(\sigma(M))^* = \sigma(M^*)$. If Δ is not empty, then L_i^* follows from earlier L_j^*'s by the Frege rule $R'' = (P\Lambda)^*/(PL)^*$ and substitution σ', where σ' is the substitution obtained by simultaneously applying the substitution σ and $\bigvee(\neg A_1, \ldots, \neg A_k)/P$, where Δ is $(A_1, \ldots, A_k)$. We need the fact that for any line M with no occurrence of P, $\sigma'((PM)^*) = (\Delta\sigma(M))^*$.

Thus π^* is a derivation in fr($\mathcal{N}$) for every derivation π in $\mathcal{N}$. Notice that since $(\to A)^* = A$, if π is a derivation in $\mathcal{N}$ of B from $A_1 \ldots, A_l$ then π^* is a derivation in fr($\mathcal{N}$) of B from $A_1, \ldots, A_l$. Further, notice that $\lambda(\pi^*) = \lambda(\pi)$ and $\rho(\pi^*) \leq c\rho(\pi)$, where the constant c depends only on the underlying connective set κ.

Although the constructions above allow us to translate back and forth between Frege and natural deduction systems, the following result still needs a separate proof.

Theorem 3.3 *Given natural deduction systems $\mathcal{N}_1$ and $\mathcal{N}_2$ over κ there is a function f in $\mathcal{L}$ and a constant c such that for all lines $L_1, \ldots, L_n, L$ and derivations π, if $L_1, \ldots, L_n \vdash^{\pi}_{\mathcal{N}_1} L$, then $L_1, \ldots, L_n \vdash^{f(\pi)}_{\mathcal{N}_2} L$, and $\lambda(f(\pi)) \leq c\lambda(\pi)$ and $\rho(f(\pi)) \leq c\rho(\pi)$.*

The proof is very similar to the proof of Theorem 2.3. Lemma 2.5 is replaced by the statement that if π is a derivation in $\mathcal{N}$ of line M from lines $M_1, \ldots, M_k$, then $\Delta\sigma(\pi)$ is a derivation of $\Delta\sigma(M)$ from $\Delta\sigma(M_1), \ldots, \Delta\sigma(M_k)$. ∎

Corollary 3.4 *Let κ be any adequate set of connectives. All Frege and natural deduction systems over κ p-simulate all other Frege and natural deduction systems over κ. Hence one such system over κ is polynomially bounded if and if all such systems over κ are polynomially bounded.*

The corollary follows immediately from Theorems 2.3 and 3.3, together with the constructions nd($\mathcal{F}$) and fr($\mathcal{N}$) given above. ∎

Reckhow [Rec76] treats a kind of natural deduction system in which Γ in a line $\Gamma \to A$ is regarded as a set of formulas rather than a sequence of formulas. Such a system might allow for shorter proofs, since in effect there are implicit rules which allow Γ to be reordered. In [Rec76] it is shown that the above corollary holds for this system, and that the second part holds even when the systems have different connective sets.

The corollary also holds for Gentzen systems with cut, provided a Gentzen proof is considered to be a sequence of sequents, so that a given occurrence of a sequent can be used more than once in a proof, as opposed to the more usual definition that a Gentzen proof is a tree of sequents. When a Gentzen proof is defined to be a tree, an exponential lower bound for the number of sequents in a minimum cut-free proof of a formula follows from an unpublished result of Statman. More recently, Cook and Rackoff have an unpublished result showing an exponential lower bound for Gentzen proofs considered as sequences, provided both the cut and thinning rules are disallowed.

4 Extended Frege Systems

The previous sections have indicated that certain standard proof systems for the propositional calculus are about equally powerful. We now look for natural extensions of these systems which might be more powerful, in the sense that they yield shorter proofs. To motivate this search, we try to use Frege systems to simulate an informal proof of the "pigeon-hole principle".

One statement of the pigeon-hole principle is that no injective function maps $\{1, 2, \ldots, n\}$ to $\{1, 2, \ldots, n-1\}$, $n \geq 2$. For each value of n, this statement may be formalized in the propositional calculus as follows. Let $P_{ij}, 1 \leq i \leq n, 1 \leq j \leq n-1$, be a set of atoms, whose intended meaning is "i is mapped to j". Let $\mathscr{S}_n$ be the set (or sometimes the conjunction of the formulas in the set) $\{P_{i1} \vee \cdots \vee P_{i,n-1} | 1 \leq i \leq n\}$ $\bigcup \{\neg P_{ik} \vee \neg P_{jk} | 1 \leq i < j \leq n, 1 \leq k \leq n-1\}$. If a truth assignment were given for which each formula in $\mathscr{S}_n$ is true then one could define a function f which by the first set of disjunctions is from $\{1, 2, \ldots, n\}$ to $\{1, 2, \ldots, n-1\}$ and which by the second set is injective. Thus the formula $A_n = \neg \mathscr{S}_n$ is a tautology.

An informal proof of the pigeon-hole principle proceeds by induction on n. It is obvious for $n = 2$. In general, if f: $\{1, \ldots, n\} \to \{1, \ldots, n-1\}$, then let f': $\{1, \ldots, n-1\} \to \{1, \ldots, n-2\}$ be defined by $f'(i) = f(i)$ if $f(i) \neq n-1$; otherwise $f'(i) = f(n)$. If f is injective, it is easy to see that f' is also, contradicting the induction hypothesis.

To mimic this proof in a Frege system, we try to deduce $\mathscr{S}_{n-1}$ from $\mathscr{S}_n$. For each i, j, we introduce a formula B_{ij} which means $f'(i) = j$. $B_{ij} = P_{ij} \vee (P_{i,n-1} \,\&\, P_{nj})$, $1 \leq i \leq n-1, 1 \leq j \leq n-2$. Let σ_{n-1} be the substitution B_{ij}/P_{ij} $(1 \leq i \leq n-1, 1 \leq j \leq n-2)$.

The argument that f injective implies f' injective shows $\mathscr{S}_n \models \sigma_{n-1}(\mathscr{S}_{n-1})$. By completeness, $\mathscr{S}_n \vdash \sigma_{n-1}(\mathscr{S}_{n-1})$. Similarly, $\mathscr{S}_{n-1} \vdash \sigma_{n-2}(\mathscr{S}_{n-2})$, so by Lemma 2.5, there is a derivation of the same number of lines showing $\sigma_{n-1}(\mathscr{S}_{n-1}) \vdash \sigma_{n-1}\sigma_{n-2}(\mathscr{S}_{n-2})$, so $\mathscr{S}_n \vdash \sigma_{n-1}\sigma_{n-2}(\mathscr{S}_{n-2})$. Proceeding this way, we finally obtain a derivation showing $\mathscr{S}_n \vdash \sigma_{n-1}\cdots\sigma_2(\mathscr{S}_2)$. But $\mathscr{S}_2$ is $\{P_{11}, P_{21}, \neg P_{11} \vee \neg P_{21}\}$, from which a contradiction is easily derived, so by the deduction theorem, $\vdash \neg\mathscr{S}_n$; i. e. $\vdash A_n$.

It is not hard to see that by choosing the rules of our Frege system conveniently, the derivation of $\sigma_{n-1}(\mathscr{S}_{n-1})$ from $\mathscr{S}_n$ has $O(n^3)$ lines. Hence the entire proof of A_n has $O(n^4) = O(N^{4/3})$ lines, where N is $|A_n|$. On the other hand, each application of a substitution σ_i triples the length of a formula, so the longest formulas in the proof of A_n grow exponentially in n.

A simple device to reduce the formula length in the above proof is to introduce new atoms which abbreviate the formulas B_{ij}. Thus the atom Q^1_{ij} has a defining formula $Q^1_{ij} \equiv (P_{ij} \vee (P_{i,n-1} \,\&\, P_{nj}))$, $1 \le i \le n-1$, $1 \le j \le n-2$. From these defining formulas and the formulas $\mathscr{S}_n$, the formulas $\tau_{n-1}(\mathscr{S}_{n-1})$ are easily derived, where τ_{n-1} is the substitution Q^1_{ij}/P_{ij} $(1 \le i \le n-1, 1 \le j \le n-2)$. In general, a new atom Q^{k+1}_{ij} is introduced for $\sigma_{n-1}\cdots\sigma_{n-k}(B_{ij})$ with defining formula $Q^{k+1}_{ij} \equiv (Q^k_{ij} \vee (Q^k_{i,n-k-1} \,\&\, Q^k_{n-k,j}))$, and the formulas $\tau_{n-k-1}(\mathscr{S}_{n-k-1})$ are easily derived from these defining formulas and the formulas $\tau_{n-k}(\mathscr{S}_{n-k})$ where τ_{n-k} is the substitution Q^k_{ij}/P_{ij} $(1 \le i \le n-k, 1 \le j \le \mathrm{n}-k-1)$. In this way, a contradiction is derived from $\mathscr{S}_n$ in $O(n^4)$ lines, where now each formula has length only $O(n)$. Hence A_n has a proof of length $O(n^5)$ in this framework. This kind of proof system can be formalized as follows:

Definition 4.1 An *extended Frege system* over a connective set κ is a proof system which consists of a Frege system $\mathscr{F}$ over κ together with the *extension rule* which allows formulas of the form $P \equiv A$ to be added to a derivation, where A is any formula over κ, and P is any "new" atom. (P must not occur in A, in any lines preceding $P \equiv A$, or in any hypotheses to the derivation. P can occur in later lines, but not in the last line.) We say P is a *defined* atom and $P \equiv A$ is its *defining formula.* If $\equiv$ is not in κ, we choose some short formula $P \sim Q$ over κ which is equivalent to $P \equiv Q$, and let $P \sim A$ be the defining formula for P. The extended Frege system based on $\mathscr{F}$ is denoted by $e\mathscr{F}$.

(The extension rule was first suggested by Tseitin [Tse70], in the context of resolution proofs.)

Proposition 4.2 **Soundness of e$\mathscr{F}$**
If $A_1, \ldots, A_n \vdash_{e\mathscr{F}} B$, *then* $A_1, \ldots, A_n \models B$.

Proof. Let τ be any truth assignment to the atoms of $A_1, \ldots, A_n$ and B which satisfies $A_1, \ldots, A_n$. Then τ can be extended to make each line in the derivation true. In

particular, if $P \equiv A$ is a defining formula, then P has not occurred earlier in the derivation, so we are free to extend τ so $\tau(P) = \tau(A)$. Hence $\tau(B)$ is true, since B is the last line of the derivation. ∎

Although the extension rule apparently allows the lengths of formulas in a derivation to be greatly reduced, the following result shows the number of lines in a proof cannot be much reduced.

Proposition 4.3 *If π is a derivation of B from $A_1, \ldots, A_n$ in $e\mathscr{F}$, then there is a derivation π' of B from $A_1, \ldots, A_n$ in $\mathscr{F}$ with $\lambda(\pi') \leq \lambda(\pi) + cm$ where c depends only on $\mathscr{F}$, and m is the number of defining formulas in π.*

Proof. Suppose $P_i \sim C_i, 1 \leq i \leq m$, are the defining formulas in π (given in the order in which they occur in π). Then π is a derivation in $\mathscr{F}$ of B from $A_1, \ldots, A_n$, $P_1 \sim C_1, \ldots, P_m \sim C_m$. Now let σ be the composed substitution

$$\frac{C_m}{P_m} \circ \frac{C_{m-1}}{P_{m-1}} \circ \cdots \circ \frac{C_1}{P_1}.$$

By Lemma 2.5, $\sigma(\pi)$ is a derivation of σB from $\sigma A_1, \ldots, \sigma A_n, \sigma(P_1 \sim C_1), \ldots, \sigma(P_m \sim C_m)$. By the restrictions on the defined atoms P_i, $\sigma(\pi)$ is a derivation in $\mathscr{F}$ of B from $A_1, \ldots, A_n, \sigma C_1 \sim \sigma C_1, \ldots, \sigma C_m \sim \sigma C_m$. But $Q \sim Q$ has some fixed proof in $\mathscr{F}$ of some number of lines (say c lines), so by Lemma 2.5, each $\sigma C_i \sim \sigma C_i$ has a proof in $\mathscr{F}$ of c lines. Also $\lambda(\sigma(\pi)) = \lambda(\pi)$. Hence we construct π' from $\sigma(\pi)$ together with these m proofs, and the proposition follows. ∎

Of course the formulas of π' can grow exponentially in m, even if the formulas of π are short, as shown by the pigeon-hole example at the beginning of this section.

We mentioned that Reckhow [Rec76] strengthened Theorem 2.3 to cover the case of different connective sets, but the proof was complicated by the difficulties of finding a short translation for a formula containing $\equiv$ into one containing, say, just $\&$, $\vee$, and $\neg$. In the case of extended Frege systems, this difficulty can be circumvented. Theorem 4.5 below states that if the number of lines in the shortest proof of a tautology A is bounded by some function $L(l(A))$ in some extended Frege system, then essentially the same is true of any extended Frege system over any connective set, and furthermore the lengths of the formulas in a proof need not be much longer than the formula proved. (The latter is in sharp contrast to the apparent situation for Frege proofs without extension.)

Theorem 4.5 *Suppose $e\mathscr{F}$ and $e\mathscr{F}'$ are extended Frege systems over κ and κ', respectively, and suppose $L(n) \leq n$ is a natural number function such that every tautology A over κ has a proof π in $e\mathscr{F}$ with $\lambda(\pi) \leq L(l(A))$. Then every tautology A' over κ' has a proof π' in*

$e\mathscr{F}'$ such that $\lambda(\pi') \leq cL(cl(A'))$ and $\rho(\pi') \leq cl(A')$, where the constant c depends only on $\mathscr{F}$ and $\mathscr{F}'$.

Theorem 4.6 **Statman**[1]

For any extended Frege system $e\mathscr{F}$ and tautology A, if π is a proof of A in $e\mathscr{F}$, then there is a proof π' of A in $e\mathscr{F}$ such that $\lambda(\pi') \leq c(\lambda(\pi) + l(A))$ and $\rho(\pi') \leq cl(A)$, where the constant c depends only on $\mathscr{F}$.

Corollary 4.7 **To Theorem 4.5**

A given extended Frege system is polynomially bounded if and only if all extended Frege systems over all connective sets are polynomially bounded. Also, an extended Frege system $e\mathscr{F}$ is polynomially bounded if and only if there is a polynomial bound on the number of lines in proofs in $e\mathscr{F}$. Hence, if $\mathscr{P} \neq \mathscr{NP}$, then there is no polynomial bound on the number of lines in proofs in extended Frege systems, Frege systems, or (by Section 3) natural deduction systems.

Propositions Theorems 4.5, 4.6, and Corollary 4.7 are evidence that the extended Frege systems are a very natural class of proof system. Further evidence is provided by results in [Coo76b], which show that extended Frege system proofs can simulate the proof of any theorem of a certain number theory system PV. ("Simulate" here means something similar to the way in which extended Frege proofs simulate the proof of the pigeon hole principle in the example given at the beginning of this section.) The same paper [Coo76b] shows that extended Frege systems are the most efficient systems whose soundness is provable in PV.

The remainder of this section is devoted to proving Theorems 4.5 and 4.6. Let us start by showing that a bound on proof length in an extended Frege system gives us a bound on derivation length.

Lemma 4.8 *Suppose $e\mathscr{F}$ and $L(n)$ satisfy the hypotheses of Theorem 4.5. If $A_1, \ldots, A_m$, B are formulas over κ such that $A_1, \ldots, A_m \models B$, then there is a derivation π in $e\mathscr{F}$ of B from $A_1, \ldots, A_m$ with $\lambda(\pi) \leq cL(cn)$, where $n = l(A_1) + \cdots + l(A_m) + l(B)$, and c depends only on $\mathscr{F}$.*

Proof. Suppose first that the connective set κ of $\mathscr{F}$ contains $\vee$ and $\neg$. Since $A_1, \ldots, A_m \models B$, we have $\models (\neg A_1(\neg A_2 \vee \cdots \vee (\neg A_m) \vee B)\cdots))$. Hence this formula has a proof π' in $e\mathscr{F}$ with $\lambda(\pi') \leq L(n), n = l(A_1) + \cdots + l(A_m) + l(B)$. If we assume

1. After proving a version of Theorem 4.5 without the bound on $\rho(\pi')$ in course notes [CR+76], the first author received an earlier version of Statman [Sta77] and realized the proof in the notes could be strengthened to yield the present Theorems 4.5 and 4.6. Statman's theorem in [Sta77] has a more general setting than Theorem 4.6, but a weaker bound on $\lambda(\pi')$. The authors wish to thank Martin Dowd for helpful discussions concerning Theorem 4.6.

$\mathscr{F}$ has the cut rule

$$\frac{P, \neg P \vee Q}{Q}$$

then by appending m applications of this rule to π', we obtain a derivation π of B from $A_1, \ldots, A_m$ satisfying the lemma, with $\lambda(\pi) \leq 2L(n)$. If the cut rule is not in $\mathscr{F}$, then by Theorem 2.3 the rule can be simulated to produce a derivation π with $\lambda(\pi) \leq cL(n)$.

If $\vee$ or $\neg$ is not in κ, one can check that nevertheless there are formulas $O(P, Q)$ and $N(P)$ over κ equivalent to $P \vee Q$ and $\neg P$, respectively, such that $O(P, Q)$ and $N(P)$ have at most one occurrence each of P and Q (see Notation 3.1). In this case we obtain the bound $\lambda(\pi) \leq cL(cn)$. ∎

To prove Theorems 4.5 and 4.6 we need the notion of a defining set of formulas $\text{def}(A)$ for a formula A. We assume that every formula B (over any connective set) has associated with it an atom P_B such that P_Q is Q for any atom Q, and distinct nonatomic formulas have distinct associated atoms. To be definite, we could let P_B be the string consisting of the letter P followed by the string B, if B is nonatomic. In any case, we shall also assume for convenience later, that there are infinitely many atoms P, called *admissible* atoms, which are *not* of the form P_B for any nonatomic B.

Let us call a formula A *admissible* if all its atoms are admissible. If A is admissible, then every truth assignment τ to the atoms of A has a unique extension τ' to the atoms P_B, B any subformula of A, such that $\tau'(P_B) = \tau(B)$. We shall define $\text{def}(A)$ such that any extension τ'' of τ satisfies $\text{def}(A)$ iff τ'' agrees with τ' on the atoms P_B. For example, if A is $Q \vee (R \,\&\, S)$, then $\text{def}(A)$ might be $\{(P_{(R\&S)} \equiv (R \,\&\, S)), (P_A \equiv Q \vee P_{(R\&S)})\}$. In fact, it is useful to more generally define $\text{def}_\kappa(A)$, where κ is any adequate set of connectives, perhaps different from the set of connectives appearing in A.

Definition 4.9 Let κ_1 and κ_2 be connective sets. Corresponding to each nullary connective (constant) K_1 in κ_1 we associate a fixed formula K_2 over κ_2 equivalent to K_1; corresponding to each unary connective u_1 over κ_1 we associate a fixed formula u_2P over κ_2 equivalent to u_1P, and corresponding to each binary connective $\circ_1$ in κ_1 we associate a fixed formula $P \circ_2 Q$ over κ_2 equivalent to $P \circ_1 Q$. We assume the formulas $P \sim_1 Q$ over κ_1 and $P \sim_2 Q$ over κ_2 are each equivalent to $P \equiv Q$. For each formula A_1 over κ_1 we associate a set $\text{def}_{\kappa_2}(A_1)$ of formulas over κ_2 defined by induction on the length of A_1 as follows:

$\text{def}_{\kappa_2}(P) = \emptyset$ (the empty set) for each atom P.
$\text{def}_{\kappa_2}(K_1) = \{P_{K_1} \sim_2 K_2\}$ for each constant K_1 in κ_1.

$\mathrm{def}_{\kappa_2}(u_1A) = \mathrm{def}_{\kappa_2}(A) \cup \{P_{u_1A} \sim_2 u_2P_A\}$, for each unary connective u_1 in κ_1.

$\mathrm{def}_{\kappa_2}(A \circ_1 B) = \mathrm{def}_{\kappa_2}(A) \cup \mathrm{def}_{\kappa_2}(B) \cup \{P_{A\circ_1 B} \sim_2 P_A \circ_2 P_B\}$, for each binary connective $\circ_1$ in κ_1.

In case $\kappa_1 = \kappa_2$, we assume $K_1 = K_2, u_1 = u_2$, and $\circ_1 = \circ_2$. It is easy to check that the total number of occurrences of atoms in $\mathrm{def}_{\kappa_2}(A)$ is bounded by a linear function of $l(A)$.

Lemma 4.10 *Suppose $e\mathscr{F}$ is an extended Frege system over κ, A is an admissible formula over κ, and $\mathrm{def}_\kappa(A) \vdash^{\pi}_{e\mathscr{F}} P_A$. Then for some π' we have $\vdash^{\pi'}_{e\mathscr{F}} A$, where $\lambda(\pi') \le \lambda(\pi) + cl(A)$ and $\rho(\pi') \le (\rho(\pi) + c)l(A)$, and c depends only on $\mathscr{F}$.*

Proof. Let σ be the simultaneous substitution E/P_E for all nonatomic subformulas E of A, so in particular $\sigma P_A = A$. Then every formula in $\sigma(\mathrm{def}_\kappa(A))$ is an instance of $P \sim P$, and each of these instances will have a proof in $\mathscr{F}$ of some fixed number of lines, and a number of atoms bounded by a constant times $l(A)$. These proofs, together with $\sigma(\pi)$, comprise π'. ∎

Lemma 4.11 *If $e\mathscr{F}$ and $e\mathscr{F}'$ are extended Frege systems over κ and κ' respectively, A' is an admissible formula over κ', and $\mathrm{def}_\kappa(A') \vdash^{\pi}_{e\mathscr{F}} P_{A'}$, then for some derivation π', $\mathrm{def}_{\kappa'}(A') \vdash^{\pi'}_{e\mathscr{F}'} P_{A'}$, where $\lambda(\pi') \le c\lambda(\pi)$ and $\rho(\pi') \le d$, and the constants c and d depend only on $\mathscr{F}$ and $\mathscr{F}'$.*

Proof. Suppose π is $B_1, \ldots, B_m$. We may assume, by renaming if necessary, that all atoms of each B_i are admissible, except possible those which occur in the hypotheses or conclusion of π (i.e. except those of the form $P_{C'}$, where C' is a subformula of A'). We shall construct the derivation π' in $e\mathscr{F}'$ by filling out the skeleton derivation $P_{B_1}, \ldots, P_{B_m}$. (Notice that P_{B_m} is $P_{A'}$, since B_m is $P_{A'}$ and in general $P_Q = Q$ for any atom Q.) In fact, we shall show that for some constants c and d depending only on $\mathscr{F}$ and $\mathscr{F}'$, each P_{B_i} can be derived from earlier P_{B_j}'s and $\mathrm{def}_{\kappa'}(A')$ in at most c lines by formulas C with $l(C) \le d$. ∎

To see how to derive P_{B_i} in π' we consider three cases, depending on how B_i was obtained in π. For each of these cases we assume that some of the formulas of $\mathrm{def}_{\kappa'}(B_i)$ are available in π', either because they are among the hypotheses $\mathrm{def}_{\kappa'}(A')$ of π' or because they are introduced at the beginning of π' by the extension rule. The defining formula for P_C, where C is a subformula of B_i, is in $\mathrm{def}_{\kappa'}(A')$ if C is also a subformula of A'. If C is not a subformula of A', then the defining formula for P_C can legally be included in π' by the extension rule.

Case I. B_i is a hypothesis for π, so B_i is in $\mathrm{def}_\kappa(A')$. We may assume B_i has the form $P_{C'} \sim (P_{D'} \circ P_{E'})$, where C', D', E' are subformulas of A', $P \circ Q$ is the fixed formula over κ equivalent to $P \circ' Q$, and C' is $D' \circ' E'$. (The cases of unary and 0-ary

connectives are similar.) Then $P_{C'} \sim' (P_{D'} \circ' P_{E'})$ is in $\mathrm{def}_{\kappa'}(A')$, and so is a hypothesis of π'. Let $H(\circ')$ be the formula $P \sim (Q \circ R)$ over κ. Note that $H(\circ')$ depends only on the connective $\circ'$, and not otherwise on B_i. Then the rule

$$R = \frac{P \sim' (Q \circ' R),\ \mathrm{def}_{\kappa'}(H(\circ'))}{P_{H(\circ')}}$$

is sound, so by Theorem 2.3 we may assume it is a rule of $\mathscr{F}'$. Let σ be an extension of the substitution

$$\frac{P_{C'}, P_{D'}, P_{E'}, P_{B_i}}{P, Q, R, P_{H(\circ')}}$$

such that $\sigma(\mathrm{def}_{\kappa'}(H(\circ'))) = \mathrm{def}_{\kappa'}(B_i)$. Then P_{B_i} follows in one step by R and σ from $\mathrm{def}_{\kappa'}(A')$ and $\mathrm{def}_{\kappa'}(B_i)$.

Case II. B_i is introduced in π by the extension rule. Then B_i has the form $P \sim C$, where P is a new defined atom. The constraints governing the use of the extension rule imply that P does not occur in the hypotheses or conclusion of π, and by our assumption at the beginning of this proof, P is admissible. Therefore, P does not occur in the hypotheses or conclusion of π'. We note that the formula $P \sim' P_C$, together with any subset of the formulas of $\mathrm{def}_{\kappa'}(B_i)$ not introduced earlier could be introduced by the extension rule in π', after any necessary formulas of $\mathrm{def}_{\kappa'}(B_{i-1})$ and before formulas of $\mathrm{def}_{\kappa'}(B_{i+1})$ are introduced. The order of introduction could be $\mathrm{def}_{\kappa'}(C), P \sim' P_C$, followed by one or more formulas whose conjunction is equivalent to $P_{B_i} \sim' (P \sim' P_C)$. This last formula itself will be in $\mathrm{def}_{\kappa'}(B_i)$ if $\equiv$ is in κ, in which case B_i is $P \equiv C$. In this case, it follows from Theorem 2.3 that P_{B_i} can be deduced in a bounded number of bounded steps in π' from $P \sim' P_C$ and $P_{B_i} \sim' (P \sim' P_C)$. If $\equiv$ is not in κ, there are nevertheless a bounded number of formulas in $\mathrm{def}_{\kappa'}(B_i)$ which imply $P_{B_i} \sim' (P \sim' P_c)$, and the number and structure of these formulas depends only on the way $\equiv$ is represented in κ and κ'. Hence again P_{B_i} can be deduced in π' from $\mathrm{def}_{\kappa'}(B_i)$ and $P \sim' P_C$ by a bounded number of bounded formulas.

Case III. B_i follows from earlier formulas in π by a rule $R = (C_1, \ldots, C_k)/D$ in $\mathscr{F}$ by the substitution σ. Then $C_1, \ldots, C_k \models D$, so the rule

$$R' = \frac{\mathrm{def}_{\kappa'}(D),\ \mathrm{def}_{\kappa'}(C_1), \ldots,\ \mathrm{def}_{\kappa'}(C_k), P_{C_1}, \ldots, P_{C_\kappa}}{P_D}$$

is sound, and by Theorem 2.3 we may assume it is a rule of $\mathscr{F}'$. We may assume all formulas $C_1, \ldots, C_k, D$ are admissible. Let σ' be the composition of the substitutions $\sigma(E)/P_E$ for all subformulas E of formulas in the set $\{C_1, \ldots, C_k, D\}$. Then $\sigma'(\mathrm{def}_{\kappa'}(C_j)) \subseteq \mathrm{def}_{\kappa'}(\sigma(C_j)), 1 \leq j \leq k$, and $\sigma'(\mathrm{def}_{\kappa'}(D)) \subseteq \mathrm{def}_{\kappa'}(\sigma(D))$. Of course

each $\sigma(C_j)$ is some $B_l, 1 < i$, and $\sigma(D)$ is B_i. By the induction hypothesis $P_{\sigma(C_j)}$ occurs earlier in π'. Hence P_{B_i} follows by R' and σ' from earlier formulas π' and a bounded number of formulas from $\mathrm{def}_{\kappa'}(B_l)$, for various B_l.

This completes the proof of Lemma 4.11.

Now assume the hypotheses of Theorem 4.5, and let A' be any valid formula over κ'. We may assume A' is admissible, for if not, we may rename the atoms in A' so that it is admissible, find a suitable proof of the result, and then rename all atoms in the proof to obtain a suitable proof of A'. Then $\mathrm{def}_\kappa(A') \models P_{A'}$, so by hypothesis, the bounds on $l(\mathrm{def}_\kappa(A'))$, and Lemma 4.8, there is a derivation π in $e\mathscr{F}$ of $P_{A'}$ from $\mathrm{def}_{\kappa'}(A')$ such that $\lambda(\pi) \leq c_1 L(c_1 l(A'))$. By Lemma 4.11, there is a derivation π' in $e\mathscr{F}'$ of $P_{A'}$ from $\mathrm{def}_\kappa(A')$ such that $\lambda(\pi') \leq c_2 L(c_1 l(A'))$ and $\rho(\pi') \leq d$. Theorem 4.5 now follows by Lemma 4.10.

To prove Theorem 4.6, we may assume as above that A is admissible. By induction on the length of B, it is easy to see that for every admissible formula B over κ there is a derivation π_B in $\mathscr{F}$ of $P_B \sim B$ from $\mathrm{def}_\kappa(B)$ such that $\lambda(\pi_B) \leq c_1 l(B)$ and $\rho(\pi_B) \leq c_2 l(B)$, where κ is the connective set of $\mathscr{F}$. By putting together π_A with π in the theorem, we obtain a derivation π_1 of P_A from $\mathrm{def}_\kappa(A)$ such that $\lambda(\pi_1) \leq c_3(\lambda(\pi) + l(A))$ and $\rho(\pi_1) \leq c_4 l(\pi)$. We now apply Lemma 4.11 with $\kappa' = \kappa$, $e\mathscr{F}' = e\mathscr{F}$, and $\pi = \pi_1$ to modify π_1 so its formulas have bounded length, and finally apply Lemma 4.10 to the resulting derivation. ∎

5 The Substitution Rule

Frege's original propositional proof system [Fre67] tacitly assumed the following:

Substitution Rule 5.1 From A conclude σA, for any substitution σ in the notation of the system.

Definition 5.2 A *Frege system with substitution*, $s\mathscr{F}$, is obtained from a Frege system $\mathscr{F}$ by addition of the substitution rule. Hypotheses are not allowed in derivations in $s\mathscr{F}$.

The reason hypotheses are not allowed in $s\mathscr{F}$-derivations is that in general not $A \models \sigma A$. Thus the substitution rule is unsound in this sense. On the other hand, if $\models A$ then $\models \sigma A$, so if $\vdash_{s\mathscr{F}} A$ then $\models A$. In other words, $s\mathscr{F}$ is a sound system for proving tautologies, but not for deriving formulas from hypotheses.

The theorem below shows that Frege systems with substitution can p-simulate extended Frege systems. The converse may be false, however. (We conjecture Frege systems with substitution are not p-verifiable in the sense of [Coo76b], whereas extended Frege systems are p-verifiable.)

Theorem 5.3 *Given an extended Frege system $e\mathscr{F}$ there is a function f in $\mathscr{L}$ and constant c such that for all proofs π and formulas A, if $\vdash^{\pi}_{e\mathscr{F}} A$, then $\vdash^{f(\pi)}_{s\mathscr{F}} A$,* and $\lambda(f(\pi)) \leq c\lambda(\pi)\rho(\pi)$ and $\rho(f(\pi)) \leq c\lambda(\pi)\rho(\pi)$.

Proof. Suppose $P_1 \sim C_1, \ldots, P_k \sim C_k$ are the defining formulas introduced by extension in π. As discussed before Theorem 3.3, $\mathscr{F}$ can be turned into a natural deduction system $\mathscr{N}$ by including the rules

$$R_1 = \frac{P \to Q}{\to \neg P \vee Q} \quad \text{and} \quad R_2 = \frac{\to \neg P \vee Q}{P \to Q}.$$

Let us assume in addition that $\mathscr{N}$ has the rules

$$R_3 = \frac{P \to Q}{P, R \to Q} \quad \text{and} \quad R_4 = \frac{P \to Q}{R, P \to Q},$$

and the axiom $P \to P$. Then for each $i, 1 \leq i \leq k$, the line $E_1, \ldots, E_k \to E_i$ can be derived from the axiom and $k-1$ uses of R_3 and R_4, for any formulas $E_1, \ldots, E_k$. The derivations of these k lines, together with π, describe a derivation π_1 in $\mathscr{N}$ of $E_1, \ldots, E_k \to A$, where now E_i is the defining formula $P_i \sim C_i$, and $\lambda(\pi_1) \leq \lambda(\pi) + k^2$ and $\rho(\pi_1) \leq (k+1)\rho(\pi)$. Now by adding k applications of rule R_1, we obtain a derivation in $\mathscr{N}$ of B, where B is $\neg E_1 \vee (\neg E_2 \vee \cdots \vee (\neg E_k \vee A)\cdots)$. Hence noting $k < \lambda(\pi)$, we have by the proof of Corollary 3.4 a derivation π_2 in $\mathscr{F}$ of B, where $\lambda(\pi_2) \leq c_1(\lambda(\pi))^2$ and $\rho(\pi_2) \leq c_1\lambda(\pi)\rho(\pi)$. Now assume the defining formulas $P_1 \sim C_1, \ldots, P_k \sim C_k$ are numbered in reverse of the order in which they appear in π. Then $P_1 \sim C_1$ appears last, so P_1 has no occurrence in any C_i or in A. By applying the substitution rule to B with the substitution C_1/P_1, and applying the Frege rule $(\neg(P \sim P) \vee Q)/Q$, we can derive $(\neg E_2 \vee \cdots \vee (\neg E_k \vee A)\cdots)$ from B. By $k-1$ further applications of the substitution rule and this Frege rule, each of the E_i's can be pruned, and we obtain a proof of A in $s\mathscr{F}$ which satisfies the conditions of the theorem. ∎

By combining the above theorem with Theorem 4.5, we obtain the following.

Corollary 5.4 *If there exists a polynomially bounded extended Frege system, then all Frege systems with substitution over all connectives sets are polynomially bounded.* ∎

A result similar to Theorem 4.5 can be proved for Frege systems with substitution, using the methods in that proof and in the above argument. In particular, one Frege system with substitution is polynomially bounded if and only if all such systems over all connective sets are polynomially bounded. Reckhow [Rec76] proves this result by different methods.

References

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

[Chu56] A. Church. 1956. *Introduction to Mathematical Logic*, vol. I. Princeton University Press, Princeton.

[Coo71] S. A. Cook. May 3–5. 1971. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*. Shaker Heights, Ohio. ACM, 151–158. DOI: https://doi.org/10.1145/800157.805047.

[Coo76] S. A. Cook. 1976. A short proof of the pigeon-hole principle using extended resolution. *SIGACT News* 8, 4, 28–32. DOI: https://doi.org/10.1145/1008335.1008338.

[CR74] S. A. Cook and R. A. Reckhow. April 30–May 2. 1974. On the lengths of proofs in the propositional calculus (preliminary version). In *Proceedings of the 6th Annual ACM Symposium on Theory of Computing*. Seattle, Washington. ACM, 135–148. DOI: https://doi.org/10.1145/800119.803893.

[CR+76] S. A. Cook, C. W. Rackoff, et al. 1976. Lecture notes for CSC2429F, "Topics in the theory of computation", a course presented by the Department of Computer Science, University of Toronto during the fall.

[Fre67] G. Frege. 1867. *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. Halle, 1879. English translation in *From Frege to Godel, a Source Book in Mathematical Logic* (J. van Heijenoort, Editor), Harvard University Press, Cambridge, 1–82.

[Kar72] R. M. Karp. 1972. Reducibility among combinatorial problems. In *Proceedings of a Symposium on the Complexity of Computer Computations*, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York. Springer, 85–103. DOI: https://doi.org/10.1007/978-1-4684-2001-2_9.

[vNeu27] J. von Neumann. 1927. Zur Hilbertschen Beweistheorie. *Mathematische Zeitschrift* 26, 1–46.

[Pra65] D. Prawitz. 1965. *Natural Deduction, A Proof-Theoretical Study*. Vol. 3: Stockholm Studies in Philosophy. Almqvist & Wiskell.

[Rec76] R. A. Reckhow. 1976. *On the Lengths of Proofs in the Propositional Calculus*. Ph.D. thesis. Department of Computer Science, University of Toronto.

[Sta77] R. Statman. 1977. Complexity of derivations from quantifier-free horn formulae, mechanical introduction of explicit definitions, and refinement of completeness theorems. *Proceedings of the Logic Colloquium* 76 (Gandy and Hyland, Editors), *Studies in Logic and Foundations of Mathematics*, Vol. 87, North-Holland, Amsterdam.

[Tse70] G. S. Tseitin. 1970. On the complexity of derivation in propositional calculus. *Semin. Math., V. A. Steklov Math. Inst., Leningrad* 8, 115–125; translation from 1968. *Zap. Nauchn. Semin. Leningr. Otd. Mat. Inst. Steklova* 8, 234–259.

12 Feasibly Constructive Proofs and the Propositional Calculus (Preliminary Version)

Stephen A. Cook

1 Introduction

The motivation for this work comes from two general sources. The first source is the basic open question in complexity theory of whether *P* equals *NP* (see [Coo71b] and [Kar72]). Our approach is to try to show they are not equal, by trying to show that the set of tautologies is not in *NP* (of course its complement *is* in *NP*). This is equivalent to showing that no proof system (in the general sense defined in [CR74]) for the tautologies is "super" in the sense that there is a short proof for every tautology. Extended resolution is an example of a powerful proof system for tautologies that can simulate most standard proof systems (see [CR74]). The Main Theorem (5.5) in this paper describes the power of extended resolution in a way that may provide a handle for showing it is not super.

The second motivation comes from constructive mathematics. A constructive proof of, say, a statement $\forall x A$ must provide an effective means of finding a proof of A for each value of x, but nothing is said about how long this proof is as a function of x. If the function is exponential or super exponential, then for short values of x the length of the proof of the instance of A may exceed the number of electrons in the universe. Thus one can question the sense in which our original "constructive" proof provides a method of verifying $\forall x A$ for such values of x. Parikh [Par71]

Originally published in STOC '75: Proceedings of the seventh annual ACM symposium on Theory of computing May 1975 Pages 83–97.

makes similar points, and goes on to suggest an "anthropomorphic" formal system for number theory in which induction can only be applied to formulas with bounded quantifiers. But even a quantifier bounded by n may require time exponential in the length of (the decimal notation for) n to check all possible values of the quantified variable (unless $P = NP$), so Parikh's system is apparently still not feasibly constructive.

In section 2, I introduce the system PV for number theory, and it is this system which I suggest properly formalizes the notion of a feasibly constructive proof. The formulas in PV are equations $t = u$, (for example, $x \cdot (y + z) = x \cdot y + x \cdot z$) where t and u are terms built from variables, constants, and function symbols ranging over L, the class of functions computable in time bounded by a polynomial in the length of their arguments. The system PV is the analog for L of the quantifier-free theory of primitive recursive arithmetic developed by Skolem [Sko23] and formalized by others (see [Goo57]). A result necessary for the construction of the system is Cobham's theorem [Cob65] which characterizes L as the least class of functions containing certain initial functions, and closed under substitution and limited recursion on notation (see section 2). Thus all the functions in L (except the initial functions) can be introduced by a sequence of defining equations. The axioms of PV are these defining equations, and the rules of PV are the usual rules for equality, together with "induction on notation".

All proofs in PV are feasibly constructive in the following sense. Suppose an identity, say $f(x) = g(x)$, has a proof Π in PV. Then there is a polynomial $p_\Pi(n)$ such that Π provides a uniform method of verifying within $P_\Pi(|x_0|)$ steps that a given natural number x_0 satisfies $f(x_0) = g(x_0)$. If such a uniform method exists, I will say the equation is *polynomially verifiable* (or p-verifiable).

The reader's first reaction might be that if both f and g are in L, then there is always a polynomial $p(n)$ so that the time required to evaluate them at x_0 is bounded by $P(|x_0|)$, and if $f(x) = g(x)$ is a true identity, then it should be p-verifiable. The point is that the verification method must be uniform, in the sense that one can see (by the proof Π) that the verification will always succeed. Not all true identities are provable, so not all are p-verifiable.

There is a similar situation in constructive (or intuitionistic) number theory. The Kleene-Nelson theorem ([Kle52], p. 504) states that if a formula $\forall x A$ has a constructive proof, then it is recursively realizable in the sense that there is a recursive function f which takes x_0 into a proof of $A\frac{x_0}{x}$ (more properly, $f(x_0)$ is a number which "realizes" $A\frac{x_0}{x}$). The converse is false. One can find a formula $\forall x A$ which is recursively realizable, but not constructively provable, since one cannot prove that the realizing recursive function works. Similarly, any true equation $f(x) = g(x)$ in PV

is recursively realizable (in fact, L-realizable), but not all are p-verifiable (i.e. have feasibly constructive proofs).

I argue in section 2 that provable equations in PV are p-verifiable. I also conjecture the converse is true, which leads to

Verifiability Thesis 1.1 An equation $t = u$ of PV is provable in PV if and only if it is p-verifiable.

This statement is similar to Church's thesis, in that one can never prove that PV is powerful enough, since the notion of p-verifiable is informally defined. We present evidence for the power of PV in this paper by giving examples of things that are provable in PV, and by presenting the system PV1 in section 3 which appears to be more powerful than PV, but isn't.

Another argument for the power of PV that can be made is this. There is evidence that intuitionistic number theory, as formalized by Kleene [Kle52], is equivalent to a quantifier-free theory in which functions are introduced by ordinal recursion up to ϵ_0. From this point of view, PV is the same kind of quantifier-free theory, except the kind of recursion allowed is restricted so that only functions in L can be defined.

In section 2, the system PV is described in detail, and some simple examples of proofs in the system are given. The Valuation Theorem (2.18) states that all true equations in PV without variables are provable in PV.

In section 3, the system PV1 is presented. This system allows formulas to be truth functional combinations of equations, instead of just equations, and is much more convenient than PV for formalizing proofs. Nevertheless, theorem 3.10 states that any equation provable in PV1 is provable in PV.

The second Gödel Incompleteness theorem for PV, stating that the consistency of PV cannot be proved in PV, is proved in outline in section 4. I am aware of only one other treatment in the literature of this theorem for a free-variable system, and that is in [Ros61]. (However, there seems to be a mistake in [Ros61], since theorem 16, p. 134 fails when $f(s(x))$ is neither identically zero nor identically non-zero.)

In section 5, the proof system *extended resolution* is described, and the notion of a p-verifiable proof system for the propositional calculus is defined. The Main Theorem (5.5) states that a proof system f for the propositional calculus is p-verifiable iff extended resolution can simulate f efficiently, and the proof that the simulation works can be formalized in PV. The "if" part is proved in outline.

Section 6 describes how to develop propositional formulas which express the truth of equations $t = u$ of PV for bounded values of the variables in t and u. The ER Simulation Theorem (6.8) states that if $t = u$ is provable in PV, then

there is a polynomial (in the length of the bound on the variables) bound on the length of the minimal extended resolution proofs of the associated propositional formulas. The "only if" part of the Main Theorem is then proved in outline from this.

In section 7, it is shown how the Gödel Incompleteness theorem implies that the system PV, as a proof system for the propositional calculus, is not itself p-verifiable.

Finally, section 8 offers some conclusions and directions for future research.

2 The System PV

I will use dyadic notation (see Smullyan [Smu61]) to denote natural numbers.[1] The dyadic notation for the natural number n is the unique string $d_k d_{k-1} \ldots d_0$ over the alphabet $\{1, 2\}$ such that $\sum_{i=0}^{k} d_i 2^i = n$. In particular, the dyadic notation for 0 is the empty string. The dyadic successor functions $s_1(x)$ and $s_2(x)$ are defined by $s_i(x) = 2x + i, i = 1, 2$, and correspond to concatenating the digits 1 and 2, respectively, on the right end of the dyadic notation for x. I shall thus abbreviate $s_i(x)$ by xi.

A function f comes from functions $g_1, \ldots, g_m$ by the operation of *substitution* iff some equation of the form

$$f(x_1, \ldots, x_n) = t \tag{2.1}$$

holds for all $x_1, \ldots, x_n$, where t is a syntactically correct term built up from the variables $x_1, \ldots, x_n$, numerals for the natural numbers, and the function symbols $g_1, \ldots, g_m$.

A function f comes from functions g, h_1, h_2, k_1, k_2 by the operation of *limited recursion* on *dyadic notation* iff

$$f(0, \bar{y}) = g(\bar{y}) \tag{2.2}$$

$$f(xi, \bar{y}) = h_i(x, \bar{y}, f(x, \bar{y})), \quad i = 1, 2 \tag{2.3}$$

$$f(x, \bar{y}) \leq k_i(x, \bar{y}), \quad i = 1, 2 \tag{2.4}$$

for all natural number values of the variables, where $\bar{y} = (y_1, \ldots, y_k)$. We allow the case $k = 0$, in which g is a constant.

Cobham's class L can be defined to be the set of functions f on the natural numbers such that for some Turing machine Z and some polynomial p, for all natural numbers $x_1, \ldots, x_n$, Z computes $f(x_1 \ldots, x_n)$ within $p(|x_1| + \ldots + |x_n|)$ steps , where $|x|$ is the length of the dyadic notation for x.

1. The trouble with the more conventional binary notation is the necessity of proving the consistency of the analogs of equations 2.2 and 2.3 when $x = i = 0$.

Definition 2.5 The dyadic notation for $\otimes(x,y)$ is the dyadic notation for x concatenated with itself $|y|$ times.

Theorem 2.6 **Cobham**
L is the least class of functions which includes the initial functions s_1, s_2, and $\otimes$, and which is closed under the operations of substitution and limited recursion on dyadic notation.

Cobham stated this result in [Cob65], in a slightly different form. I am not aware of any published proof of the theorem, although Lascar gave a proof in some unpublished seminar notes [Las67].

The formal system PV will have function symbols with defining equations of the forms 2.1, 2.2, and 2.3. I want only functions in L to be definable in PV, which means the inequalities 2.4 must be satisfied for some functions k_1, k_2 in L. It is not hard to see that the question, given g, h_1, h_2, k_1, k_2, of whether the function f defined by 2.2 and 2.3 satisfies 2.4 is recursively undecidable. I want, however, for the proof predicate in PV to be not only decidable, but definable in PV. Therefore, I shall require that before a function f can be introduced by 2.2 and 2.3, a proof must be available in PV that f does not grow too fast. It is awkward to require that 2.4 be proved directly in PV, because it obviously cannot be proved without using f, whose status in PV is uncertain until after the proof is carried out. Thus the proof will instead verify the inequality

$$|h_i(x,\bar{y},z)| \leq |z^* k_i(x,\bar{y})|, \quad i = 1,2$$

for some previously defined functions k_1 and k_2 (not those in 2.4), where * indicates concatenation. It is easy to see that this inequality guarantees that f is in L if k_1 and k_2 are in L, since then $|f(x,\bar{y})| \leq |g(\bar{y})| + |k(0,\bar{y})| + |k(d_1,\bar{y})| + \ldots + |k(d_1 \ldots d_k,\bar{y})|$ where $d_1 \ldots d_{k+1}$ is the dyadic notation for x and $k(x,\bar{y}) = k_1(x,\bar{y}) + k_2(x,\bar{y})$.

In order to specify formally what constitutes a proof of this inequality, we must introduce enough initial functions in PV to define the relation $|x| \leq |y|$. Thus we introduce a function TR(x) (TR for "trim") which deletes the right-most digit of x. From this, a function LESS(x,y) can be defined whose value is x with the right-most $|y|$ digits deleted. Thus $|x| \leq |y|$ iff LESS(x,y) $= 0$. In addition, we need * (concatenation) as an initial function, and also $\otimes$ (see 2.5). The purpose of $\otimes$ is to allow formation of functions in PV by composition which grow sufficiently fast to dominate any function in L.

Function symbols in PV will be defined later to be certain strings of symbols which encode the complete derivation from initial functions of the function they stand for. In particular, the defining equation(s) and number of arguments (arity) for a function symbol can be determined by inspection from the symbol.

The set of *terms* of PV is defined inductively as follows. (i) 0 is a term, any variable x is a term, and any function symbol f of arity 0 is a term. (ii) If $t_1, \ldots, t_k$ are terms, and f is a function symbol of arity $k \geq 1$, then $f(t_1, \ldots, t_k)$ is a term. An *equation* is a string of the form $t = u$, where t and u are terms. A *derivation* in PV of an equation E from equations $E_1, \ldots, E_n$ is a string of equations of the form $D_1, \ldots, D_\ell$, such that D_ℓ is E, and each $D_i, 1 \leq i \leq \ell$, is either some is either some E_j, a defining equation for a function symbol, or follows from earlier equations in the string by a rule of PV (see below). If such a derivation exists, we shall write $E_1, \ldots, E_n \vdash_{\mathrm{PV}} E$, or simply $\vdash_{\mathrm{PV}} E$, if there are no hypotheses (the symbol PV here will sometimes be deleted). A derivation of E from no hypotheses is a *proof* of E.

Rules of PV

(Here t, u, v are any terms, x is a variable, and $\bar{y}$ is a k-tuple of variables, $k \geq 0$.)

R1. $t = u \vdash u = t$

R2. $t = u, u = v \vdash t = v$

R3. $t_1 = u_1, \ldots, t_k = u_k \vdash f(t_1, \ldots, t_k) = f(u_1, \ldots, u_k)$,
for any k-place function symbol $f, k \geq 1$.

R4. $t = u \vdash t\frac{v}{x} = u\frac{v}{x}$,
where $\frac{v}{x}$ indicates substitution of the term v for the variable x.

R5. (Induction on notation) $E_1, \ldots, E_6 \vdash f_1(x, \bar{y}) = f_2(x, \bar{y})$,
where $E_1, \ldots, E_6$ are the equations 2.2 and 2.3 with f replaced by f_1 and by f_2.

The definition of proof is not yet complete, because the notion of function symbol (and hence of term and equation) and associated defining equations has not yet been specified. These notions must actually be defined inductively simultaneously with the definition of proof, because of our requirement that the boundedness of functions be proved in PV. The arity of a function symbol is the number of arguments, and the order of the symbol is roughly the depth of nesting of recursion on notation used to define it. We define the *order* of a proof to be the greatest of the orders of the function symbols occurring in it. Now we can complete the definitions of all these notions simultaneously and recursively as follows:

The *initial* function symbols all have order 0. These are the symbol 0 (of arity 0), $\mathrm{s}_1, \mathrm{s}_2$, TR (each of arity 1) and $^*, \otimes$, LESS (each of arity 2). There are no defining equations for 0, s_1 and s_2, and the defining equations for the others are (here x_1 means $\mathrm{s}_1(x)$, x_2 means $\mathrm{s}_2(x)$):

TR: $\mathrm{TR}(0) = 0$

$\mathrm{TR}(xi) = x, \quad i = 1,2$

*: $^*(x,0) = x$

$^*(x,yi) = \mathrm{s}_i(^*(x,y)), \quad i = 1,2$

$\otimes$: $\otimes(x,0) = 0$

$\otimes(x,yi) =^* (x, \otimes(x,y)), \quad i = 1,2$

LESS: $\mathrm{LESS}(x,0) = x$

$\mathrm{LESS}(x,yi) = \mathrm{TR}(\mathrm{LESS}(x,y)), \quad i = 1,2$

Note: We use infix notation for * and $\otimes$ after this.

It t is a term, and k is the maximum of the orders of the function symbols occurring in t, and all variables in t are among the variables $x_1, \ldots, x_n, n \geq 0$, then $\lambda x_1 \ldots x_n\ t\rho$ is a function symbol of arity n and order k. The defining equation is $f(x_1, \ldots, x_n) = t$, if $n \geq 1$, and $f = t$, if $n = 0$, where f is $\lambda x_1 \ldots x_n t\rho$.

If g, h_1, h_2, k_1, k_2 are function symbols of arity $n-1, n+1, n+1, n$, and n, respectively, $(n \geq 1)$ and if k is the maximum of the orders of the five function symbols, and if Π_i, $i = 1,2$ are proofs of order k or less of $\mathrm{LESS}(h_i(x,\bar{y},z), z^* k_i(x,\bar{y})) = 0$, $i = 1,2$, then $\langle [g,h_1,h_2,k_1,k_2][\Pi_1][\Pi_2]\rangle$ is a function symbol of arity n and order $k+1$. If f denotes this function symbol, then the three defining equations for f are

$$f(0,\bar{y}) = g(\bar{y}) \quad (or f(0) = g, \text{ if } n = 1)$$
$$f(xi,\bar{y}) = h_i(x,\bar{y},f(x,\bar{y})), \quad i = 1,2$$

All function symbols must be formed in these ways. This completes the formal specification of the system PV.

As examples of proofs on PV, let us verify some simple properties of LESS or TR.

$$\vdash_{\mathrm{PV}} \mathrm{TR}(\mathrm{LESS}(xi,y)) = \mathrm{LESS}(x,y), \quad i = 1,2 \tag{2.7}$$

The strategy is to use R5 (induction on notation). To do this we introduce a new function symbol f (formally, f is $\lambda xy\mathrm{TR}(\mathrm{LESS}(yi,x))\rho$) with defining equation $f(x,y) = \mathrm{TR}(\mathrm{LESS}(yi,x))$. Also a function symbol LESS$'$ is introduced with defining equation $\mathrm{LESS}'(x,y) = \mathrm{LESS}(y,x)$. Now the hypotheses of the induction rule can be verified, when f_1 is f and f_2 is LESS$'$, and $g(y) = y$, and $h_j(x,y,z) = \mathrm{TR}(z); j = 1,2$. Hence $f(x,y) = \mathrm{LESS}'(x,y)$, from which 2.7 follows by R1, R2, and R4, and the defining equations for f and LESS$'$.

$$\vdash_{\mathrm{PV}} \mathrm{LESS}(x,x) = 0 \tag{2.8}$$

This is shown by induction on x, using 2.7 with y replaced by x.

$$\vdash_{\mathrm{PV}} \ \mathrm{LESS}(x, yi^*z) = \mathrm{TR}(\mathrm{LESS}(x, y^*z)), \quad i = 1, 2 \tag{2.9}$$

This is proved by induction on z. Here $h_j(x, y, z, u) = \mathrm{TR}(u)$.

$$\vdash_{\mathrm{PV}} \ \mathrm{LESS}(x, y^*z) = \mathrm{LESS}(x, z^*y) \tag{2.10}$$

Again this is proved by induction on y, using 2.9, and the same function h_j above.

$$\vdash_{\mathrm{PV}} \ \mathrm{LESS}(x, x^*y) = 0 \tag{2.11}$$

The proof is induction on y, using 2.8.

The intended semantics of PV should be clear. Every function symbol f stands for a uniquely defined function in L, which we can denote by $\phi(f)$. (The reader can give a precise definition of $\phi(f)$ by induction on the length of the function symbol f.) An equation $t = u$ in PV is true iff its universal closure is true in the domain of natural numbers, when all function symbols receive their standard interpretations.

We say a function F on the natural numbers is *definable* in PV iff $\phi(f) = F$ for some function symbol f of PV. By Cobham's theorem, every function definable in PV is clearly in L, but the converse is far from obvious, because of our requirement that the bounding inequalities be provable in PV. Nevertheless, the converse is true.

Theorem 2.12 Every function in L is definable in PV.

To prove this requires a reproving of half of Cobham's theorem, showing that the functions introduced by limited recursion on notation can have their bounding inequalities proved in PV. We will not give the argument here.

Below we introduce two functions in PV which we will use in the next section. The defining equations given do not strictly fit the format for recursion on notation, since the function symbols g, h_1, h_2, k_1, k_2 would have to be introduced explicitly. However, the reader should have no trouble doing this.
Note: $\mathrm{s}_1(0)$ is abbreviated by 1, and $\mathrm{s}_2(0)$ is abbreviated by 2.

$$\overline{\mathrm{sg}}(0) = 1 \tag{2.13}$$
$$\overline{\mathrm{sg}}(xi) = 0, \quad i = 1, 2$$
$$\mathrm{sg}(0) = 0 \tag{2.14}$$
$$\mathrm{sg}(xi) = 1, \quad i = 1, 2$$
$$\mathrm{CON}(0, y) = 0 \tag{2.15}$$
$$\mathrm{CON}(xi, y) = \mathrm{sg}(y)$$

The bounding inequalities for the above three functions are easily proved in PV from the defining equations for LESS and TR.

We now wish to argue in support of one part of the Verifiability Thesis (1.1), namely that only p-verifiable equations are provable in PV. Our argument includes an outline of a highly constructive consistency proof for PV, and it could be formalized in, say, primitive recursive number theory, to show there is no proof in PV of $0 = 1$. An indication of how a similar argument showing the consistency of elementary arithmetic (in the sense of Kalmar) could be carried out in primitive recursive arithmetic was given in Rose [CR67].

Proposition 2.16 If $\vdash_{\mathrm{PV}} t = u$, then the equation $t = u$ is p-verifiable.

Corollary 2.17 Not $\vdash_{\mathrm{PV}} 0 = 1$.

Our argument for establishing 2.16 proceeds by induction on the length of the proof of $t = u$ (here *length* counts the length of the function symbols in the proof). Thus suppose $\ell \geq 1$, and the proposition holds for all proofs of length $< \ell$. Let Π be a proof of $t = u$ of length ℓ. If $t = u$ is a defining equation for a function symbol f, then the equation holds by definition of f. However, the time required to verify the equation for a particular value of the arguments is equal to the time to compute f at that value, so we must be sure that this computation time is bounded by a polynomial in the length of the arguments. Here we apply the induction hypothesis, both to be sure that f does not grow too fast (we know this partly because if f is defined by recursion, then there are proofs of length less than ℓ, establishing a bound on the growth rate) and that all functions used in defining f can be computed in polynomial time.

Now suppose $t = u$ follows from earlier equations in Π by one of the rules $R1, \ldots, R5$.

We will consider R4 as an interesting example. Thus (changing the roles of t and u to be consistent with the notation of R4) we assume by the induction hypothesis that $t = u$ is p-verifiable, and also that the equations defining the functions in the term v give a polynomial time method of evaluating v. Thus, to verify $t\frac{v}{x} = u\frac{v}{x}$ for particular values for the arguments, we first evaluate v at the argument values, obtaining v_0, and then (using the induction hypothesis) verify $t = u$ at these argument values except we let x have the value v_0. Note that, by the induction hypothesis, we are confident that the equation will hold at the values. Further, since a composition of polynomials is a polynomial, the whole process is bounded in time by a polynomial in the length of the arguments.

We leave the other rules to the reader.

Notice that nothing is said about how the verification time grows with the length of the proof Π. In fact, it is easy to see that the naive bound on this time

is at least exponential in the length of Π for fixed argument values, and we will prove in section 7 that PV itself is, in a sense, not p-verifiable.

The final result in this section is the following:

Theorem 2.18 **Valuation Theorem**
If $t = u$ is a true equation of PV without variables, then $\vdash_{\mathrm{PV}} t = u$.

Definition 2.19 The *numeral* $\overline{n}$ for the natural number n is the unique term in PV of the form $\mathrm{s}_{i_1}(\mathrm{s}_{i_2}(\ldots \mathrm{s}_{i_k}(0)\ldots)$ whose value is n. In particular, the *numeral* for 0 is '0'.

Lemma 2.20 Every true equation in PV of the form $f(\overline{n}_1, \ldots, \overline{n}_k) = \overline{m}$ is provable in PV.

First let us note that the valuation theorem follows from the lemma. One shows by induction on the length of t (using the lemma) that if $t = \overline{n}$ is a true equation, then it is provable in PV (rules R1, R2, R3 are all that is needed for this). But $t = \overline{n}, u = \overline{n} \vdash_{\mathrm{PV}} t = u$.

The lemma is proved by induction on the length of the function symbol f, where we take the lengths of s_1 and s_2 to be 0, and the lengths of TR, *, $\otimes$, and LESS to be 1, 2, 3, and 4, respectively. If f is s_1 or s_2, then our task is to show $\vdash_{\mathrm{PV}} \overline{m} = \overline{m}$. But the identity function has defining equation $I(x) = x$, from which we may conclude $x = x$ by R1 and R2, and $\overline{m} = \overline{m}$ by R4.

Now suppose f is $\lambda x_1 \ldots x_n t\rho$ for some term t. Then the defining equation for f is $f(x_1, \ldots, x_n) = t$. If $f(\overline{n}_1, \ldots, \overline{n}_k) = \overline{m}$ is true, then $t\frac{\overline{n}_1,\ldots,\overline{n}_k}{x_1,\ldots,x_k} = \overline{m}$ is true. Since the induction hypothesis applies to each function symbol in t, the argument made two paragraphs above can be applied to show this last equation is provable in PV. Hence, by R4 and R2, $\vdash_{\mathrm{PV}} f(\overline{n}_1, \ldots, \overline{n}_k) = \overline{m}$.

Finally, suppose f is introduced by recursion on notation, so that it has defining equations 2.2 and 2.3. (I intend to include the initial functions TR, *, $\otimes$, and LESS in this case too.) Then one can see by induction on p that if $f(\overline{p}, \overline{n}, \ldots, \overline{n}_k) = \overline{m}$ is true, it is provable in PV. (Notice that the main induction hypothesis holds for the function symbols g, h_1, h_2.)

3 The System PV1

The goal now is to construct a system PV1 in which it is easier to formalize proofs than in PV, and then show that every equation provable in PV1 is provable in PV, and conversely.

As a first step, we notice that it is often easier to define a function by simultaneous recursion on several variables at once, rather than on just one variable, as in

2.2 and 2.3. For example, addition is easily defined this way as follows:

$$x + 0 = 0 + x = x$$
$$xi + yj = \begin{cases} (x+y)2 & \text{if } i = j = 1 \\ (\mathrm{s}(x+y))1 & \text{if } i \neq j \\ (\mathrm{s}(x+y))2 & \text{if } i = j = 2 \end{cases}$$

where $\mathrm{s}(x) = x + 1$.

More generally, $f(x, y, \overline{z})$ is defined from $g_{00}, g_{01}, g_{10}, \{h_{ij}, k_{ij} | i, j \in \{1, 2\}\}$ by *limited 2-recursion on dyadic notation* iff

$$f(0, 0, \overline{z}) = g_{00}(\overline{z}) \tag{3.1}$$
$$f(0, yj, \overline{z}) = g_{01}(y, \overline{z}) \tag{3.2}$$
$$f(xi, 0, \overline{z}) = g_{10}(x, \overline{z}) \tag{3.3}$$
$$f(xi, yj, \overline{z}) = h_{ij}(x, y, \overline{z}, f(x, y, \overline{z})), i, j \in \{1, 2\} \tag{3.4}$$
$$\mathrm{LESS}(h_{ij}(x, y, \overline{z}, u), u^* k_{ij}(x, y, \overline{z})) = 0, i, j \in \{1, 2\} \tag{3.5}$$

The reason for using three initial defining equations (3.1, 3.2, 3.3) instead of just two, defining $f(x, 0, \overline{z})$ and $f(0, y, \overline{z})$, is to avoid the necessity of proving the consistency of the equations when $x = y = 0$.

Theorem 3.6 Suppose there are function symbols $g_{00}, g_{01}, g_{10}, \{h_{ij}, k_{ij} | i, j \in \{1, 2\}\}$ in PV such that the four equations 3.5 are each provable in PV. Then there is a function symbol f in PV such that each of the equations 3.1, . . ., 3.4 is provable in PV.

The proof will not be given here.

It is also useful to have a rule allowing induction on notation on several variables at once.

Theorem 3.7 Suppose the equations

$$f(x_1, \ldots, x_n, \overline{y})\tfrac{0}{x_i} = g_i(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n, \overline{y}), 1 \leq i \leq n \tag{3.8}$$
$$f(x_1 i_1, \ldots, x_n i_n, \overline{y}) = h_{i_1}, \ldots, i_n(x_1, \ldots, x_n, \overline{y}, f(x_1, \ldots, x_n, \overline{y})), \quad (i_1, \ldots, i_n) \in \{1, 2\}^n \tag{3.9}$$

($2^n + n$ equations altogether) are each provable in PV when f is replaced by f_1 and again when f is replaced by f_2. Then $f_1(x_1, \ldots, x_n, \overline{y}) = f_2(x_1, \ldots, x_n, \overline{y})$ is provable in PV.

The proof will not be given here.

The system PV1 can now be defined. Variables, function symbols, terms, and equations are the same as in PV. Formulas in PV1 are either equations, or truth-functional combinations of equations, using the truth-functional connectives $\&, \vee, \neg, \supset, \equiv$. The axioms and axiom schemes of PV1 are the following:

El. $x = x$

E2. $x = y \supset y = x$

E3. $(x = y \,\&\, y = z) \supset x = z$

$(\text{E4})_f$. $(x_1 = y_1 \,\&\, \ldots \,\&\, x_k = y_k) \supset f(x_1, \ldots, x_k) = f(y_1, \ldots, y_k)$ for each $k \geq 1$ and each k-place function symbol f in PV

E5. $(x = y) \equiv (xi = yi), \quad i = 1, 2$

E6. $\neg x1 = x2$

E7. $\neg 0 = xi, \quad i = 1, 2$

DEF. The defining equations for any function symbol in PV are axioms in PV1. Further, the equations 3.1, . . ., 3.4 are axioms in PV1, provided the equations 3.5 are provable in PV1 without using these instances of 3.1–3.4, and provided that the function symbol f is the one given by theorem 3.6. Finally, the defining equations of the initial functions TR, *, $\otimes$, and LESS, are axioms of PV1.

TAUTOLOGY. Any truth-functionally valid formula of PV1 is an axiom of PV1.

The rules of PV1 are the following:

SUBST. $A \vdash A\frac{t}{x}$, where A is any formula of PV1, t is any term, and x is any variable.

IMP. $A_1, \ldots, A_n, \vdash B$, where the formula B is a truth-functional consequence of formulas $A_1, \ldots, A_n$.

n-INDUCTION, $n \geq 1$:

$$\{A\tfrac{0}{x_i} | 1 \leq i \leq n\}, \{A \supset A\tfrac{x_1 i_1, \ldots, x_n i_n}{x_1, \ldots, x_n} \mid (i_1, \ldots, i_n) \in \{1, 2\}^n\} \vdash A$$

For example, 1-induction is the rule

$$A\tfrac{0}{x}, A \supset A\tfrac{x1}{x}, A \supset A\tfrac{x2}{x} \vdash A$$

Proofs and derivations in PV1 are described in a way similar to PV.

We use the notation Cl(A) to mean the universal closure of A. We say a formula A of PV1 is true if Cl(A) is true in the domain of natural numbers, when the function symbols receive their standard meanings. The reader is warned that if the terms t and u have variables, then this interpretation means that $\neg t = u$ is not the negation

of $t = u$. For example, $\mathrm{sg}(x) = 0$ and $\neg\mathrm{sg}(x) = 0$ are both false, since their universal closures are both false in the natural numbers.

Theorem 3.10 An equation $t = u$ is a theorem of PV1 if and only if it is a theorem of PV.

The proof is omitted for lack of space.

As a measure of the power and usefulness of the system PV1, we prove the following result.

Theorem 3.11 If $A_1, \ldots, A_n, B$ are formulas in PV1, and $\mathrm{Cl}(B)$ can be derived from $\mathrm{Cl}(A_1), \ldots, \mathrm{Cl}(A_n)$ in the predicate calculus with equality, then $A_1, \ldots, A_n \vdash_{\mathrm{PV1}} B$.

Proof. Suppose the hypotheses of the theorem are satisfied. Then $\mathrm{Cl}(B)$ is a logical consequence of $\mathrm{Cl}(A_1), \ldots, \mathrm{Cl}(A_n), \mathrm{Cl}(E_1), \ldots, \mathrm{Cl}(E_4)$ in the predicate calculus (without the equality axioms). Thus $\mathrm{Cl}(A) \supset \mathrm{Cl}(B)$ is a quantificationally valid formula, where A is $(A_1 \,\&\, \ldots \,\&\, A_n \,\&\, E_1 \,\&\, \ldots \,\&\, E_4)$. By the Herbrand theorem (see [CL73]), there are substitutions $\sigma_1, \ldots, \sigma_k$ such that

$$\bigvee_{i=1}^{k} (A\sigma_i \supset B\tfrac{c_1,\ldots,c_r}{x_1,\ldots,x_r}) \tag{3.12}$$

is truth-functionally valid where $c_1, \ldots, c_r$ are new distinct constant symbols, $x_1, \ldots, x_r$ are the variables occurring in B, and each σ_i is a substitution of "ground" terms (built from $c_1, \ldots, c_r$ and constant symbols of PV by applying function symbols of PV) for the variables in A. If we let σ_i' be the substitution resulting when σ_i is followed by the substitution $\frac{x_1,\ldots,x_r}{c_1,\ldots,c_r}$, then the formula

$$\bigvee_{i=1}^{k} (A\sigma_i' \supset B) \tag{3.13}$$

is "isomorphic" to 3.12, and hence it is also truth-functionally valid. It follows, since 3.13 is a formula of PV1, that it is an axiom of PV1 (by TAUTOLOGY). Furthermore, by the rule SUBST, each of the formulas $E_1\sigma_i', \ldots, E_4\sigma_i', 1 \leq i \leq k$, is a theorem of PV1, and $A_1\sigma_i', \ldots, A_n\sigma_i', 1 \leq i \leq k$, can be derived in PV1 from the hypotheses $A_1, \ldots, A_n$. Hence, by the rule IMP, we see that $A_1, \ldots, A_n \vdash_{\mathrm{PV1}} B$.

4 The Gödel Incompleteness Theorem for PV

The main theorem in this section states that the consistency of PV cannot be proved in PV. This will be applied in section 7 to show that the system PV, as a proof system for the propositional calculus, is not p-verifiable.

It is easy to see that PV is incomplete, because the equivalence problem for functions in L is not recursively enumerable. But we need to know that a proof of this incompleteness can be given in PV itself so that we can follow Gödel's method of proving that a theory cannot have a proof of its own consistency.

The first step is to assign "Gödel numbers" to the terms, equations, and proofs in PV. Notice that an object of any of these three kinds has been defined to be a string of symbols. The underlying alphabet of symbols is infinite, because we assume there are an unlimited number of variables at our disposal. However, we can agree that a variable is just the symbol x followed by a finite string on the alphabet $\{1, 2\}$. Hence any term, equation, or proof, is a finite string on some fixed alphabet A of at most 32 symbols. We can code each symbol σ in A by a unique five-digit code $\psi(\sigma)$ over the alphabet $\{1, 2\}$. Then the Gödel number of a string $\sigma_1 \ldots \sigma_k$ is the number whose dyadic notation is $\psi(\sigma_1) \ldots \psi(\sigma_k)$. The number of an object C is denoted by $[C]$. The important property of Gödel numbers from our point of view is that an object C and the dyadic notation for $[C]$ can he obtained from each other within time bounded by polynomials in the lengths of $[C]$ and C, respectively.

We define the function *proof* on the natural numbers by

$$\text{proof}(m, n) = \begin{cases} 1 & \text{if } m \text{ is the number of an equation } t = u, \text{ and } n \text{ is} \\ & \text{the number of a proof in } PV \text{ of } t = u \\ 0 & \text{otherwise} \end{cases}$$

Next we define the function *sub* as follows: $\text{sub}(m) = n$ if $m = [t = u]$ and $n = [(t = u)\frac{\overline{m}}{x}]$, for some equation $t = u$, where $\overline{m}$ is the numeral for m. If m is not of the form $[t = u]$, then $\text{sub}(m) = 0$.

It is not hard to see that both the functions proof and sub can be computed in time bounded by a polynomial in the lengths of their arguments, so that both functions are in L. By theorem 2.12 there are function symbols PROOF and SUB in PV which define proof and sub, respectively. (We assume that the defining equations for these function symbols represent a straightforward algorithm for computing the functions.) Let

$$r = [\text{PROOF}(\text{SUB}(x), y) = 0] \tag{4.1}$$

Then

$$s = \text{sub}(r) = [\text{PROOF}(\text{SUB}(\overline{r}), y) = 0] \tag{4.2}$$

Thus equation number s says "I am not provable".

Theorem 4.3 Equation number s has no proof in PV.

Proof. Suppose, to the contrary, that p is the number of a proof of equation number s. By the valuation theorem (2.18), we have $\vdash_{\text{PV}} \text{PROOF}(\text{SUB}(\overline{r}), \overline{p}) = 1$. But

by assumption, $\vdash_{\mathrm{PV}} \mathrm{PROOF}(\mathrm{SUB}(\overline{r}), y) = 0$, so by the rules R4, R1, and R2 of PV, $\vdash_{\mathrm{PV}} 0 = 1$. This contradicts the consistency of PV (theorem 2.17), establishing the present theorem.

Now let CON(PV) stand for the equation $\mathrm{PROOF}([0 = 1], y) = 0$. This is a true equation of PV, asserting that the equation 0 = 1 has no proof in PV.

Theorem 4.4 CON(PV) has no proof in PV.

The idea, of course, is to show that the proof of theorem 4.3 can be formalized in PV. We will actually work in the system PV1, since this is easier. The first step is to formalize the valuation theorem (2.18) in PV1. The proof of 2.18 shows how to construct, for each function symbol f of PV, a function gen_f in L such that $\mathrm{gen}_f(n_1, \ldots, n_k, m)$ is the number of a proof in PV of the equation $f(\overline{n}_1, \ldots, \overline{n}_k) = \overline{m}$, provided the equation is true, and $\mathrm{gen}_f(n_1, \ldots, n_k, m) = 0$ otherwise. The function $\mathrm{form}_f(n_1, \ldots, n_k, m) = [f(\overline{n}_1, \ldots, \overline{n}_k) = \overline{m}]$ is certainly in L. It should be possible to show

Lemma 4.5 $\vdash_{\mathrm{PV}} f(x_1, \ldots, x_k) = y \supset \mathrm{PROOF}(\mathrm{FORM}_f(x_1, \ldots, x_k, y), \mathrm{GEN}_f(x_1, \ldots, x_k, y)) = 1$ for each function symbol f of PV, where FORM_f and GEN_f are the function symbols defining form_f and gen_f, respectively.

Now let us apply the lemma when f is PROOF, and substitute $\overline{s}$ (from 4.2) and 1 for two of the variables, to obtain

$$\vdash_{\mathrm{PV1}} \mathrm{PROOF}(\overline{s}, y) = 1 \supset \mathrm{PROOF}(\mathrm{FORM}(\overline{s}, y, 1), \mathrm{GEN}(\overline{s}, y, 1)) = 1 \tag{4.6}$$

where we have left off the subscripts on FORM and GEN. By definition, $\mathrm{form}_{\mathrm{PROOF}}(s, n, 1) = [\mathrm{PROOF}(\overline{s}, \overline{n}) = 1]$, and for each value of n, a proof (say number p) in PV of the equation in brackets together with a proof (say number q) in PV of formula number s (see 4.2) gives rise easily to a proof (say number $\mathrm{contra}(p, q, n)$) in PV of $0 = 1$. If we let CONTRA define the function contra, then one can prove the last statement in PV1.

Lemma 4.7 $\vdash_{\mathrm{PV1}} (\mathrm{PROOF}(\mathrm{FORM}(\overline{s}, y, 1), z) = 1 \& \mathrm{PROOF}(\overline{s}, u) = 1) \supset \mathrm{PROOF}([0 = 1], \mathrm{CONTRA}(z, u, y)) = 1$

Now lemma 4.7 with $\mathrm{GEN}(\overline{s}, y, 1)$ substituted for z and y substituted for u, together with 4.6 gives us immediately by the rule IMP of PV1

$$\vdash_{\mathrm{PV1}} \mathrm{PROOF}(\overline{s}, y) = 1 \supset \mathrm{PROOF}([0 = 1], t) = 1 \tag{4.8}$$

where t is $\mathrm{CONTRA}(\mathrm{GEN}(\overline{s}, y, 1), y, y))$.

By axiom E7 of PV1, $\vdash_{\text{PV1}} \neg 0 = 1$. Hence, by substituting t for y in the definition of CON(PV), we have by IMP and equality reasoning, from 4.8,

$$\vdash_{\text{PV1}} \text{CON}(PV) \supset \neg\text{PROOF}(\bar{s}, y) = 1 \tag{4.9}$$

Simple reasoning shows $\vdash_{\text{PV1}} \neg\text{PROOF}(x,y) = 1 \supset \text{PROOF}(x,y) = 0$, and since by the valuation theorem, $\vdash_{\text{PV}} \bar{s} = \text{SUB}(\bar{r})$, we have by 4.9

$$\vdash_{\text{PV1}} \text{CON}(PV) \supset \text{PROOF}(\text{SUB}(\bar{r}), y) = 0 \tag{4.10}$$

Thus, if $\vdash_{\text{PV}}$ CON(PV), then $\vdash_{\text{PV1}}$ CON(PV) (by theorem 3.10), so $\vdash_{\text{PV1}}$ PROOF(SUB($\bar{r}$),y) = 0 so $\vdash_{\text{PV}}$ PROOF(SUB($\bar{r}$),y) = 0 (again by 3.10), which contradicts theorem 4.3. This completes our outline of the proof of theorem 4.4.

5 Propositional Calculus and the Main Theorem

Propositional formulas will be formed in the usual way from the connectives $\&, \vee, \neg, \supset, \equiv$, and from an infinite list of atoms. We will define an *atom* to be the letters ATOM followed by a string on {1,2}, so that formulas are certain strings on a certain fixed finite alphabet. We can assign Gödel numbers to the strings as in section 4, and we will write $[A]$ for the number of the formula A. A *tautology* is a valid propositional formula, and we will use TAUT to denote the set of Gödel numbers of tautologies.

A *proof system* (for TAUT) is a function f in L from the set of natural numbers *onto* TAUT. (This differs from the definition in [CR74] in that numbers are used instead of strings.) If f is a proof system, and $f(x) = [A]$, then x is (or codes) a *proof* of A.

The paper [CR74] describes a large number of standard proof systems, and compares them from the point of view of length of proof. The system we are interested in here is a very powerful system called *extended resolution* (ER), which can efficiently simulate any of the standard systems, except possibly Frege systems with a substitution rule. The idea of extended resolution is due to Tseitin [Tse70].

The system ER can be defined as follows. A *literal* is an atom or a negation of an atom. The complement $\overline{L}$ of a literal L is given by $\overline{P} = \neg P$, $\neg\overline{P} = P$, where P is an atom. A *clause* is a disjunction $(L_1 \vee \ldots \vee L_k)$ of literals, $k \geq 0$, with no literal repeated. If $k = 0$ the clause (called the *empty clause*) is denoted by $\Box$. If A is a propositional formula, then we associate a literal L_B with every subformula B of A by the conditions (i) if B is an atom, then L_B is B, (ii) if B is $\neg C$, then L_B is $\overline{L_C}$, and (iii) if B is $(C \vee D), (C \,\&\, D), (C \supset D)$, or $(C \equiv D)$, then L_B is some uniquely associated with B.

If F is a propositional formula, then CNF(F) denotes some set of clauses whose conjunction is equivalent to F (and which is not unnecessarily long). Now we

associate with every propositional formula A a set $\mathrm{def}(A)$ of clauses by the conditions (i) $\mathrm{def}(P) = \emptyset$ if P is an atom, (ii) $\mathrm{def}(\neg B) = \mathrm{def}(B)$, (iii) $\mathrm{def}(B \circ C) = \mathrm{def}(B) \cup \mathrm{def}(C) \cup \mathrm{CNF}(L_{B \vee C} \equiv (L_B \circ L_C))$, where $\circ$ is $\&, \vee, \supset$, or $\equiv$.

Lemma 5.1

a) Any truth assignment τ to the atoms of A has a unique extension τ' to the atoms of $\mathrm{def}(A)$ which makes (each clause in) $\mathrm{def}(A)$ true. In fact, $\tau'(L_B) = \tau(B)$ for each subformula B of A, so in particular, $\tau'(L_A) = \tau(A)$.

b) A is a tautology if and only if L_A is a truth-functional consequence of $\mathrm{def}(A)$.

c) There is a function f in L which satisfies $f([A]) = [\mathrm{def}(A)]$.

Part a) is proved by induction on the length of A. Part b) follows immediately from a). For part c), observe that $\mathrm{def}(A)$ has at most three times as many clauses as A has connectives, and these clauses are easily found.

Notice that, in contrast to $\mathrm{def}(A)$, $\mathrm{CNF}(A)$ is not in general computable in polynomial time, simply because some formulas have a shortest conjunctive normal form which is exponential in their length. (For example, $(P_1 \,\&\, P_2 \vee \ldots \vee P_{2n-1} \,\&\, P_{2n})$).

If a clause C_1 is $(L_1 \vee \ldots \vee L_i \vee L \vee L_{i+1} \vee \ldots \vee L_k)$ and C_2 is $(M_1 \vee \ldots \vee M_j \vee \overline{L} \vee M_{j+1} \vee \ldots \vee M_\ell)$ then the *resolvent* of C_1 and C_2 is the clause which results from deleting repetitions of literals from $(L_1 \vee \ldots \vee L_k \vee M_1 \vee \ldots \vee M_\ell)$.

The *extension* rule for an atom P allows the introduction of the three or four clauses in $\mathrm{CNF}(P \equiv (L_1 \circ L_2))$, where $\circ$ is $\&, \vee, \supset, or \equiv$, provided P and $\overline{P}$ are distinct from L_1 and L_2. An ER *proof* of a formula A is a string $A^* C_1^* \ldots^* C_k^{**} C_{k+1}^* \ldots^* C_n$, where C_n is $L_A, \{C_1, \ldots, C_k\}$ are the clauses in $\mathrm{def}(A)$, and each C_i for $i > k$ is either a resolvent of two earlier C_j's, or is introduced by the extension rule for some atom P which has no earlier occurrence in the string. Any string not of the above form is, by convention, an ER proof of $(P \vee \neg P)$, for some fixed atom P. The proof system ER is the function such that $ER(n) = [A]$, provided the dyadic notation of n codes an ER proof of A. It is easy to see the function ER is in L.

It follows from lemma 5.1 and a slight modification of the usual completeness theorem for ground resolution (see [CL73], for example) that every tautology A has an ER proof in which the extension rule is not used. (The purpose of the extension rule is to give shorter proofs.) I now prove the converse explicitly, since I want to argue later that the proof can be formalized in PV.

Theorem 5.2 **Soundness of ER**

If a formula A has an ER proof, then A is a tautology.

Proof. If A is, $\neg P \vee P$, then A is obviously a tautology. Otherwise, the proof has the form $A^* C_1^* \ldots^* C_k^{**} C_{k+1}^* \ldots^* C_n$ described earlier. Let τ be any truth assignment to the atoms of A. Then, as mentioned in lemma 5.1, τ can be extended to a truth assignment τ' to the atoms L_B of $\mathrm{def}(A)$ such that τ' makes all clauses in $\mathrm{def}(A)$

true and $\tau'(L_A) = \tau(A)$. Hence τ' makes $C_1, \ldots, C_k$ true. Further, each time clauses D_1, D_2, D_3 are introduced by the extension rule for an atom P, τ' can be extended to τ'' whose domain includes P in such a way that D_1, D_2, D_3 are true under τ'' (for example, if the clauses are CNF$(P = (L_1 \vee L_2))$, then $\tau''(P)$ is $\tau'(L_1 \vee L_2)$). Thus there is an extension τ_1 of τ' which makes all clauses C_i introduced by extension true. It is easy to see that any truth assignment which makes two clauses true must make any resolvent of those clauses true. Hence, by induction on i, we see that τ_1 makes C_i true for $1 \leq i \leq n$. In particular, τ_1 makes $C_n = L_A$ true. Since $\tau_1(L_A) = \tau'(L_A) = \tau(A)$, τ makes A true. Since τ is an arbitrary truth assignment to A, A is a tautology.

The above argument shows more than just the soundness of ER. It shows that an ER proof of A provides a uniform method of checking rapidly that a given truth assignment satisfies A; namely check that τ_1 satisfies successively $C_1, C_2, \ldots, C_n = L_A$, and check successively that $\tau_1(B) \equiv \tau_1(L_B)$ for larger and larger subformulas B of A, and finally check that $\tau(A) = \tau_1(L_A) = \mathit{true}$. Thus ER is a *p-verifiable* proof system in the following sense.

Definition 5.3 **Informal definition**
A proof system F for TAUT is *p-verifiable* iff there is a polynomial $p(n)$ such that given a proof x in the system of a formula A, x gives a uniform way of verifying within $p(|x|)$ steps that an arbitrary truth assignment to A satisfies A.

It is easy to see that all the usual "Frege" systems (see [CR74]) for the propositional calculus satisfy this definition, in addition to ER. On the other hand, if the substitution rule (from A conclude $A\sigma$, where σ substitutes formulas for atoms) is added to Frege systems, then it is no longer clear that the system is p-verifiable. A proof of A in such a system does provide a way of verifying that a given truth assignment τ satisfies A, but since a formula B in the proof may have several substitution instances in the proof, and each of these instances can again have several instances, and so on, we may end up having to verify B for exponentially (in the length of the proof) many truth assignments to check that A comes out true under the single assignment τ. Also, there is no reason to think that a proof system for TAUT which incorporates Peano number theory or set theory is p-verifiable.

To make the notion of p-verifiable proof system precise, let us code a truth assignment τ as a string $(P_1, \tau(P_1)), (P_2, \tau(P_2)), \ldots, (P_k, \tau(P_k))$ listing the atoms in its domain and the truth value assigned to these atoms. This string in turn can be coded as a string on $\{1, 2\}$, and $[\tau]$ will denote the number whose dyadic notation

is this last string. Then we can define a function tr in L such that

$$\mathrm{tr}([A],[\tau]) = \begin{cases} 1 & \text{if } \tau(A) \text{ is true} \\ 0 & \text{if } \tau(A) \text{ is false} \end{cases}$$

We can make the convention that τ assigns false to all atoms of A for which a value is not explicitly given, so that $\tau(A)$ is defined for any formula A and truth assignment τ and every number n codes some truth assignment. Let TR be a function symbol in PV which defines tr.

Definition 5.4 **Formal Definition**
A proof system f for TAUT is *p-verifiable* iff there is some function symbol F in PV defining f such that $\vdash_{\mathrm{PV}} \mathrm{TR}(F(x),y) = 1$.

It is worth pointing out that this formal definition depends on the particular function symbol TR chosen to define tr. That is, it depends on the algorithm chosen to compute tr. Presumably, if TR and TR′ both represent straightforward algorithms for computing tr, then $\vdash_{\mathrm{PV}} \mathrm{TR}(x,y) = \mathrm{TR}'(x,y)$, so definition 5.4 would be the same for TR and TR′.

The formal definition requires that the soundness of f be provable in PV. If one believes the verifiability thesis (1.1), then it is easy to see that the formal definition captures the informal one.

In [CR74], a notion of one proof system simulating another is defined. Here I would like to sharpen that notion and say that a proof system f_1 *p-simulates* a proof system f_2 iff there is a function g in L such that $f_2(n) = f_1(g(n))$ for all n. Further, f_1 *p-verifiably simulates* f_2 iff there exist functions symbols F_1, F_2 in PV defining f_1, f_2, respectively, and a function symbol G such that $\vdash_{\mathrm{PV}} F_2(x) = F_1(G(x))$.

Now I can state the main theorem of this paper, which characterizes the p-verifiable proof systems.

Theorem 5.5 **Main Theorem**
A proof system f for tautologies is p-verifiable if and only if extended resolution p-verifiably simulates f.

Theorem 5.6 Extended resolution p-verifiably simulates any Frege system (see [CR74]).

Corollary 5.7 Every Frege system is a p-verifiable proof system.

Theorem 5.6 can be proved by formalizing in PV the proof in [CR74] which shows that ER simulates any Frege system. The argument will not be given here.

The following lemma is needed for the Main Theorem.

Lemma 5.8 ER is p-verifiable. That is, $\vdash_{\mathrm{PV}} \mathrm{TR}(\mathrm{EXTRES}(x),y) = 1$, where EXTRES is a suitable function symbol in PV defining ER.

The proof amounts to showing the proof of 5.2 (Soundness of ER) can be formalized in PV. (Of course, in practice it is easier to work in PV1.) Thus one defines a function tauone(n) in L such that when $n = [\tau]$, then tauone(n) $= [\tau_1]$, where τ_1 is the truth assignment described in that argument. Then the formal versions of the equations $\tau_1(L_A) = \tau'(L_A) = \tau(A)$ are provable in PV, and TR(ER(x), y) $= 1$ follows. The details are omitted.

The "if" part of the Main Theorem follows easily from the lemma. For suppose ER p-verifiably simulates f. Then $\vdash_{\mathrm{PV}} F(x) = \mathrm{EXTRES}(G(x))$, where F defines f. If rule R3 of PV (with TR for f) is applied to this equation and the result applied with transitivity to 5.8 with $G(x)$ for x, we obtain $\vdash_{\mathrm{PV}} \mathrm{TR}(F(x), y) = 1$. Hence f is p-verifiable.

The converse to the Main Theorem is more difficult and will be dealt with in the next section.

6 Propositional Formulas Assigned to Equations of PV

To prove the "only if" part of theorem 5.5 I propose to first prove that extended resolution can p-simulate any p-verifiable proof system, and then argue that this proof can be formalized in PV. This first proof is carried out by assigning, for each m, a propositional formula to each equation $t = u$ which says, roughly speaking, "the equation holds when variables are restricted so that the dyadic notations for all relevant functions have length at most m". I then argue that if $\vdash_{\mathrm{PV}} t = u$, then there is an ER proof of the formula whose length is bounded by a polynomial in n. Applying this result to the equation $\mathrm{TR}(F(x), y) = 1$ (which is provable in PV if F represents a p-verifiable proof system f), one can see that there is an ER proof of formula number $f(n)$ which is not much longer than the proof n.

Proceeding more formally, let us fix the integer m > 0. We associate with every term t of PV the atoms $P_0[t], P_1[t], \ldots, P_m[t]$ *and* $Q_0[t], Q_1[t], \ldots, Q_m[t]$. We will call these the *atoms of* t. The intended meanings are

$$P_i[t] \equiv \begin{cases} \text{true if } i\text{th dyadic digit (i.e. coefficient of } 2^i\text{) of } t \text{ is 2} \\ \text{false if this digit is 1 irrelevant if the dyadic length of } t \text{ is } < i+1 \end{cases}$$

$$Q_i[t] \equiv \begin{cases} \text{true if coefficient of } 2^i \text{ in } t \text{ is defined (i.e. } t \geq 2^{i+1} - 1) \\ \text{false otherwise} \end{cases}$$

Now we can define, for each term t and each truth assignment τ to the atoms of t which satisfies $Q_i[t] \supset Q_{i-1}[t]$, $1 \leq i \leq m$, a number $\mathrm{val}_m(t, \tau)$ which is the number whose dyadic notation is determined by these intended meanings. Next we associate a propositional formula $\mathrm{prop}_m[t]$ with the term t (the subscript m will

sometimes be omitted). Among the atoms of this formula are some of the atoms of t and the atoms of the variables which occur in t. This formula has the following property:

6.1 Semantic Correctness of prop_m

Let the term t of PV with variables $x_1, \ldots, x_n$ define the function $f(x_1, \ldots, x_n)$, and let τ be a truth assignment which satisfies $\text{prop}_m[t]$ and such that when f is evaluated (according to the defining equations in PV) at $x_i = \text{val}_m(x_i, \tau), 1 \leq i \leq n$, no value of any number appearing in the computation exceeds m in dyadic length. Then $\text{val}_m(t, \tau) = f(\text{val}_m(x_1, \tau), \ldots, \text{val}_m(x_n, \tau))$.

To define $\text{prop}_m[t]$ in general in such a way that 6.1 holds, we start with the following special cases.

$$\text{prop}_m[x] \text{ is } \mathop{\&}_{i=1}^{m} Q_i[x] \supset Q_{i-1}[x], \text{ for each variable } x. \tag{6.2}$$

$$\text{prop}_m[0] \text{ is } \mathop{\&}_{i=0}^{m} \neg Q_i[0] \tag{6.3}$$

$$\text{prop}_m[\text{s}_1(x)] \text{ is} \tag{6.4}$$

$$(\text{prop}_m[x] \mathop{\&}_{i=0}^{m-1} (P_{i+1}[\text{s}_1(x)] \equiv P_i[x])$$

$$\&\neg P_0[\text{s}_1(x)]$$

$$\&Q_0[\text{s}_1(x)] \mathop{\&}_{i=0}^{m-1} (Q_{i+1}[\text{s}_1(x)] \equiv Q_i[x]))$$

$$\text{prop}_m[\text{s}_2(x)] \text{ is defined similarly.} \tag{6.5}$$

Let $\sigma = \frac{t_1,\ldots,t_k}{x_1,\ldots,x_k}$ be a substitution (regarded as a transformation) of terms for variables. The function ψ takes a substitution and an atom of t into an atom of $t\sigma$, and is defined by the equations $\psi(\sigma, P_i[t]) = P_i[t\sigma]$, and $\psi(\sigma, Q_i[t]) = Q_i[t\sigma]$, where $t\sigma$ is the term resulting when σ is applied to t. ψ can be extended in an obvious way so that its second argument is any propositional formula in the atoms of various terms t. Thus $\psi(\sigma, \neg A) = \neg\psi(\sigma, A)$, and $\psi(\sigma, (A \circ B)) = (\psi(\sigma, A) \circ, A) \circ \psi(\sigma, B))$, where $\circ$ *is* $\&, \vee, \supset$, *or* $\equiv$. The formulas prop_m will satisfy the following property:

Substitution Principle 6.6 Let $\sigma = \frac{t_1,\ldots,t_k}{x_1,\ldots,x_k}$. Then $\text{prop}_m[t\sigma] \Leftrightarrow \mathop{\&}_{i=1}^{k} \text{prop}_m[t_i] \& \psi\,(\sigma, \text{prop}_m[t])$, where $\Leftrightarrow$ can be read "is truth-functionally equivalent to".

For example, if t is $\text{s}_1(x)$ and σ is $\frac{0}{x}$, then this principle and 6.3, 6.4, say that $\text{prop}_m[\text{s}_1(0)]$ is a conjunction of formulas, including $\neg Q_i[0]$, $1 \leq i \leq m$, and $\neg P_0[\text{s}_1(0)]$, and $Q_0[\text{s}_1(0)]$, and $Q_{i+1}[\text{s}_1(0)] \equiv Q_i[0]$, $0 \leq i \leq m-1$. These formulas imply $\neg P_0[\text{s}_1(0)], Q_0[\text{s}_1(0)]$, and $\neg Q_i[\text{s}_1(0)], 1 \leq i \leq m$, which completely specify the dyadic notation for $\text{s}_1(0)$ (=1).

Now suppose $\text{prop}_m[f(x_1, \ldots, x_k)]$ has been defined for all function symbols f is a certain set S. Then we can inductively define $\text{prop}_m[t]$ for each term t built from 0, variables, and function symbols in S by

$$\text{prop}_m[f(t_1, \ldots, t_k)] = \mathop{\&}_{i=1}^{k} \text{prop}_m[t_i] \;\&\; \psi(\frac{t_1, \ldots, t_k}{x_1, \ldots, x_k}, \text{prop}_m[f(x_1, \ldots, x_k)]) \tag{6.7}$$

To complete the definition of $\text{prop}_m[t]$ for all terms t, it suffices to show how to define $\text{prop}_m[f(x_1, \ldots, x_n)]$ for each of the two ways of defining new function symbols. First, suppose f is $\lambda x_1, \ldots, x_n t\rho$, where $\text{prop}_m[t]$ has been defined. Then the defining equation for f is $f(x_1, \ldots, x_n) = t$, and we define

$$\begin{aligned} &\text{prop}_m[f(x_1, \ldots, x_n)] \text{ is} \\ &(\text{prop}_m[t] \mathop{\&}_{i=0}^{m} (P_i[f(x_1, \ldots, x_n)] \equiv P_i[t]) \\ &\mathop{\&}_{i=0}^{m} (Q_i[f(x_1, \ldots, x_n)] \equiv Q_i[t])) \end{aligned} \tag{6.8}$$

The case in which f is defined by recursion on notation is more complicated, and is omitted for lack of space. This completes the definition of $\text{prop}_m[t]$, for all terms t.

Now suppose $x_1, \ldots, x_r$ is a list of all the variables appearing in the terms t and u, and n, m are positive integers with $n \leq m$. Then $|t = u|_m^n$ is the propositional formula

$$\begin{aligned} &((\text{prop}_m[t] \;\&\; \text{prop}_m[u]) \;\&\; (\neg Q_{n+1}[x_1] \;\&\; \ldots \;\&\neg Q_{n+1}[x_r])) \supset \\ &(\mathop{\&}_{i=0}^{m} Q_i[t] \supset (P_i[t] \equiv P_i[u]) \mathop{\&}_{i=0}^{m} (Q_i[t] \equiv Q_i[u])) \end{aligned}$$

We say that m is a *bounding value* for n *relative* to $t = u$ if the terms t and u can be evaluated by the relevant defining equations for all values of their variables of dyadic length n or less without having any value in the computation exceed m in dyadic length.

Theorem 6.8 **ER Simulation Theorem**
Suppose Π is a proof in PV of $t = u$. Then there is a polynomial $p(m)$ (depending on Π) such that for all n, m, if m is a bounding value for n relative to $t = u$, then $|t = u|_m^n$ has an extended resolution proof of length at most $p(m)$.

The proof is by induction on the length of Π, and is omitted.

Using this theorem, we can sketch the proof of the "only if" part of theorem 5.5. Thus suppose f is a p-verifiable proof system, and suppose F is a function symbol in PV which defines f, such that $\vdash_{\text{PV}} \text{TR}(F(x), y) = 1$. Since all functions

used in defining F and TR are in L, it follows that one can find a polynomial q with natural number coefficients such that for all n, $q(n)$ is a bounding value for n relative to $\mathrm{TR}(F(x),y) = 1$, and $q(n) \to \infty$. By theorem 6.8, there is a polynomial $p(n)$ such that $|\mathrm{TR}(F(x),y) = 1|^n_{q(n)}$ has an ER proof of length at most $p(q(n))$, for all n.

Now let $P_1,\ldots,P_k$ be the atoms of some propositional formula A. A truth assignment τ to these atoms determines a number $[\tau]$ in a straightforward manner, and using a variable y for $[\tau]$, we can use the extension rule to introduce a set CL of clauses defining the atoms $P_i[y]$ and $Q_i[y]$ in terms of the atoms $P_1,\ldots,P_k$. Thus any truth assignment τ' which satisfies all clauses in CL must have the property that if τ is the restriction of τ' to $P_1,\ldots,P_k$, then $[\tau]$ is the value of y whose dyadic notation is represented by $\tau'(P_i[y]),\tau'(Q_i[y]),1 \le i \le m$, for suitable m. Since any truth assignment τ'' satisfying $\mathrm{def}(A)$ must have $\tau''(A) = \tau''(L_A)$, one can see from the way TR is defined that there is an ER derivation of $L_A \equiv Q_0[\mathrm{TR}([\overline{A}],y)]$ from $\mathrm{def}(A)$, CL, and, $\mathrm{prop}_m[\mathrm{TR}([\overline{A}],y)]$, for suitable m. Further, there is a polynomial $r(n)$ such that for all formulas A, this ER derivation has length at most $r([A])$.

Now suppose A has a proof a in the system f; that is, suppose $f(a) = [A]$. Then by the valuation theorem 2.18, $\vdash_{\mathrm{PV}} F(\overline{a}) = [\overline{A}]$, and one can verify that $|F(\overline{a}) = [\overline{A}]|^n_{q_1(n)}$ has an ER proof of length not exceeding $p_1(q_1(n))$, where $n = |a|$, for some polynomials P_1 and q_1. Putting this ER proof together with the ones in the preceding two paragraphs, and noting that the clauses in CL, $\mathrm{def}(\mathrm{prop}_m[t])$ for all terms t involved can be introduced by the extension rule, we come up with an ER proof $g(a)$ of A of length not exceeding $p_2(|a|)$, for some polynomial P_2. Thus $\mathrm{ER}(g(a)) = f(a)$ for all a, and since $g(a)$ is in L, ER p-simulates f.

To complete the proof of theorem 5.5 it is necessary to show $\vdash_{\mathrm{PV}} \mathrm{ER}(G(x)) = F(x)$, where G is a function symbol in PV defining g. This amounts to showing the above argument can be formalized in PV, which I will not do here. It is not hard to check, however, that the above argument is feasibly constructive, so that if one believes the verifiability thesis (1.1), the formalization is not necessary.

7 PV as a Propositional Proof System

Any formal system for number theory can be treated as a proof system for TAUT by regarding a proof of the formalization of $\mathrm{tr}([A],y) = 1$ as a proof of A. In particular, if Π is a proof in PV of $\mathrm{TR}([\overline{A}],y) = 1$, then Π is a proof in PV of A. We can define a function pv in L which satisfies $\mathrm{pv}([\Pi]) = [A]$ if Π is a proof of A. Thus pv is a proof system for TAUT in the general sense defined in section 5.

Theorem 7.1 The system pv is not p-verifiable.

Lemma 7.2 $\vdash_{\mathrm{PV1}} \mathrm{TR}(\mathrm{PV}(x), y) = 1 \supset \mathrm{PROOF}([0 = 1], z) = 0$

The lemma says that the statement "if pv is p-verifiable, then PV is consistent" is provable in PV1. I prove the lemma by giving an informal argument for the implication which is readily formalized in PV1, using the techniques of section 4.

By hypothesis,

(1) $\mathrm{tr}(\mathrm{pv}(m), n) = 1$ for all m and n.

It is not hard to see that

(2) $\vdash_{\mathrm{PV}} \mathrm{TR}([P \,\&\, \neg P], y) = 0$.

Now suppose PV is inconsistent, so that

(3) $\vdash_{\mathrm{PV}} 0 = 1$.

Then from (2) and (3),

(4) $\vdash_{\mathrm{PV}} \mathrm{TR}([P \,\&\, \neg P], y) = 1$.

If Π is a proof in PV of (4), then

(5) $\mathrm{pv}([\Pi]) = [P \,\&\, \neg P]$.

Combining (1) and (5), we have

(6) $\mathrm{tr}([P \,\&\, \neg P], n) = 1$, for all n.

But (6) is absurd, since $\mathrm{tr}([P \,\&\, \neg P], n) = 0$. Hence our assumption that PV is inconsistent is untenable, so PV is consistent.

From lemma 7.2, we see that if $\vdash_{\mathrm{PV}} \mathrm{TR}(\mathrm{PV}(x), y) = 1$, then $\vdash_{\mathrm{PV1}}, \mathrm{CON}(\mathrm{PV})$, so $\vdash_{\mathrm{PV}} \mathrm{CON}(\mathrm{PV})$, violating theorem 4.4. Therefore pv is not p-verifiable.

8 Conclusions and Future Research

(1) There should be alternative formalizations of PV. These would make the verifiability thesis (1.1) more convincing and make it easier to formalize arguments in PV. One such formalization should be a programming approach, where proving $f(x) = g(x)$ amounts to proving the equivalence of two programs.

(2) If one believes that all feasibly constructive arguments can be formalized in PV, then it is worthwhile seeing which parts of mathematics can be so formalized. I think that a good part of elementary number theory (such as the unique factorization theorem) can be formalized in PV, although the results will have to be formulated carefully. For example, the function $p_1^{e_1} p_2^{e_2} \ldots p_k^{e_k}$ is not in L and so it is not definable in PV. However, the relation $n = p_1^{e_1} p_2^{e_2} \ldots p_k^{e_k}$ *is* an L-relation, and its characteristic function is definable in PV. As another example of formulation problems, it is hard to see at first how to formulate in PV the completeness of a proof

system for TAUT such as ER, since there is no function g in L taking an arbitrary tautology number $[A]$ into an ER proof of A (unless $P = NP$).

However, there is a function g in L which takes a code for a tautology A together with a list $\tau_1, \ldots, \tau_k$ of all truth assignments to A into an ER proof of A, and the equation $ER(G([A^* \tau_1^* \ldots^* \tau_k])) = [A]$ should be provable in PV. This formulation of completeness says that given a formula A, together with a verification that all truth assignments to A make A true, one can find an ER proof of A. This statement certainly incorporates the information that every tautology has an ER proof.

(3) The question that lead me to the system PV in the first place is the question of whether extended resolution is a super proof system. I conjecture that it is not. A possible approach to showing this is by proving some sort of converse to the ER simulation theorem (6.8). Specifically, I conjecture that the propositional formulas $|\text{CON(PV)}|^n_{q(n)}$ have no ER proofs bounded in length by a polynomial in n, where $q(n)$ is a bounding value for n relative to CON(PV).

(4) It would be interesting to prove that a Frege system with substitution (see [CR74]) is not p-verifiable. A likely approach is to show that such a system p-verifiably simulates pv, which would mean that if such a system were p-verifiable, so would be pv, violating theorem 7.1.

Acknowledgments

I would like to thank Robert Constable for helpful discussions concerning constructive mathematics, and for a critical reading of the first half of this manuscript. I would like to thank Martin Dowd for helping with the technical aspects of k-recursion and k-induction on notation.

References

[CL73] C. L. Chang and C. T. Lee. 1973. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press.

[CR67] J. P. Cleave and H. E. Rose. 1967. E^n-arithmetic. In Crossley (Ed.), *Sets, Models, and Recursion Theory*. North-Holland, Amsterdam.

[Cob65] A. Cobham. 1965. The intrinsic computational difficulty of functions. In *Proceedings of the 1964 International Congress for Logic, Methodology, and Philosophy of Science*. North-Holland, Amsterdam, 24–30.

[Coo71] S. A. Cook. 1971. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, May 3–5, 1971, Shaker Heights, Ohio. ACM, 151–158. DOI: https://doi.org/10.1145/800157.805047.

[CR74] S. A. Cook and R. A. Reckhow. 1974. On the lengths of proofs in the propositional calculus (preliminary version). In *Proceedings of the 6th Annual ACM Symposium*

on Theory of Computing, April 30–May 2, 1974, Seattle, Washington. ACM, 135–148. DOI: https://doi.org/10.1145/800119.803893.

[Goo57] R. L. Goodstein. 1957. *Recursive Number Theory*. North-Holland, Amsterdam.

[Kar72] R. M. Karp. 1972. Reducibility among combinatorial problems. In *Proceedings of a Symposium on the Complexity of Computer Computations*, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York. Springer, 85–103. DOI: https://doi.org/10.1007/978-1-4684-2001-2_9.

[Kle52] S. C. Kleene. 1952. *Introduction to Metamathematics*. D. Van Nostrand, New York, NY.

[Las67] D. Lascar. March. 1967. Cobham's characterization of *L*. In *Topics in the Theory of Computation.* notes of a seminar conducted by R. M. Baer and S. A. Cook, Dept. of mathematics, University of California at Berkeley. (Unpublished).

[Par71] R. Parikh. 1971. Existence and feasibility in arithmetic. *J. Symb. Log.* 36, 3, 494–508. DOI: https://doi.org/10.2307/2269958.

[Ros61] H. E. Rose. 1961. On the consistency and undecidability of recursive arithmetic. *Zeitschr. f. math. Logik and Grundlogen d. Math. Bd.* 7, S. 124–135.

[Sko23] T. Skolem. 1923. *Begründung der Elementaren Arithmetik*. Videnskapsselskapets Skrifter. I. Matematisk-naturvidenskabelig Klasse. No. 6. Utgit for Fridtjof Nansens Fond.

[Smu61] R. M. Smullyan. 1961. *Theory of Formal Systems, Revised Edition*. Annals of Mathematical Studies. Vol. 47, Princeton.

[Tse70] G. S. Tseitin. 1970. On the complexity of derivation in propositional calculus. *Semin. Math., V. A. Steklov Math. Inst., Leningrad* 8, 115–125; translation from 1968. *Zap. Nauchn. Semin. Leningr. Otd. Mat. Inst. Steklova* 8, 234–259.

Towards a Complexity Theory of Synchronous Parallel Computation

Stephen A. Cook

Abstract

This is largely an expository paper on the general theory of synchronous parallel computation. The models of parallel computers discussed include uniform circuit families, alternating Turing machines, conglomerates, vector machines, and parallel random access machines. A classification of these models indicates the need for still more; so "aggregates" and "hardware modification machines" are introduced. The resources sequential time, space, parallel time, circuit size and depth, hardware size etc., are discussed and interrelated. Work in progress at Toronto is mentioned and basic open questions are listed.

1 Introduction

There is now a well developed computational complexity theory of sequential computation. The precisely "right" computer model is not completely clear, but the main contenders for this model do not differ markedly from each other in their computing efficiency. These contenders are multitape Turing machines, possibly with storage structures more general than linear tapes, and various versions of random access machines. Of these models, the storage modification machine (SMM) made popular by Schönhage [Sch79] carries the most conviction as a stable and

Presented at the *Symposium über Logik und Algorithmik* in honour of Ernst Specker, Zürich, February 1980. https://www.e-periodica.ch/digbib/view?pid=ens-001%3A1981%3A27
L'Enseignement Mathematique XXVII: 99–124 (1981). https://www.e-periodica.ch/digbib/view?pid=ens-001%3A1981%3A27#257
Original DOI: 10.5169/seals-51742

general model of a sequential computer; where we take sequential to mean the number of active elements is bounded in time.

To be sure, there is a feeling that one step of an SMM may be a little too powerful. It is hard to imagine a mechanism for reconnecting a given edge out of a node ν_1 in the storage structure to a node ν_2 in one step, when the candidates for ν_2 from the perspective of the whole computation are unlimited. But the fact remains that if we restrict ourselves to fixed storage structures, there is no single structure or class of structures which seems to be just right. (Certainly multitape Turing machines are too restrictive.) On the other hand, for random access machine models one is never quite sure which set of operations should be primitive, and whether to charge more than one time unit for an operation capable of manipulating arbitrarily large integers.

Whatever the sequential model, it is clear that the main resources of interest are time and space. Let me repeat that the differences among the leading models in the time and space needed to execute algorithms are minor. And the theory of sequential time and space complexity is a rich and interesting one.

In the past few years it has become increasingly clear that the most powerful computers of the future will not be sequential but parallel. An entire processor can now be placed on a VLSI chip that is so small and cheap that it is not hard to imagine a machine of the future consisting of millions of such processors connected together and operating synchronously. The questions then become: How should the machine be organized and what can be done with the result? Hence the need for a theory of parallel computation. (A second motivation, of course, is that the human brain appears to be a parallel computer.)

I should point out here that the theory I have in mind deals only with synchronous computers. There is indeed a great and interesting literature on asynchronous processes, and the theory has applications when the processes in question cannot easily be synchronized (such as distributed computer systems or operating systems). The theory discussed here assumes one parallel computer whose elements have been designed from scratch to operate synchronously.

The first problem in this theory is to find the right mathematical model of a parallel computer. The parallel models in the literature fall roughly into two classes: those with fixed structure and those with modifiable structure. The fixed structure parallel models correspond to sequential machines with fixed storage structure, namely Turing machines with "tapes" which may be more general than linear arrays, but cannot be modified. The parallel analogs of these include Borodin's uniform circuit families [Bor77], Goldschlager's conglomerates [Gol78], [Gol77], and Hoover's uniform infinite circuits [Hoo79].

The modifiable sequential machines include SMM's and random access machines (RAM's). (Indirect addressing in a RAM gives the effect of a modifiable storage structure, and in fact RAM's which can only add and subtract one are equivalent to SMM's [Sch79].) The modifiable parallel machines include various parallel RAM's, such as SIMDAG's [Gol78] and P-RAM's [SS79] and [FW78], as well as vector machines as defined in [PS78]. As yet no parallel analog of SMM's has appeared, but a tentative candidate is introduced in Section 5.

Fortunately, all these models are roughly equivalent from the point of view of computation time, in the sense that each can simulate another while at most cubing the computation time. In fact, the sets or functions computed by each in time $S^{O(1)}$ (i.e. time polynomial in S: this notation appears in [Pip79]) are the same as those computed by a Turing machine in space $S^{O(1)}$ for any well behaved time bound S. (This phenomenon was observed, for example, in [CS76], and called the "parallel computation thesis" in [Gol78].)

This thesis can be made more specific as follows: For the fixed structure parallel machines; namely, uniform circuit families, conglomerates, "aggregates" (see Section 4) and uniform infinite circuits (see [Hoo79]),

$$\text{parallel time } (S) \subseteq \text{DSPACE } (S) \subseteq \text{NSPACE } (S) \subseteq \text{parallel time } (S^2) \tag{1.1}$$

(See [HU79] for the meaning of DSPACE and NSPACE.)

On the other hand, the modifiable parallel machines tend to be more powerful, and the inclusions become (at least for SIMDAG's and the P-RAM's of [FW78]):

$$\text{parallel time } (S) \subseteq \text{DSPACE } (S^2) \subseteq \text{parallel time } (S^2). \tag{1.2}$$

(For SIMDAG's, the stronger statement NSPACE $(S) \subseteq$ parallel time (S) also holds [Gol78]).

The modifiable parallel models that have been proposed so far all share the same problem as the sequential RAM models: The choice of primitive operations seems arbitrary, and most of these operations (such as shifts in vector machines and random access to global storage in P-RAM's) seem too powerful to be primitive. Hence I propose a new modifiable parallel model: "Hardware Modification Machines" (HMM's), to be the parallel analog of SMM's. These are discussed in Section 5.

Time and space are the fundamental resources in sequential complexity theory. What are their analogs in the parallel theory? Obviously, parallel time plays a fundamental role. The second important parallel resource, I think, should be hardware size; that is, the number of elements of a machine which are active during

a computation. For conglomerates, hardware size is the number of active finite state machines, and for vector machines it is the sum of the lengths of the vectors. For SIMDAG's and P-RAM's it corresponds roughly to the number of processors, although it should take into account the total memory used. For circuits, the circuit size is an upper bound on hardware size, but the traditional restriction that circuits are acyclic disallows elements to be reused during a computation and hence may give an unrealistically large value for size. Hence "aggregates" are introduced in Section 4. These can be thought of either as circuits with cycles, or as finite conglomerates.

Section 2 discusses two fundamental fixed structure parallel models; namely, uniform circuit families and alternating Turing machines. These turn out to be nearly equivalent. Section 3 gives examples which are log depth complete for deterministic log space, and hence may distinguish between two similar classes: deterministic log space and uniform log circuit depth. Section 4 discusses two fixed structure models useful for considering hardware size as well as parallel time; namely, conglomerates and aggregates. Section 5 introduces hardware modification machines, and Section 6 surveys other modifiable parallel models, such as vector machines and parallel RAM's. Section 7 discusses characterizations and interrelationships between two complexity classes defined by simultaneous resource bounds; namely, NC and SC Finally, Section 8 lists some open problems.

2 Circuits and Alternating Turing Machines

Perhaps the simplest model for measuring the parallel time to compute a function is the combinational circuit (or simply a circuit). (See [Sav76] and [Pat76] for general discussions of circuits.)

Notation $B_n = \{f \mid \{0,1\}^n \to \{0,1\}\}$ = the set of all Boolean functions of rank n.

Definition A *circuit* α *with n inputs* is a finite directed acyclic graph such that each node has a label from $\{x_1, \ldots, x_n\} \cup B_0 \cup B_1 \cup B_2$. A node labelled x_i must have indegree zero, and is called an *input* node. A node ν with label $g \in B_i$ must have indegree i, and one edge into ν is associated with each argument of g. Certain nodes are designated *output* nodes. When the variables x_i are assigned values from $\{0,1\}$ every node ν assumes a unique value in $\{0,1\}$, so that ν computes some function f_ν of $x_1, \ldots, x_n$. We say the circuit α *computes* f if $f = f_\nu$ for some output node ν.

We shall assume that every node ν has a path from ν to some output. That is, we assume there are no syntactically superfluous nodes.

Let $c(\alpha)$ (the complexity of α) be the number of *gates* (i.e. nodes other than inputs) in α, and let $d(\alpha)$ (depth of α) be the length of the longest path in α. If $f \in B_n$, then $c(f) = \min\{c(\alpha) \mid \alpha \text{ computes } f\}$ and $d(f) = \min\{d(\alpha) \mid \alpha \text{computes } f\}$.

If $A \subseteq \{0,1\}^*$, then $A^n = A \cap \{0,1\}^n$. We can regard A^n as a member of B_n by the convention $A^n\ (x_1, \ldots, x_n) = 1$ iff $(x_1 \ldots x_n) \in A^n$. A family $\{\alpha_n\}$ of circuits *computes* A iff α_n computes A^n for all n, and each α_n has a unique output node.

Notation Let $S, T : \mathbb{N}^+ \to \mathbb{R}$. Then

$$\begin{aligned} \text{SIZE}\ (T) &= \{A \mid \exists\ \{\alpha_n\} : \{\alpha_n\} \text{ computes } A \text{ and } c(\alpha_n) = \text{O}(T(n))\} \\ \text{DEPTH}\ (S) &= \{A \mid \exists\ \{\alpha_n\} : \{\alpha_n\} \text{ computes } A \text{ and } d(\alpha_n) = \text{O}(S(n))\} \end{aligned}$$

We shall always assume $T(n) \geqslant n$ and $S(n) \geqslant \log n$.

These complexity classes are strange in that they include nonrecursive sets A. In fact, by Lupanov's result (see [Sav76]) SIZE $(2^n/n) = 2^{\{0,1\}^*}$, and by disjunctive normal form DEPTH$(n) = 2^{\{0,1\}^*}$. Nevertheless they are mathematically interesting, and have intuitive significance especially for lower bound results. In particular, a proof that $A \notin$ DEPTH (S) means that no parallel computer with fixed circuitry could compute A in time O(S). This is because the parallel computation could be unwound to form a circuit with constant delay at each gate. Our assumption that circuits have bounded fan-in (in fact fan-in two) is justified by engineering experience that any general design for a gate with n inputs has a delay at least proportional to $\log n$. On the other hand, Hoover [Hoo79] gives results that show that an assumption of fan-*out* two would not materially alter either the depth or the size complexity of a set A.

Although the circuit depth to compute A is a reasonable lower bound on the parallel time required, it is not a reasonable upper bound in general (unless we want parallel machines to compute nonrecursive sets). Borodin [Bor77] proposed making it reasonable by requiring that the family $\{\alpha_n\}$ computing A be uniform in some sense. The trouble is there is no clearly correct choice for the definition of *uniform*. (See Ruzzo [Ruz79a] for a discussion of various possibilities.) Here we shall adopt the following definition, which has gained some acceptance (see [Coo79], [Ruz79a], and [Pip79]):

Definition A family $\{\alpha_n\}$ of circuits is *uniform* provided some deterministic Turing machine can compute the transformation $1^n \to \bar{\alpha}_n$ in space O$(\log c(\alpha_n))$. (Here $\bar{\alpha}_n$ is a binary string coding the circuit α_n in some reasonable fashion.)

We can now define the uniform complexity classes

$$\begin{aligned} \text{USIZE}\ (T) &= \{A \mid \exists \text{ uniform } \{\alpha_n\} : \{\alpha_n\} \text{ computes } A \text{ and } c(\alpha_n) \\ &= \text{O}(T(n))\} \\ \text{UDEPTH}\ (S) &= \{A \mid \exists \text{ uniform } \{\alpha_n\} : \{\alpha_n\} \text{ computes } A \text{ and } d(\alpha_n) \\ &= \text{O}(S(n))\} \end{aligned}$$

Notice that the size $c(\alpha_n)$ is not mentioned in the definition of UDEPTH so it can be taken to be as large as possible consistent with $d(\alpha_n)$. In fact, every circuit of depth d with a unique output can be expanded into an equivalent tree circuit of size $2^d - 1$. Also, our assumption of no superfluous nodes implies that no unique-output circuit of depth d can have more than $2^d - 1$ nodes. This leads to the following

Proposition 2.1 *The class* UDEPTH (S) *remains unchanged if the definition of uniform family* $\{\alpha_n\}$ *is changed to require that the transformation* $1^n \to \bar{\alpha}_n$ *be computable in deterministic space* $O(d(\alpha_n))$ *instead of* $O(\log(c(\alpha_n)))$.

The alternative definition of uniform is in fact the one given by Borodin [Bor77].

Borodin expresses the general thesis in [Bor77] that circuit size corresponds to Turing machine time and circuit depth corresponds to Turing machine space. (If we identify uniform circuit depth with parallel time, then the second assertion is an instance of the parallel computation thesis stated in Section 1.) One precise statement of Borodin's thesis is the following:

Theorem 2.1 *If* $[\log T]$ *is fully space constructible, then*

$$\text{USIZE } (T^{O(1)}) = \text{DTIME } (T^{O(1)}).$$

If S *is fully space constructible, then*

$$\text{UDEPTH } (S^{O(1)}) = \text{DSPACE}(S^{O(1)}).$$

(See [HU79] for the definitions of *constructible*, *DTIME* and *DSPACE*. We have altered the definitions of the latter so they contain only subsets of $\{0,1\}^*$.)

The first equation is easy in this crude form, and in fact can be made considerably more precise (see [Pip79]).

The second equation is a consequence of the following result of Borodin [Bor77], which states that the inclusions (1.1) hold for uniform circuit depth.

Theorem 2.2 *If* S *is fully space constructable, then*

$$\begin{aligned} \text{UDEPTH } (S) &\subseteq \text{DSPACE } (S), \quad \textit{and} \\ \text{NSPACE} &\subseteq \text{UDEPTH } (S^2). \end{aligned}$$

Corollary **Savitch's Theorem**
If S *is fully space constructable, then* $\text{NSPACE}\,(S) \subseteq \text{DSPACE}\,(S^2)$.

Let us sketch the proof of theorem 2.2. The *circuit value problem* (see [HU79]) is the set of all binary strings encoding systems $\langle x_1, \ldots, x_n; \alpha \rangle$ where each $x_i \in \{0,1\}$ and α is a circuit whose unique output is 1 when its n inputs take on the values $x_1, \ldots, x_n$.

Lemma 2.1 *There is a deterministic Turing machine M which recognizes the circuit value problem, and on an input encoding $\langle x_1, \ldots, x_n; \alpha \rangle$, M uses space $O(d(\alpha))$.*

The idea is to perform a depth first search of α from the output node taking left descendants first. M stores the number of the node ν currently examined, together with one symbol for each node on the path followed from the root to ν. This symbol is either a marker L, if the search is proceeding on through the left input of the node; or the value of the left input if this value has been determined and the search is proceeding on to the right.

The first inclusion of Theorem 2.2 follows from Lemma 2.1 and Proposition 2.1.

To prove the second inclusion, recall that the graph reachability problem (GRP) (see [HU79]) is the set of all binary strings encoding the adjacency matrix of a digraph G on nodes $\{1, 2, \ldots, N\}$ such that G has a path from node 1 to node N.

Lemma 2.2 $\mathrm{GRP} \in \mathrm{UDEPTH}(\log^2 n)$.

The proof involves constructing a circuit which computes the transitive closure of a Boolean matrix by repeated squaring. The circuit has $O(\log n)$ stages, and each stage has depth $O(\log n)$ and computes the Boolean square of the matrix resulting from the previous stage. The circuit can be constructed by a deterministic Turing machine in space $O(\log n)$.

Given a nondeterministic S space bounded Turing machine M and the input length n, a circuit α_n is constructed which does the following on an input string w of length n. α_n first computes the adjacency matrix A of the graph whose nodes are the possible configurations of M with an input of length n, and whose edges represent possible steps in a computation with input w. We can assume M has an initial configuration labelled 1 and a unique accepting configuration labelled N. α_n now solves the graph reachability problem for A according to Lemma 2.2. The solution to the problem is positive iff M accepts w. Using Lemma 2.2 it is not hard to see that α_n has depth $O(S^2)$ and can be constructed in deterministic space $O(S^2)$ (in fact, space $O(S)$).

Theorem 2.1 represents one way to make precise Borodin's thesis that size corresponds to time and depth to space. Alternatively, instead of making the circuit family $\{\alpha_n\}$ uniform one can make Turing machines nonuniform (see [Sch76]). We borrow from Pippenger's terminology [Pip79]. Suppose $g : \{0,1\}^* \to \{0,1\}^*$. We say that a (deterministic or non-deterministic) multitape Turing machine *accepts A modulo g* provided that M accepts A under the condition that in addition to the normal input $x \in \{0,1\}^*$ on a read only input tape M is also provided with $g(x)$ on a separate read only tape called the *reference* tape. The space used by M is the work tape space plus $\lceil \log |g(x)| \rceil$, where $|w|$ is the length of w. (The term $\lceil \log |g(x)| \rceil$ was

not counted in [Pip79], but it should be, since it represents the amount of information stored by the position of the head on the reference tape.) The function g is *length determined* if $g(x)$ depends only on $|x|$, and not otherwise on x. A *nonuniform* machine is a machine M together with a length determined function g. It accepts A provided it accepts A modulo g. We add (NONUNIFORM) after a complexity class to indicate the machines are allowed to be nonuniform.

There is an alternative and more elegant definition of nonuniform space. We say that A is in DSPACE (S) (NONUNIFORM) provided there is a family $\{F_n\}$ of finite automata, each with a two-way read only input tape, such that F_n recognizes A^n, and $\log |F_n| = \mathrm{O}(S(n))$, where $|F_n|$ is the number of states of F_n. It is not hard to verify that this definition is equivalent to the one in the previous paragraph (recall our convention that $S(n) \geqslant \log n$).

The above definition does not work for time. However, Les Valiant pointed out that we could change the definition of nonuniform Turing machine to be a family $\{M_n\}$ of Turing machines instead of a single Turing machine with a reference tape. The time complexity $T(n)$ of such a family would be the maximum of $|M_n|$ and the worst case running time of M_n on inputs of length n. The space complexity $S(n)$ would be $\log |M_n|$ plus the worst case space used by M_n on inputs of length n. This gives the same definition of nonuniform space as before, but the nonuniform time is only the same up to application of a polynomial.

In any case, theorems 2.1 and 2.2 have the following analogs for nonuniform machines:

Theorem 2.3

$$\begin{aligned} \text{SIZE}\ (T^{\mathrm{O}(1)}) &= \text{DTIME}\ (T^{\mathrm{O}(1)})\ \text{(NONUNIFORM)}, \quad \textit{and} \\ \text{DEPTH}\ (S^{\mathrm{O}(1)}) &= \text{DSPACE}\ (S^{\mathrm{O}(1)})\ \text{(NONUNIFORM)}. \end{aligned}$$

Theorem 2.4

$$\begin{aligned} &\text{DEPTH}\ (S) \subseteq \text{DSPACE}\ (S)\ \ \text{(NONUNIFORM)}, \quad \textit{and} \\ &\text{NSPACE}\ (S)\ \ \text{(NONUNIFORM)} \subseteq \text{DEPTH}\ (S^2). \end{aligned}$$

To prove these results, the nonuniform machines simulate the circuits by letting $g(x)$ provide a description of the circuit for inputs of length $|x|$. Conversely, a circuit family $\{\alpha_n\}$ can simulate a nonuniform machine by building into α_n the value of $g(x)$ for $|x| = n$.

Note that the following nonuniform version of Savitch's theorem is a consequence of Theorem 2.4:

Corollary

$$\text{NSPACE}\ (S)\ \ (\text{NONUNIFORM}) \subseteq \text{DSPACE}\ (S^2)\ \ (\text{NONUNIFORM}).$$

In other words, a 2^s-state 2NFA can be simulated by a $2^{O(s^2)}$-state 2DFA for inputs of length $n \leqslant 2^s$.

A second interesting model of parallel computation, which falls in the fixed structure category, is the *alternating Turing machine* (ATM) ([CS76], [Koz76], [CKS78]). An ATM is a generalization of a nondeterministic multitape Turing machine. A nondeterministic machine has *existential* states, for which there are several possible next states, and at least one of the alternatives must lead eventually to an accepting state. In addition to existential states, an ATM also has *universal* states, for which *all* of the possible next states must lead to an accepting state. We define the accepting state to be a universal state with no successors. Every state is either universal or existential. Thus an *accepting computation* of an ATM M *with input* w is a finite tree whose nodes are labelled with configurations of M, such that i) every universal node (i.e. node whose configuration has a universal state) must have all possible next configurations as children, ii) every existential node must have at least one possible next configuration as a child, and iii) the root is the initial configuration. In order for M to operate in sublinear time we assume it has "random access" to the bits of w instead of a read only input tape. That is, M has a special index tape, and when M writes an index i on the index tape and assumes one of a distinguished set of index states, the i-th symbol of the input w is placed on the index tape. We say M *accepts* w in time s and space l if there is an accepting computation of M with input w whose longest path from root to leaf is s or less, and such that no configuration in the computation has tapes of length exceeding l. The complexity classes for time and space for ATM's are designated ATIME (S) and ASPACE (L), respectively, and we always assume $S(n), L(n) \geqslant \log\ n$.

As different as ATM's may seem from uniform circuit families, there is a remarkably close correspondence between alternating time and circuit depth, and between alternating space and circuit size. Unfortunately, our definition of *uniform* for circuits is too weak to express the correspondence precisely. Ruzzo gives a number of alternative definitions, of which the strongest is the following: $\{\alpha_n\}$ is U_E *uniform* iff the connection language L_{EC} can be recognized by a deterministic Turing machine in time $O(\log c(\alpha_n))$. Here L_{EC} consists of those quadruples (n, g, p, x) such that if g' is the gate reached by following the path $p \in \{\text{L}, \text{R}\}^*$ (where $|p| \leqslant \log c\,(\alpha_n)$) in circuit α_n back from gate g (L, R refer to left and right input, respectively) then g' has label x if $x \in B_2$, and $g' = x$ otherwise. (Assume n, g and x are expressed in binary notation.)

If we use the notation, for example, $U_E DEPTH(S)$ to indicate this notion of uniformity, then we have

$$\begin{aligned} ATIME(S) &= U_E DEPTH(S), \quad \text{and} \\ ASPACE(L) &= U_E SIZE(2^{O(L)}), \end{aligned}$$

assuming $S(n)$ can be computed in deterministic time $S(n)$, and $L(n)$ can be computed in deterministic time $L(n)$ given n in binary notation. In fact, Ruzzo [Ruz79a] proves the stronger result that the equivalences hold simultaneously. Let us use the notation ATIME-SPACE (S, L) for the class of sets accepted simultaneously in time S and space L on an ATM, (note that this may be a proper subset of the intersection of $ATIME(S)$ and $ASPACE(L)$), and analogous notation for other simultaneous classes. Then

Theorem 2.5 $ATIME\text{-}SPACE(S, L) = U_E DEPTH\text{-}SIZE(S, 2^{O(L)})$, *provided S and L are computable in deterministic time* $O(S)$.

Ruzzo shows the above result still holds when U_E is replaced by U, provided $S \geqslant L^2$.

From their definition, ATM's appear to model a restricted form of parallel computation, because the "processors" in the model are restricted to be Turing machines, and they must be organized in the form of an and-or-tree. This makes Theorem 2.5 all the more interesting. On the other hand, ATM's are more pleasing in one way than circuit families, because there is no question of how to define *uniform*. Each ATM is automatically uniform. In fact, ATM's may be the best candidate proposed so far for defining parallel time, at least in the fixed structure category. But this remains to be seen. The one clear drawback of ATM's is that they do not seem to have any resource that corresponds to hardware size (see Section 4).

3 Log Depth vs Log Space

As far as we know, the second inclusions in Theorems 2.2 and 2.4 cannot be improved, even when NSPACE is replaced by DSPACE. (Of course an improvement for NSPACE would improve Savitch's theorem.) Taking $S(n) = \log n$ as the most basic case, it is interesting to look for examples of sets in $DSPACE(\log n)$ which do not appear to be in $DEPTH(\log n)$. Addition of n n-digit binary numbers, and multiplication of two n-digit binary numbers both can be done in $O(\log n)$ circuit depth (see [Sav76]), as can sorting n n-digit binary numbers (see [MP75]). On the other hand, the "cycle free problem" is in $DSPACE(\log n)$ but does not appear to be in $DEPTH(\log n)$.

Definition The cycle free problem (CFP) is the set of all binary codes for symmetric Boolean $N \times N$ adjacency matrices A of undirected cycle-free graphs.

One can define functions $f : \{0,1\}^* \to \{0,1\}^*$ computable in depth S (or uniform depth S) using circuits with several outputs. We say A_1 is *log depth reducible* to A_2 (respectively *uniformly log depth reducible*) iff there is some function f computable in depth $O(\log n)$ (respectively uniform depth $O(\log n)$) such that $w \in A_1$ iff $f(w) \in A_2$, for all w. We say A is *log depth complete* for the class $\mathscr{S}$ iff $A \in \mathscr{S}$, and every $A' \in \mathscr{S}$ is log depth reducible to A. The uniform case is defined similarly. The main ideas in the proof of the following result appear in Hong [Hon80].

Theorem 3.1
(a) CFP *is uniformly log depth complete for* DSPACE $(\log n)$.
(b) CFP *is log depth complete for* DSPACE $(\log n)$ (NONUNIFORM).

Corollary
(a) DSPACE $(\log n)$ = UDEPTH $(\log n)$ iff CFP $\in$ UDEPTH$(\log n)$.
(b) DSPACE $(\log n)$ (NONUNIFORM) = DEPTH $(\log n)$ *iff* CFP $\in$ DEPTH $(\log n)$.

We note that because UDEPTH $(\log n) \subseteq$ DEPTH $(\log n)$, the first equation in the Corollary implies the second. This fact does not seem to be obvious without using the CFP.

To prove CFP $\in$ DSPACE $(\log n)$, Hong devised an algorithm for moving several pebbles around the input graph in an attempt to do a depth first search of each of its components. To prove that every

$$A \in \text{DSPACE } (\log n)$$

is uniformly log depth reducible to CFP, one can, given an input w, define a graph whose nodes are $\mathscr{C} \times \{0, 1, \ldots, T\}$, where $\mathscr{C}$ is the set of possible configurations of the Turing machine M with input w, where M accepts the complement of A in space $O(\log n)$, and T is an upper bound on the computation time. Two nodes (c,t) and $c', t')$ are adjacent iff either $c \to c'$ in one step and $t' = t+1$, or $c' \to c$ and $t = t'+1$. If we let c_0 be the initial configuration and c_f be the unique accepting configuration, then we also add an edge between $(c_0, 0)$ and $\big(c_f, T\big)$. Using the fact that M is deterministic, it is not hard to see that M accepts w iff the graph has a cycle.

A second example for which theorem 3.1 applies is GAP1: the graph reachability problem for directed graphs of outdegree one. The completeness of GAP1 for DSPACE $(\log n)$ is proved for reducibilities other than log depth in [Jon75] and in [HIM78]. The proof of theorem 3.1 for GAP1 is easier than for CFP.

The following example is interesting, because it is complete for nonuniform $\log n$ space, but no one knows how to solve it in uniform $\log n$ space.

Definition The undirected graph reachability problem (URP) is the set of codes of symmetric adjacency matrices of graphs with nodes $\{1, 2, \ldots, N\}$ with a path from node 1 to node N.

Theorem 3.2 URP *is log depth complete for* DSPACE $(\log n)$ (NONUNIFORM).

That URP $\in$ DSPACE $(\log n)$ (NONUNIFORM) follows from the existence of a short universal covering string for all n-node undirected connected oriented graphs of fixed degree (see [AKL+79]). The reducibility proof is similar to the above argument.

Many interesting problems have $O(\log^2 n)$ as the best known upper bound for both deterministic space and uniform depth. It is interesting to try to reduce these to each other via log depth or uniform log depth reducibility, so as to cut down the number of equivalence classes of problems classified by their depth complexity. For example, the directed graph reachability problem (GRP) is well known to be log space complete for NSPACE $(\log n)$ (see [HU79]). In fact, it is also *uniform log depth* complete for NSPACE $(\log n)$. Two other examples are finding the integer part of the quotient of two n-digit binary numbers, and raising an n-digit number to the power n. The best known upper bound for both problems for both space and depth is $O(\log^2 n)$. Hoover [Hoo79] shows that each is log depth reducible to the other, although one of the reductions is not uniform. As a matter of interest, Hoover also points out that the base conversion problem (say converting binary notation to ternary) is in nonuniform depth $O(\log n)$ (because the powers of two in ternary can be built in), but the best space upper bound and uniform depth upper bound is $O(\log^2 n)$.

4 Conglomerates and Aggregates

Uniform circuits and ATM's are good models for measuring parallel time, but neither is right for measuring the second important resource mentioned in the introduction, namely hardware size. What is needed is to allow circuits to have cycles. Goldschlager's conglomerates [Gol78] satisfy this requirement. Briefly, a *conglomerate* is an infinite collection $\{M_0, M_1, \ldots, \}$ of identical deterministic finite state machines connected together in some manner. Each machine has $r \geqslant 1$ inputs and one output, and the connection function f specifies for some inputs of some machines the output of which machine it is connected to. (Inputs left unconnected receive some fixed symbol b.) Cycles are allowed in the connection graph. Initially at time 0, the first n machines $M_1, \ldots, M_n$ store the symbols of the input string $w_1\, w_2\, \ldots\, w_n$, and all other machines start in the initial state q_0. At subsequent times $1, 2, \ldots$ each machine assumes a new state and transmits output symbols in a manner determined by its input symbols and state at the previous step.

The conglomerate accepts its input if at any time machine M_0 enters the special state q'.

The uniformity condition for conglomerates specifies that the connection function f can be computed within some space bound P on a Turing machine, where $f(i_1, i_2 \ldots i_k) = s$, if machine M_s is reached by starting with M_0 and tracing back via input i_1 then input i_2 of that machine, and so on. The linear space bound $P(n) = n$ suffices in order for the inclusions (1.1) to hold when parallel time is taken to mean conglomerate time. We have not considered the question of which uniformity condition makes conglomerate time equivalent to uniform circuit depth.

Goldschlager did not define or study the "hardware size" of a conglomerate computation. Rather than do that now, we present a new model (developed in Dymond [Dym80]) to study, called an *aggregate* which can be viewed either as like a finite conglomerate, or like a circuit with feedback. Similar objects have been called "sequential circuits" or "logical nets" in the switching theory literature. An aggregate has different input/output conventions than these, and we assume every gate has unit delay to avoid any possibility of ambiguous computations. We interpret the result as more a "parallel circuit" than a "sequential circuit".

More formally, an *aggregate* β_n on inputs $x_1, \ldots, x_n$ is a directed graph (not necessarily acyclic) whose nodes have labels from $B_0 \cup B_1 \cup B_2 \cup \{x\}$. A node ν with label $g \in B_i$ must have indegree i, and one edge into ν is associated with each argument of g. If a node ν has label x, then ν is an *input* node and must have indegree zero. Associated with each input node ν is a *register* R_ν consisting of $\lfloor \log n \rfloor$ nodes, which specifies which input x_i is presented to x. There is a distinguished pair of nodes designated ν_0 and ν_1, called *output* nodes. A *configuration* of β_n is an assignment of 0 or 1 to each node ν of β_n called the *output* of ν. A *computation* of β_n is a sequence $C_0, C_1, \ldots$ of configurations as follows.

(a) All nodes in C_0 have output 0 except any node labelled with the constant function $1 \in B_0$.

(b) If ν has label $g \in B_i$, then in C_{t+1} ν has output equal to g applied to the input (s) of ν in C_t.

(c) If ν is an input node, then ν has output 0 in C_t for $t < \lceil \log n \rceil$, and in general in $C_{t+\lceil \log n \rceil}\nu$ has output x_{i+1}, where i is the value in binary notation of the register R_ν in C_t.

The output of β_n is defined to be the output of the node ν_0 in the first configuration C_t in which ν_1 has output 1. The running time $t(\beta_n)$ of β_n is the maximum over all inputs $x_1, \ldots, x_n$ of this index t. The *hardware size* $h(\beta_n)$ is the number of nodes in β_n.

The peculiar input conventions for aggregates need justification. The reason that inputs x_i are not fed directly into aggregates as they are for circuits is that this would entail $h(\beta_n) \geqslant n$, whereas we are interested in sublinear hardware bounds (see theorem 4.1 below). In fact, the value of an input node ν could be computed from the index stored in R_ν using a decoding circuit of size $O(n)$ and depth $O(\log n)$ (this is the reason we assume a delay of $\lceil \log n \rceil$ for R_ν to affect ν). Our convention of not counting the size of the decoding circuit is similar to the convention of not counting the input tape in measuring the space used by an off line Turing machine. (One might imagine, for example, a large number of small aggregates sharing the same large decoding circuits.)

Our input and output conventions could be modified slightly to allow aggregates to compute functions instead of to recognize sets. The particular bit computed of the function would be specified by a part of the input called the output specifier. Then aggregates could be cascaded to compute the composition of two functions in hardware size equal to the sum of the hardware sizes for each of the functions. The output ν_0 of the first aggregate β would be connected to the input ν' of β', and the register $R_{\nu'}$ of β' would be connected to the output specifier of β. The timing conventions for the input ν' of β' would be changed to allow for the uncertain delay between an input request and its answer (signalled by ν_1).

Definition A family $\{\beta_n\}$ of aggregates is *uniform* provided the transformation $1^n \to \beta_n$ can be computed in deterministic space

$$O(\log h(\bar{\beta}_n) + \log n).$$

The complexity classes defined by uniform aggregate families of bounded hardware and bounded time and of nonuniform bounded time families can be characterized as follows:

Theorem 4.1 *Let H, S be fully space constructible functions with $H(n), S(n) \geqslant \log n$. Then*

(a) UHARDWARE (H) $=$ DSPACE (H),
(b) HARDWARE (H) $\subseteq$ DSPACE (H) (NONUNIFORM),
(c) UAGTIME (S) $=$ UDEPTH (S),
(d) AGTIME (S) $=$ DEPTH (S).

This theorem shows that neither of the resources uniform hardware and uniform aggregate time define new complexity classes in themselves. However, taken together they define apparently new and natural simultaneous complexity classes. Simultaneous resource bounds are discussed in Section 7.

Proof sketch (a) and (b). A deterministic Turing machine can simulate an aggregate by updating a bit vector which has one bit for the output of each gate. A queue is

kept of the next $\lceil \log n \rceil$ input values for each input node ν to facilitate the update of these nodes. Note that there can be at most $O(H(n)/\log n)$ input nodes ν, since each has an associated register R_ν with $\lceil \log n \rceil$ gates.

An aggregate can simulate a (uniform) deterministic Turing machine for inputs of length n by having a "box" of gates devoted to each work tape square. The box records the current contents of that tape square, and if scanned, it records the state of the Turing machine. The contents of the input tape is obtained by an input node ν, whose register R_ν is attached to a counter which records the input head position. This simulation does not work for nonuniform Turing machines, since the reference tape could have length exponential in the size of the aggregate.

Proof sketch (c) and (d). An aggregate can be converted to a circuit by implementing each input node by the decoding circuit mentioned earlier. Then each gate ν is replaced by a set $\{\langle \nu, t\rangle \mid 0 \leqslant t \leqslant S(n)\}$ of gates. The gate $\langle \nu, t+1\rangle$ has inputs from $\langle w_1, t\rangle$ and $\langle w_2, t\rangle$, where w_1 and w_2 are the inputs to ν in the aggregate. The circuit output is $\langle \nu_0, S(n)\rangle$ (we can assume ν_0 retains its value in the aggregate once $\nu_1 = 1$).

To convert a circuit to an aggregate, construct an input node ν_i for each circuit input x_i. The register R_{ν_i} has constant value i. Let ν_0 be the output node for the circuit, and let ν_1 be the end of a length $S(n)$ chain of identity gates having the constant function 1 at the beginning.

More details can be found in [Dym80].

The above theorem sheds some light on the old problem of to what extent feedback in circuits helps reduce the number of required gates. The best result in that direction seems to be due to Rivest [Riv77] who gives examples showing a linear reduction in size, but only for a multiple output circuit. On the other hand, theorem 4 suggests that disallowing feedback might cause an exponential size blow up in some cases. For example, let A be a set which is log space linear time complete for DSPACE(n) (see Hong [Hon80] for a natural example). Then by equation (a), A can be recognized by an aggregate family of linear hardware size. On the other hand, as far as we know A requires exponential time on a Turing machine, so by theorem 2.1 it would follow that any *uniform* circuit family recognizing A has size at least 2^{n^ε} for some $\varepsilon > 0$ (indeed $2^{\Omega(n/\log^2 n)}$ by [Pip77]). In fact, we know of no way to reduce this bound even if we allow nonuniform circuit families.

Of course in other cases a proof that disallowing feedback causes exponential size blow up would imply P $\neq$ NP. For example, SATISFIABILITY can be recognized in linear space and hence is recognized by an aggregate family with linear hardware. If P $=$ NP, then SATISFIABILITY would be recognizable by polynomial size circuits.

We close this section with two little results about aggregates in the style "if horses can whistle then pigs can fly". This style (but not these results) comes from the paper of Karp and Lipton [KL80]. The results are intriguing because the hypotheses consist of assumptions concerning the nonuniform complexity of classes and the conclusions assert uniform complexity bounds.

Theorem 4.2 *If* $P \subseteq \text{HARDWARE}(\log n)$ *then* $P \subseteq \text{DSPACE}(\log n \log\log n)$

Proof sketch. Since the circuit value problem (CVP) is log space complete for P (see [HU79]), it suffices to prove CVP is in the second class given it is in the first class. Thus for each n we assume the existence of an aggregate β_n which correctly solves the CVP on inputs of length n, with $h(\beta_n) = O(\log n)$. A deterministic Turing machine M can represent and simulate a candidate β'_n for β_n in space $O(\log n \log\log n)$, and in fact M can cycle through all such candidates β'_n. There is no apparent way to determine in small space whether β'_n gives the correct answer for all inputs c of length n, but given a particular input c (i.e. circuit with inputs specified) M can check that β'_n gives consistent answers for each gate g of c by simulating β'_n three times, with c as input modified so that its output is each of the two inputs to g and g itself. If β'_n gives consistent answers for each gate of c, then β'_n correctly gives the output of c (i.e. tells whether $c \in$ CVP).

Theorem 4.3 *If* $\text{NP} \subseteq \text{HARDWARE}(\log n)$ *then* $\text{NP} \subseteq \text{DSPACE}(\log_n \log\log n)$.

Proof sketch. It suffices to show that SATISFIABILITY is in the second class given it is in the first class. Reasoning as above, the Turing machine M can check whether a candidate aggregate β'_n correctly tells whether a propositional formula F is satisfiable by making β'_n produce a satisfying assignment bit by bit, by plugging in partial truth assignments to F and asking β'_n about the result. The trouble is M cannot remember the partial assignments in small space. However, the problem of whether "the i-th bit is 1 in the lexocographically first assignment which β'_n says satisfies F" is in P. Thus by theorem 4.2 this bit can be determined in space $O(\log n \log\log n)$, and M can determine whether this assignment satisfies F in small space.

5 Hardware Modification Machines

As mentioned in the introduction, there is a need to define a parallel model which is more powerful than an aggregate, in that it can modify its circuits, but less powerful than existing parallel RAM models, in that each unit of hardware can only

perform a bounded amount of work in one step. We shall call the new machine a *hardware modification machine* (HMM), since it is intended to be the parallel analog of the storage modification machine. An HMM consists of a finite collection of finite state machines connected together as in a conglomerate. At each step, each machine may, in addition to assuming a new state and transmitting output signals, modify its input connections. Specifically, it may detach any of its inputs and re-attach it to a new machine which it brings into the HMM, or it may re-attach it to an output of any machine which can be reached by a path of length at most two traced backwards from the input.

One advantage of an HMM over circuits, aggregates, and conglomerates is that there is no question of uniformity. The machine is uniform because it constructs itself.

An HMM can execute an algorithm like the one described in [FW78] to simulate a deterministic S space bounded machine in time $O(S)$, and HMM time S can be simulated in deterministic space $O(S^2)$. Thus the inclusions (1.2) apply.

The theory of HMM's is developed in [Dym80].

6 Other Modifiable Models

The first published parallel model introduced and compared in power to space bounded machines was the vector machine of Pratt and Stockmeyer [PS78]. A vector machine is like a random access machine, except there are two distinct kinds of registers: index registers and vector registers. Addition, subtraction, and comparison operations can be applied to both kinds of registers, and both can be accessed via index registers. In addition, bitwise Boolean operations can be applied to vector registers, and vector registers can be shifted by an amount specified by an index register. These shift operations allow the vectors to grow in length exponentially in the computation time, and hence the bitwise vector operations represent a high degree of parallelism.

Pratt and Stockmeyer prove that vector machine time $(S) \subseteq \text{DSPACE}(S^2)$ and $\text{NSPACE}(S) \subseteq$ vector machine time (S^2), for suitable $S(n) \geqslant \log n$. These inclusions are weaker than either 1.1 or 1.2. It seems that vector machines have some very powerful operations, such as the shift, which preclude linear space simulation of time. On the other hand, they are apparently not powerful enough to allow a linear time simulation of space.

This aesthetic defect is balanced by other considerations. The model is a pleasant one, and is an extension of actual computer designs. Enough examples of vector machine algorithms are given in [PS78] to indicate the machine's suitability for the programming of parallel algorithms. Simon [Sim77] proved the surprising result that the power of vector machines is only increased by application of

a polynomial when no distinction is made between index registers and vector registers, so that a vector register can be shifted by an amount specified by another vector register.

The MRAM's and CRAM's of Hartmanis and Simon [HS74] are similar to vector machines, except they have only one type of register, and perform multiplication (or concatenation) instead of shifting. The time space simulation results are similar to those for vector machines.

A number of other variations of parallel random access machines have been introduced. One example is Goldschlager's SIMDAG [Gol78], which stands for single instruction stream, multiple data stream, global memory. This consists of a control processor (CPU) and an infinite sequence $\text{PPU}_0, \text{PPU}_1, \ldots$ of parallel processors, each connected to an infinite random access global memory. In addition, each parallel processor has a local infinite random access memory. The program is executed by the CPU, which can broadcast instructions to the active PPU's. Each instruction broadcast is executed by the first k PPU_i's, where k is stored in some location of global memory. Each PPU_i executes the same instruction, but the memory locations accessed can be indexed by the subscript i, and so can be different for different PPU_i's. The simulations proved for SIMDAG's are a little stronger than 1.2; namely, $\text{SIMDAGTIME}(S) \subseteq \text{DSPACE}(S^2)$, and $\text{NSPACE}(S) \subseteq \text{SIMDAGTIME}(S)$. The reason that nondeterministic space S instead of just deterministic space S can be simulated in time $O(S)$ is apparently because of a powerful SIMDAG instruction which allows any number of PPU's to store into memory at once. If two or more try to store into the same location, the lowest numbered processor succeeds. This gives the effect of a huge fan-in being executed in one step.

The P-RAM of Fortune and Wyllie [FW78] is similar to the SIMDAG, except different parallel processors can be executing different parts of their program at once, so it is "multiple instruction stream". Also, there is no instruction comparable to the SIMDAG's instruction which allows a potentially unbounded number of processors to try to store in a given location at once. Wyllie shows in [Wyl79] that the multiple instruction stream gives only a constant factor time advantage over SIMDAG's. On the other hand, the unbounded fan-in for SIMDAG's seems to be a real advantage, since the time space simulation results for P-RAM's are those of 1.2; weaker than for SIMDAG's.

The PRAM's of [SS79] have no global memory, but a given processor can initiate offspring processors. The time space simulation results in [SS79] are weaker than either 1.1 or 1.2.

In conclusion, all the parallel models in this section have powerful instructions which cannot be considered primitive.

7 Simultaneous Resource Bounds

In Section 2 we indicated that sequential time is roughly equivalent to uniform circuit size, and sequential space is roughly equivalent to uniform circuit depth. A natural question to ask is whether simultaneous time and space bounds are roughly equivalent to simultaneous uniform size and depth bounds. To be more specific, the well known class P can be characterized as either DTIME $(n^{O(1)})$ or as USIZE $(n^{O(1)})$, and "polylog space" is both DSPACE $\left((\log n)^{O(1)}\right)$ and UDEPTH $\left((\log n)^{O(1)}\right)$. If we use the notation DTIME-SPACE (T, S) to refer to the class of sets accepted by some deterministic Turing machine which runs both in time T and space S, then the class referred to as PLOPS (polynomial time and polylog space) in [Coo79] (and now called SC by agreement among several authors), can be written DTIME-SPACE $\left(n^{O(1)}, (\log n)^{O(1)}\right)$. Note that SC is presumably a proper subset of $P \cap$ DSPACE $\left((\log n)^{O(1)}\right)$. For example, the graph reachability problem (GRP) (see Section 2) is in the intersection class but not known to be in SC.

The corresponding circuit-defined class is USIZE-DEPTH $\left(n^{O(1)}, (\log n)^{O(1)}\right)$, which is called NC in [Coo79] and [Ruz79a] after Pippenger, who first characterized it (see theorem 7.1). Again NC is presumably a proper subset of the intersection class, although it is remarkably difficult to think of a natural example of something in the intersection class but not in NC. (The universal set UPL defined below may be an artificial example.)

Getting back to the original question, we now ask whether SC = NC? The answer is apparently no, because there are natural problems in NC which do not appear to be in SC. One example is GRP (or any other complete problem for NSPACE $(\log n)$). Other examples are integer division and integer powering (see Section 3). (Technically these should be made into recognition problems by specifying an index i as part of the input and asking whether the i-th output digit is 1.) And another class of examples are those context free languages which we don't know how to put into SC (see theorem 7.5 below).

Conversely, it is not so easy to find natural candidates for the difference set SC-NC. In fact, it is difficult to find sets in SC which are not clearly in DSPACE $(\log n)$ (and therefore in NC). Any universal deterministic context free language (DCFL) provides an example because of the result in [Coo79], but again Ruzzo proved that all CFL's are in NC.

One can still concoct artificial candidates for SC-NC. For example, a universal set UPL for SC^2 $\left(SC^2 = \text{DTIME-SPACE}\left(n^{O(1)}, \log^2 n\right)\right)$ can be constructed as follows: Design a machine M which shuts itself off if it attempts to use more than $\log^2 n$ space or n^2 time. Let M on an input coding a pair (x, y) simulate machine number x

on input y; and accept iff neither its pace nor time bound is exceeded and machine x accepts y. Then UPL is log space complete for SC^2 and does not appear to be in NC.

We conclude that time and space together do not seem to be even roughly equivalent to uniform size and depth together. However, Pippenger [Pip79] proves that time and reversal together do correspond to size and depth together. Here the *reversal* of a computation of a multitape Turing machine is the number of times any of its heads changes direction (a hesitation is not a reversal). Pippenger proves

Theorem 7.1 $\mathrm{NC} = \mathrm{DTIME\text{-}REVERSAL}\left(n^{O(1)}, (\log n)^{O(1)}\right)$.

This is one characterization of NC, and Ruzzo [Ruz79a] points out several others. In fact, NC appears to be a very stable and interesting class. Intuitively, it is comprised of all problems which can be solved very rapidly on a parallel computer of feasible size. To make this statement more evident, we point out NC can also be characterized in terms of aggregates (see Section 4).

Theorem 7.2 $\mathrm{NC} = \mathrm{UHARDWARE\text{-}AGTIME}\left(n^{O(1)}, (\log n)^{O(1)}\right)$.

This follows immediately from the definition of NC and the discussion in Section 4 about converting circuits to aggregates and vice versa.

I would like to mention three of Ruzzo's [Ruz79a] characterizations of NC. First, Ruzzo gives several alternative definitions of *uniform circuit family*, including our definition in Section 2, and proves that NC remains the same for all of them. In particular, NC remains unchanged when the strong definition of U_E uniform is chosen. From this and theorem 2.5 Ruzzo concludes the second characterization:

Theorem 7.3 $\mathrm{NC} = \mathrm{ATIME\text{-}SPACE}\left((\log n)^{O(1)}, \log n\right)$.

The third characterization is

Theorem 7.4 $\mathrm{NC} = \mathrm{AuxPDA\ TIME\text{-}SPACE}\left(2^{\log n^{O(1)}}, \log n\right)$.

Here AuxPDA stands for auxiliary pushdown automaton. The theorem holds whether it is deterministic or nondeterministic.

We now sketch the proof of another interesting Ruzzo result:

Theorem 7.5 *Every context free language is in* NC.

Part of the interest of the proof is that it was apparently discovered using ATM's (via theorem 7.3), which is an indication that ATM's are a useful tool for discovering and expressing parallel algorithms. The proof of 7.5 follows the classical [LSH65] proof that every CFL is in DSPACE $(\log^2 n)$, but needs a new idea. As in [LSH65], we assume the grammar is in Chomsky form, and try to verify the existence of a parse tree for the input string whose nodes have the form (σ, i, j) (which is *valid* if symbol

σ generates the segment of the input between the i-th and j-th symbols inclusive). The ATM algorithm proceeds by guessing (via an existential state) a node (σ, i, j) which generates between one-third and two-thirds of the input string and then verifies (via a universal state) that both (1) (σ, i, j) is a valid node, and (2) the original root is valid given (σ, i, j) is valid. These two subproblems are solved by executing the algorithm recursively. Since the depth of the recursion is $O(\log n)$, the alternating time is $O(\log^2 n)$, but unfortunately a general recursive call to the algorithm must remember up to $\log n$ hypothesis nodes $(\sigma_1, i_1, j_1), \ldots, (\sigma_k, i_k, j_k)$, which require a total of $\Omega(\log^2 n)$ space to express, so theorem 7.3 does not apply. The new idea is to keep the number of hypothesis nodes down to two, by guessing at a common ancestor to two of them whenever three hypotheses would otherwise be formed. This keeps the space down to $O(\log n)$, so 7.3 applies.

In addition to comparing time and reversal to size and depth, Pippenger also shows time and space together are roughly equivalent to size and width. To define the last resource, let us say a circuit is *synchronous* if its gates can be divided into levels such that all inputs to the gates at level l are either input nodes x_i or are from gates at level $l-1$. Then the *width* of a synchronous circuit is the maximum of the number of gates at any level. Pippenger also gives a suitable definition of width for nonsynchronous circuits and proves several relations among width, size, space and time, of which the following is a corollary:

Theorem 7.6 $\mathrm{SC} = \text{USIZE-WIDTH}\ \left(n^{O(1)}, (\log n)^{O(1)}\right)$.

Dymond [Dym80] extends Pippenger's results to relate space and reversals to uniform width and depth. Two easy observations along these lines are that theorem 7.6 still holds if USIZE is replaced by UDEPTH, and SC remains unchanged if time is replaced by reversal in its definition. In addition, it is not hard to see that SC can be characterized in terms of aggregates as follows:

Theorem 7.7 $\mathrm{SC} = \text{UHARDWARE-AGTIME}\ \left((\log n)^{O(1)}, n^{O(1)}\right)$.

This result shows an interesting duality with theorem 7.2. The question of whether NC = SC becomes the question of whether hardware size can be traded for computation time in uniform aggregates, without exponential blow up in the other resource.

8 Open Questions

Among the basic open questions in computational complexity are the problems of finding lower bounds for various resources for any simple interesting problem. In particular, for sequential complexity, we don't have any nonlinear time lower

bounds nor any nonlogarithmic (i.e. $\omega(\log n)$) space lower bounds on any natural problem in the class NP. For parallel complexity, the same state of ignorance applies to nonlinear circuit size, and nonlogarithmic depth and hardware. Theorems 2.2 and 2.4 indicate that a nonlogarithmic lower bound on circuit depth may be weaker (and therefore easier to obtain) than such a bound on space, so the depth question deserves more attention. (There are already results which show the depth complexity of some simple problems cannot be $\log_2 n + O(1)$: see Neciporuk [Neč66] and Hodes and Specker [HS74].)

For simultaneous resource bounds, the situation is almost as wide open, although Borodin and Cook [BC80] have recently shown that sorting cannot be done *simultaneously* in linear time and logarithmic space. It would be interesting to get similar tradeoff results for other resource pairs, such as size versus depth and aggregate time versus hardware. Another problem is whether there exists any set whose minimum time complexity is at least, say, the square of its minimum space complexity (assuming the latter is at least $\Omega(n)$). Similarly for uniform size versus uniform depth and aggregate time versus hardware size. (We do know by Lupanov's result [Sav76] that most sets have (nonuniform) size exponential in depth.

Finally, the questions concerning SC and NC mentioned in Section 7 are worth emphasizing. In particular, it would be nice to know whether one class is included in the other, and whether they are proper subsets of their (common) intersection class.

Acknowledgement

While forming my ideas on parallel computation, and in particular while writing this manuscript I had frequent conversations with my colleagues Allan Borodin and Patrick Dymond, and their ideas as much as mine have contributed to whatever insights appear here. My thanks also to Nicholas Pippenger for some very helpful conversations.

References

[AKL+79] R. Aleliunas, R. M. Karp, R. J. Lipton, L. Lovász, and C. Rackoff. October. 1979. Random walks, universal traversal sequences, and complexity of maze problems. In *Proceedings of the 20th Annual Symposium of Foundations of Computer Science*, 218–223. DOI: https://doi.org/10.1109/SFCS.1979.34.

[Bor77] A. Borodin. 1977. On relating time and space to size and depth. *SIAM J. Comput.* 6, 4, 733–744. DOI: https://doi.org/10.1137/0206054.

[BC80] A. Borodin and S. A. Cook. April 28–30. 1980. A time–space tradeoff for sorting on a general sequential model of computation. In *Proceedings of the 12th Annual*

ACM Symposium on Theory of Computing. Los Angeles, California. ACM, 294–301. DOI: https://doi.org/10.1145/800141.804677.

[BW64] A. W. Burks and J. B. Wright. 1964. Theory of logical nets. In E. F. Moore (Ed.), *Sequential Machines: Selected Papers*. Addison-Wesley, 193–212.

[CS76] A. K. Chandra and L. J. Stockmeyer. October 25–27. 1976. Alternation. In *17th Annual Symposium on Foundations of Computer Science*. IEEE, Houston, Texas, 98–108. DOI: https://doi.org/10.1109/SFCS.1976.4.

[CKS78] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. 1978. Alternation. IBM Research Report RC 7489.

[Coo79] S. A. Cook. April 30–May 2. 1979. Deterministic CFL's are accepted simultaneously in polynomial time and log squared space. In *Proceedings of the 11th Annual ACM Symposium on Theory of Computing*. Atlanta, Georgia. ACM, 338–345. DOI: https://doi.org/10.1145/800135.804426.

[Dym80] P. Dymond. 1980. *Simultaneous Resource Bounds and Parallel Computation.* Ph.D. Dissertation. University of Toronto, Dept. of Computer Science.

[FW78] S. Fortune and J. Wyllie. May 1–3. 1978. Parallelism in random access machines. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing*. ACM, San Diego, California, 114–118. DOI: https://doi.org/10.1145/800133.804339.

[Gol77] L. M. Goldschlager. 1977. *Synchronous Parallel Computation*. Ph.D. Dissertation. University of Toronto. Also, Technical Report 114, Department of Computer Science, University of Toronto.

[Gol78] L. M. Goldschlager. May 1–3. 1978. A unified approach to models of synchronous parallel machines. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing*. ACM, San Diego, California, 89–94. DOI: https://doi.org/10.1145/800133.804336.

[HS74] J. Hartmanis and J. Simon. October 14–16. 1974. On the power of multiplication in random access machines. In *15th Annual Symposium on Switching and Automata Theory*. IEEE, New Orleans, Louisiana, 13–23. DOI: https://doi.org/10.1109/SWAT.1974.20.

[HIM78] J. Hartmanis, N. Immerman, and S. Mahaney. October. 1978. One-way log tape reductions. In *19th Annual Symposium on Foundations of Computer Science*, 65–71.

[HS68] L. Hodes and E. Specker. 1968. Lengths of formulas and elimination of quantifiers I. In K. Schutte (Ed.), *Contributions to Mathematical Logic*, North Holland Publ. Co. 175–188. DOI: https://doi.org/10.1016/S0049-237X(08)70524-X.

[Hon80] J. W. Hong. April. 1980. On some space complexity problems about the set of assignments satisfying a boolean formula. In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing*. 310–317.

[Hoo79] H. J. Hoover. December. 1979. *Some Topics in Circuit Complexity*. M.Sc. thesis and Technical Report TR-138/79. Department of Computer Science, University of Toronto.

[HU79] J. E. Hopcroft and J. D. Ullman. 1979. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley. ISBN 0-201-02988-X.

[Jon75] N. D. Jones. 1975. Space-bounded reducibility among combinatorial problems. *J. Comput. Syst. Sci.* 11, 68–85. DOI: https://doi.org/10.1016/S0022-0000(75)80050-X.

[KL80] R. M. Karp and R. J. Lipton. April. 1980. Some connections between nonuniform and uniform complexity classes. In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing.* 302–309. Also presented at the *Specker Symposium on Algorithms and Complexity*. Zurich, February 1980.

[Koz76] D. Kozen. October 25–27. 1976. On parallelism in Turing machines. In *17th Annual Symposium on Foundations of Computer Science*. IEEE, Houston, Texas, 89–97. DOI: https://doi.org/10.1109/SFCS.1976.20.

[LSH65] P. M. Lewis II, R. E. Stearns, and J. Hartmanis. October 6–8. 1965. Memory bounds for recognition of context-free and context-sensitive languages. In *6th Annual Symposium on Switching Circuit Theory and Logical Design*. IEEE, Ann Arbor, Michigan, 191–202. DOI: https://doi.org/10.1109/FOCS.1965.14.

[MP75] D. E. Muller and F. P. Preperata. April. 1975. Bounds to complexities of networks for sorting and for switching. *J. ACM* 22, 2, 195–201. DOI: https://doi.org/10.1145/321879.321882.

[Neč66] È. Nečiporuk. 1966. On a boolean function. *Doklady of the Academy of the USSR* 169, 4, 765–766. (English translation in *Soviet Mathematics Doklady 7*, 4, 999–1000.)

[Pat76] M. S. Paterson. 1976. An introduction to boolean function complexity. *Astérisque*, *Société Mathématique de France* 38–39, 183–201.

[Pip77] N. Pippenger. 1977. Fast simulation of combinational logic networks by machines without random access storage. In *15th Allerton Conference on Communication Control and Computing*, 25–33.

[Pip79] N. Pippenger. October 29–31. 1979. On simultaneous resource bounds (preliminary version). In *20th Annual Symposium on Foundations of Computer Science*. IEEE, San Juan, Puerto Rico, 307–311. DOI: https://doi.org/10.1109/SFCS.1979.29.

[PS78] V. Pratt and L. Stockmeyer. 1978. A characterization of the power of vector machines. *J. Comput. Syst. Sci.* 12, 198–221.

[Riv77] R. L. Rivest. June. 1977. The necessity of feedback in minimal monotone combinational circuits. *IEEE Trans. Comput*. 26, 6, 606–607. DOI: https://doi.org/10.1109/TC.1977.1674886.

[Ruz79] W. L. Ruzzo. October 29–31. 1979a. On uniform circuit complexity (extended abstract). In *20th Annual Symposium on Foundations of Computer Science*. IEEE, San Juan, Puerto Rico, 312–318. DOI: https://doi.org/10.1109/SFCS.1979.31.

[Sav76] J. E. Savage. 1976. *The Complexity of Computing.* Wiley, New York.

[SS79] W. Savitch and M. Stimson. January. 1979. Time bounded random access machines with parallel processing. *J. Assoc. Comput. Mach.* 26, 103–118.

[Sch76] C. P. Schnorr 1976. The network complexity and the Turing machine complexity of finite functions. *Acta Informatica* 7, 95–107. DOI: https://doi.org/10.1007/BF00265223.

[Sch79] A. Schönhage. 1979. Storage modification machines. Technical Report, Mathematisches Institut, Universität Tübingen, Germany.

[Sim77] J. Simon. May. 1977. On feasible numbers. In *Proceedings of the 9th Annual ACM Symposium on Theory of Computing*. 195–207. DOI: https://doi.org/10.1145/800105.803409.

[Wyl79] J. C. Wyllie. 1979. *The complexity of parallel computations*. Ph.D. thesis and TR-79-387. Dept. of Computer Science, Cornell University.

(Reçu le 5 juin 1980)

A Time-Space Tradeoff for Sorting on a General Sequential Model of Computation

A. Borodin and S. Cook

Abstract

In a general sequential model of computation, no restrictions are placed on the way in which the computation may proceed, except that parallel operations are not allowed. We show that in such an unrestricted environment TIME·SPACE $= \Omega(N^2/\log N)$ in order to sort N integers, each in the range $[1, N^2]$.

Key words

time-space tradeoffs, conputational complexity, sorting, time lower bounds, space lower bounds

1 Introduction

Within the field of computational complexity, our inability to establish lower bounds on the complexity of "natural problems" stands in marked contrast to the progress that has been made in algorithmic design and analysis, and the progress in characterizing the central issues. To be fair, there are the following important exceptions:

Received by the editors July 28, 1980, and in final form May 4, 1981.
Originally published in SIAM J COMPUT May 1982 Volume 11 Number 2: pp. 287–297. https://epubs.siam.org/doi/pdf/10.1137/0211022

1. Relative to an appropriate reducibility, a problem can be shown "hard" for an entire complexity class. Then diagonalization can be used to infer a corresponding complexity lower bound. For example, see the discussion in Aho, Hopcroft and Ullman [AHU74, Chapt. 11].

2. For certain natural but "structured" models of computation, we have a number of interesting lower bounds. We use "structured" in the sense of Pippenger and Valiant's [PV76] use of "conservative" to mean that the computation can only proceed within a fixed mathematical structure (e.g., a partial order for comparison based models, a ring or field for algebraic complexity) and only uses the relations and functions within that structure for the computation (see also Borodin [Bor80]). For example, using comparison trees it is well known that sorting n elements requires at least $n \log n + O(n)$ comparisons.

3. On certain nonstructured but restricted models of computation we have a few results. For example, to recognize the set $\{w \# w^R\}$ on a *one-tape* Turing machine requires $\Omega(n^2)$ steps.

A *general* sequential model of computation can be viewed as a string processing machine. While the input string may arise as the encoding of a set of mathematical objects, there is no obligation to process these objects in ways prescribed by the mathematical structure. In this context complexity is measured as a function of the input (plus output) length. If we ignore "diagonalization based results", the following barriers are well recognized:

a. To establish a nonlinear lower bound on time.

b. To establish a nonlogarithmic lower bound on space.[1]

c. To establish a nonlogarithmic lower bound on depth (= parallel time).

Having recognized these barriers, it might seem wise to see if we can at least show that for some problem we cannot simultaneously achieve (say) linear time and logarithmic space. Such a result already appears in Cobham [Cob66], where he shows that for recognizing the set of perfect squares (or for recognizing $\{w \$ w^R\}$) we must have $T \cdot S = \Omega(n^2)$ for any computational device (including a multitape T·M.) having a separate *one head-read only input tape*. Here T = number of steps, S = "capacity" = $\log_2$ (number of configurations the machine enters when processing all strings of length n). The concept of "capacity" introduced by Cobham seems to capture just that property of space which lends itself to lower bound analysis. But whereas we accept a capacity lower bound on one of Cobham's general machines to be an "intrinsic" lower bound (i.e., independent of the choice of any reasonable computational models) on space requirements, we cannot say that a $T \cdot S = \Omega(n^2)$

1. That is, prove space is not $O(\log n)$.

lower bound has the same intrinsic quality, because of the restriction of having only one input head. More specifically, by easily adapting Cobham's argument (based on Hennie's [Hen65] crossing sequence technique), Tompa [Tom78] shows that sorting m numbers, each of length $\log m$ bits (hence $n = m \log m$), requires $T \cdot S = \Omega(n^2)$. But the proof literally states and shows that merging two lists of m sorted numbers would require the same lower bound. But for merging, the use of (say) two input heads would trivially (via a linear merge) permit a simultaneous linear time and logarithmic space merge. We are then led to the following question: Given k "random access" input heads, can we sort (say on a multitape T · M. or unit cost RAM) in simultaneous linear time and logarithmic space? The main result of this paper shows that indeed this is not possible. In fact we will establish a lower bound analogous to (and based upon) the lower bound of $T \cdot S = \Omega(n^2)$ established for sorting in the structured context of "branching programs" by Borodin, et al. [BFK+79]. Specifically we show $T \cdot S = \Omega(N^2/\log N)$, N the number of inputs and $N = \Omega(n/\log n)$ where n is the input length. To the best of our knowledge this is a unique result in that it establishes a lower bound (without diagonalization) on a completely unrestricted general model of computation. Unfortunately, we have not yet been able to establish a similar bound for a set recognition problem and we should also note that our methods do not appear applicable to Knuth's [Knu72] problem of in situ sorting.

2 The Formal Model and an Outline of the Proof

In a general model of computation, we might be able to solve a given problem by processing the input string in a manner which is completely outside the mathematical domain within which the problem has been defined. For example, consider solving for the existence of a path on a graph by using Strassen's matrix multiplication algorithm and modular arithmetic (see Fischer and Meyer [FM71]). It seems almost impossible to make sense out of the individual bit operations in terms of the original problems.

The "fortunate" fact for sorting is that such a problem, with its explicit requirement for "ongoing progress" (in the sense of having to output ranks) allows us to enjoy a structured view of the computation even though we are working within a general computational model. Indeed we shall try to mimic the proof for the structured case [BFK+79]. That proof was based on the following intuitive idea: if we don't compare many elements, then we can't know the ranks of many elements for many input permutations. We will need a somewhat more involved argument to show an analogous statement for the general model.

Before discussing the model, we should define the problem formally. We consider an input of the form $x_1 \# x_2 \# \cdots \# x_N$ where each x_i is an integer in $[1, N^2]$ and

is coded in binary. Hence the total length of the input is $O(N \log N)$. The sorting problem is to output a sequence of distinct pairs $i_1, r_1; i_2, r_2; \cdots; i_N, r_N$ such that x_{i_j} has rank r_j. (Without loss of generality we can assume that x_i's are distinct.) As in Borodin et al. [BFK+79] we can define the k-ranking subproblem; namely, output a sequence $i_1, r_1; \cdots; i_l, r_l, l \geq k$ which correctly represents the ranks of l of the x_{i_j}'s. (The l indices l_j which are assigned ranks may be different for different input values.)

This definition of sorting is not standard. Usually, one requires outputting the x_t values in sorted order. However, for the model we are considering, any algorithm which sorts in this usual manner can be adapted to one which outputs pairs $\langle i_j, r_j \rangle$ by assuming that the index i has been concatenated onto x_i as the low order bits. It will follow that a TIME · SPACE lower bound in our framework for inputs in the range $[1, N^2]$ will imply the same lower bound in the usual setting for inputs in the range $[1, N^3]$.

Our formal models are as follows:

Definition An *R-way integer tree program* is an R-ary tree (hereafter called an R tree), where each branch is labelled by elements of $[1, R]$, and each internal node is labelled by some index i (referring to x_i). The interpretation is that if the computation has proceeded to an internal node labelled by "x_i" then it will continue to proceed along edge u iff $x_i = u$. Output takes place at the leaves. In particular, it should now be clear to say how a computation tree solves the k-ranking problem, or more generally how a computation tree solves the k-ranking problem for some subset I of the possible inputs. The time complexity of a computation tree is its depth; that is, the maximum number of times inputs are accessed in a computation. Since we assume all x_i are distinct, any branch which has two edges with the same label u for distinct x_i will be inaccessible. We assume these inaccessible paths have been pruned.

An R-way integer branching program (hereafter called an R branching program) is the nonstructured analogue of a comparison branching program [Tom78]. Namely, it is a directed acyclic rooted graph with each nonsink node having outdegree R, with the R out edges labelled $1, 2, \cdots, R$. Without loss of generality, we can assume that the graph is in levels and that an edge out of a node at level l is directed to a node at level $l + 1$. (See Tompa [Tom78] for a discussion of the analogous assumption for comparison branching programs.) Outputs can now occur on any edge. The time complexity is again the depth and space = capacity = $\log_2$ (number of nodes in the graph). We can now state:

Main theorem *Let τ be an R branching program for sorting N integers and let $R = R(N) = N^2$. Then $T \cdot S = \Omega(N^2 / \log N)$ where T and S denote, respectively, the time and space complexity of τ.*

Before proceeding to the proof, we should comment briefly on the generality of the model. Suppose we have a general computational machine with k read-only, "random access" heads. It should be clear that by assuming $k = 1$ we will only slow down the machine by at most a constant factor (i.e., k). Our tree and branching programs assume that we will know an entire input x_i if we access any bit of that input. Hence, we are willing to ignore the $\log N$ factor it might cost to look at a given input. Each node of the computation graph represents a distinct state of the computation. Like Cobham [Cob66], it is profitable for us to ignore completely how (and if) the storage can be represented and manipulated. Again, we are willing to ignore the time spent manipulating the storage between accesses of the input. We thus argue that our model and the time and space measures are sufficiently general that any lower bounds do reflect an intrinsic property of the function (sorting) being computed.

Having presented and justified the model, we can now informally sketch the proof. To do so, it is helpful to review the proof for the structured case [BFK+79]. The basic lemma in that proof states that a $\{<, >\}$ comparison tree program on n inputs of depth (time) t can solve the k ranking problem for at most $(t+1)^k(n-k)!$ input permutations. This lemma is applied with $k = S =$ space. Thus for any $c > 1$, by making $t = \alpha n$ for α sufficiently small, we can say that the S ranking problem has been solved for at most a fraction $(1/c)^S$ of the $n!$ possible input permutations. Now if τ is a $\{<, >\}$ branching program for sorting, we consider the computation at the ith "stage" $= (i \cdot t)$th step, $i \leqq n/S$. In going from stage i to stage $i+1$, we can only correctly calculate S more ranks for at most $n!2^S \cdot (1/c)^S$ input permutations, since there are at most 2^S nodes at the ith stage, each of which can be considered the root of a tree program. Thus by an appropriate choice of $t = \Omega(n)$ we have $c > 2$ so that in going from stage i to stage $i+1$ we have computed more than S new ranks for at most a fraction $(1/d)^S$ of all possible input permutations. It follows that we will need at least $i = n/S$ stages to complete the computation, and hence $T = \Omega(n^2/S)$.

We want to establish the analog of the basic lemma, after which the rest of the proof follows exactly as before. We will show that for any $c > 1$, we can find suitable α such that any R-tree program ($R = N^2$) of depth $t = \alpha N$ can solve the $S \log N$ ranking problem for at most a fraction $(1/c)^S$ of the the possible inputs. In our case, that are $N!\binom{R}{N}$ possible input sequences $\langle x_1, \ldots, x_N \rangle$ since we are assuming distinct $\{x_i\}$.

In viewing the proof of the structured case, we can observe that *every* path in a computation tree can successfully solve the k ranking problem for at most a fraction $(t+1)^k/[n \cdot (n-1) \cdots (n-k+1)]$ of the permutations following that path. In our case, we can see that some short paths can be very successful; indeed if we discover that some $x_i = 1$ (or $x_i = N^2$) on a given path, then we know the smallest (respectively, largest) element for every input sequence on that path. Moreover, if

we find some $x_i = 2$ (and no x_j seen so far is equal to 1) we still have a pretty good chance if we guess that x_i is the smallest element. But, we can also see intuitively that our chance of guessing correctly as to which is the smallest element starts to decrease if we have only seen a few not so small numbers.

So this will be our approach for establishing the analogous main lemma: We assert that, with sufficiently high probability, at a leaf of an R-tree program the elements that we have seen on this path will be "spread out" in such a way that there is only a small probability (i.e., for only a small fraction of all possible input sequences) that we will correctly output S ranks.

3 The Proof of the Main Lemma

Throughout this section we will be considering R tree programs τ such that each leaf θ of τ is labelled with a ranking sequence $i_1, r_1; \cdots; i_l, r_l$, where $l = l_\theta$ may depend on the leaf θ. We say τ *solves the m ranking problem for an input sequence* $\langle x_1, \ldots, x_N \rangle$ provided this input leads to a leaf θ for which $l_\theta \geqq m$, and all l_θ ranks are correctly specified (i.e., x_{i_j} is the r_jth smallest input, $1 \leqq j \leqq l_\theta$). The following notation will be maintained: $t \leqq \frac{1}{2}N$ is the depth of the R tree program τ (we may asssume all paths in τ have length t by extending shorter ones if necessary), $N \geqq 2$ is the number of input elements, k is a positive integer satisfying $2f(k) \leqq N$, and $f(k)$ stands for $k\lfloor \log N \rfloor$. We will see that $R = R(N) = N^2$ is sufficiently large for our purposes, and since all results hold a fortiori for larger R, we will assume $R = N^2$. Our proofs will be formulated in the language of probability theory and we will speak of a random input in the sense that any of the $N!\binom{R}{N}$ possible input sequences are considered to be equally likely.

We are now ready to state the main lemma, which says that any sufficiently shallow R tree program (regardless of its capacity) cannot output many ranks correctly.

Lemma 1 *For all $c > 0$ there is an $\alpha > 0$ such that for all τ with $t \leqq \alpha N$ and N sufficiently large, and for all k with $f(k) \leqq t$, the set I of inputs for which τ solves the $2f(k)$ ranking problem satisfies $\#I/\left(N!\binom{R}{N}\right) \leqq (1/c)^k$. Restated: with probability at most $(1/c)^k$, τ correctly outputs $2f(k)$ or more ranks for a random input.*

Definition A set $S = \{x_{i_1}, \cdots, x_{i_t}\}$ of inputs is $\langle \rho, k \rangle$ *spread out* if for every subset $S' \subseteq S$ with $\#S' = f(k)$ there is a subset $\{y_1, \ldots, y_k\}$ (listed in increasing order) of S' such that $y_{j+1} - y_j - 1 \geqq \rho$, for $0 \leqq j \leqq k$. (Here $y_0 = 0$ and $y_{k+1} = R + 1$.)

Lemma 2 *For all integers $\beta > 0$ there is an $\alpha > 0$ such that, for all τ with $t \leqq \alpha N$ and all k with $f(k) \leqq t$, $P[\tau, k, \beta] \leqq (1/N)^k$, where $P[\tau, k, \beta]$ is the probability that a random input $\langle x_1, \ldots, x_N \rangle$ to τ will follow a path along which the accessed input elements are not $\langle \beta R/N, k \rangle$ spread out.*

Lemma 3 *For all $d > 0$ there is an integer $\beta > 0$ such that for all τ with N sufficiently large and for all k with $2f(k) \leqq N$ if the accessed input elements $\langle x_{i_1}, \cdots, x_{i_t}\rangle$ at a leaf θ of τ are $\langle \beta R/N, k\rangle$ spread out and the ranking sequence labelling θ contains at least $2f(k)$ ranks, then the fraction of those inputs leading to θ that are correctly ranked is at most $(1/d)^k$.*

Lemma 1 follows from Lemmas 2 and 3 as follows. Choose $d \geqq 2c$ in Lemma 3 to get β and apply Lemma 2 to get α. By Lemma 2 it does no harm to assume all leaves whose accessed inputs are not spread out always correctly solve the $2f(k)$ ranking problem, and the remaining leaves either output fewer than $2f(k)$ ranks or (by Lemma 3) are correct for too few inputs.

Proof of Lemma 2. Every leaf θ of τ uniquely determines a t-tuple $(x_{i_1}, \cdots, x_{i_t})$ of accessed elements, written in the order in which they are accessed on the path to θ. Conversely, every t-tuple of distinct integers in the interval $[1, R]$ uniquely determines a leaf. Thus there is a one to one correspondence between leaves and t-tuples, and exactly a $t!$ to one correspondence between leaves and sets of t distinct integers from $[1, R]$. Further, any two leaves have the same number of input sequences $(x_1, \ldots, x_N)$ leading to them. Therefore $P[\tau, k, \beta]$ is just that fraction of sets of t distinct integers from $[1, R]$ which are not $\langle \beta R/N, k\rangle$ spread out.

Divide the interval $[1, R]$ into N equal subintervals called *bins* of length N each (recall $R = N^2$). Let $\tilde{P}[t, N, k, \delta]$ be the probability that, when t balls are drawn (without replacement) from an urn of R balls numbered $1, 2, \cdots, R$, there exists some set of $f(k)$ of the drawn balls which lie in at most δ bins[2]. We claim that $\tilde{P}[t, N, k, \delta]$ is an upper bound on $P[\tau, k, \beta]$ where $\delta = k(\beta + 1) + \beta$. For, if S is the set of t drawn balls and if every subset $S' \subseteq S$ of $f(k)$ balls lies in $\delta + 1$ or more bins $B_1, \ldots, B_{\delta+1}$ (listed in the order in which these intervals occur in $[1, R]$), then we can choose one ball from each of the k bins $B_{i(\beta+1)}, 1 \leqq j \leqq k$, to form the required subset $\{y_1, \cdots, y_k\}$ in the definition of $(\beta R/N, k) = (\beta N, k)$ spread out. This is because at least β bins lie entirely to the left of y_1, at least β bins lie entirely to the right of y_k, and any two adjacent y_j's are separated by at least β bins (β bins equals βN elements).

To estimate $\tilde{P}[t, N, k, \delta]$, let $p(b_i)$ be the probability that a particular bin B_i has at least b_i balls (after t are drawn). We claim that $\prod_{i=1}^{\delta} p(b_i)$ is an overestimate of the probability that a particular set of δ bins $B_1, \cdots, B_\delta$ get packed (respectively) with at least $b_1, \cdots, b_\delta$ balls. This is because the condition that a set of bins has some minimum number of elements can only decrease the probability that a particular

2. Here is the essential place that the $\log N$ factor in our main result $T \cdot S = \Omega(N^2/\log N)$ enters the proof. Specifically, we cannot assert that $\tilde{P}$ would be sufficiently small if $f(k)$ were $O(k)$ rather than $k \log N$.

bin has at least b_i elements. Hence

$$\tilde{P}[t,N,k,\delta] \leqq \sum_{\substack{(b_1,\cdots,b_\delta \\ \Sigma b_i = f(k) \\ b_i \geqq 0}} \binom{N}{\delta} \prod_{i=1}^{\delta} p(b_i).$$

Here $\binom{N}{\delta}$ gives the number of ways to choose a set of crowded bins, and the summation represents the number of ways to pack a set of crowded bins.

We claim that for all $c \geqq 1$ there is $\alpha > 0$ such that $p(b) \leqq (1/c)^b$ (where $t \leqq \alpha N$).

Proof. The probability that a particular bin has exactly l balls is given by

$$\frac{\binom{N}{l}\binom{N^2-N}{t-l}}{\binom{N^2}{t}} \leqq \frac{N^l \left(N^2-N\right)^{t-l}}{\left(N^2-t\right)^t} \cdot \frac{t!}{l!\,(t-1)!} \leqq \left(\frac{2}{N}\right)^l \cdot t^l = \left(\frac{2t}{N}\right)^l \leqq (2\alpha)^l.$$

Thus

$$p(b) \leqq \sum_{l=b}^{t} (2\alpha)^l < \frac{2(\alpha)^b}{1-2\alpha} \leqq \left(\frac{1}{c}\right)^b \qquad \text{for } \alpha = \frac{1}{4c},$$

assuming $b \geqq 1$. The claim is obvious if $b = 0$.

We thus have

$$\begin{aligned}
\tilde{P}[t,N,k\delta] &\leqq \sum_{b_1+\cdots+b_\delta = f(k)} \binom{N}{\delta} \prod_{i=1}^{\delta} p(b_i) \\
&\leqq \left(f(k)+1\right)^\delta N^\delta \left(\tfrac{1}{c}\right)^{f(k)} \\
\left(\text{since } f(k) < N, f(k) = k\lceil \log N \rceil\right) \quad &\leqq N^{2\delta} \left(\tfrac{1}{c}\right)^{k\lceil \log N \rceil} \\
\left(\text{for } \delta = k(\beta+1)+\beta\right) \quad &\leqq N^{2k(\beta+1)+2\beta} \left(\tfrac{1}{N^{\log c}}\right)^k \\
&\leqq \left(\tfrac{1}{N}\right)^k \qquad \text{for sufficiently large } c. \quad \blacksquare
\end{aligned}$$

Proof of Lemma 3. Let $\left\{x_{i_1}, \ldots, x_{i_t}\right\}$, be the input elements accessed on the path to θ. Suppose at θ the labels assert that x_{j_v} has rank r_v for $1 \leqq v \leqq 2f(k)$. Note that we are not necessarily implying that any $x_{j_v} \in \{x_{i_1}, \cdots, x_{i_t}\}$ but, intuitively, one would expect a better chance at "guessing" the rank of an element which has been seen. Suppose that fewer than half of the indices for which θ assigns ranks are among the set $\{i_1, \cdots, i_t\}$. Then there is a set S of $u \geqq k[\log N]$ indices i for which θ assigns a rank and whose corresponding value x_i can be anything in the set $\{1, 2, \cdots, R\} - \{x_{i_1}, \cdots, x_{i_t}\}$. In particular, all $u!$ possible orderings of the set $\{x_i | \, i \in S\}$

are possible and equally likely, and a necessary condition that θ rank them properly is that they be in the right order. Hence at most a fraction $1/u!$ of the inputs leading to θ are correctly ranked, and $1/u! \leqq (1/d)^k$ for sufficiently large N, since $u \geqq k \log N$.

The remaining case is that half or more (that is at least $k[\log N] = f(k)$) of the indices for which θ assigns ranks are among the set $\{i_1, \cdots, i_t\}$. Let S' be the set of inputs at these indices (so $\#S' \geqq f(k)$). Since $\{x_{i_1}, \cdots, x_{i_t}\}$ is $\langle \beta R/N, k\rangle$ spread out, there is a subset $\{y_1, \cdots, y_k\}$ of S' such that $y_{j+1} - y_j - 1 \geqq \beta R/N$, $0 \leqq j \leqq k$, where $y_0 = 0, y_{k+1} = R + 1$. Let θ output the assertions that y_i has rank r_j, $1 \leqq j \leqq k$. These assertions are equivalent to saying that exactly $r_j - r_{j-1} - 1$ of the inputs lie in the open interval (y_{j-1}, y_j), $1 \leqq j \leqq k+1$, where we understand that $r_0 = 0$ and $r_{k+1} = N+1$. This in turn is equivalent to saying that exactly k_j of the inputs which θ *does not access* lie in the set $C_j = (y_{j-1}, y_j) - \{x_{i_1}, \cdots, x_{i_t}\}$, where $k_j = r_j - r_{j-1} - 1 - u_j$, and $u_j = \#(y_{j-1}, y_j) \cap \{x_{i_1}, \ldots, x_{i_t}\}$ (i.e., u_j is the number of inputs which θ knows to lie between y_{j-1} and y_j).

We have thus reduced our problem to a more traditional probability setting, namely that of the hypergeometric distribution (see Feller [Fel68, p. 43]). We have a population of size $n = R - t$, made up of n_i elements of "color i" (i.e., member of the set C_i), $1 \leqq i \leqq l = k + 1$. We seek an upper bound on the probability

$$p_{k_1 \cdots k_l} = \frac{\binom{n_1}{k_1}\binom{n_2}{k_2}\cdots\binom{n_l}{k_l}}{\binom{n}{r}}. \tag{1}$$

that a sample (without replacement) of size $r = N - t = \sum_{i=1}^{l} k_i$ will contain *exactly* k_i elements of color, $i, 1 \leqq i \leqq l$. The required bound is given by Lemma 4 below. For our application we have $rn_i/n = (N-t)(\neq C_i)/(R-t) \geqq (N-t)(\beta R/N - t)/(R-t)$. But $t \leqq \frac{1}{2}N$ and furthermore[3] $R = N^2$ so that $\beta R/N - t \geqq \frac{1}{2}\beta R/N$ for $\beta \geqq 1$. Thus $rn_i/n \geqq \beta/4$ since $N - t \geqq \frac{1}{2}N$. Further $l - 1 = k \leqq N/\log N$. Hence the constraints on r, n, n_i and l for Lemma 4 will be satisfied for sufficiently large N. Lemma 3 now follows from the following:

Lemma 4 *For all $d > 0$ there exists a $\beta > 0$ such that if $rn_i/n \geqq \beta$ for $1 \leqq i \leqq l$, $r \geqq 2l\beta$, and $n \geqq 2r$, then for all $k_1, \cdots, k_l$ the hypergeometric distribution satisfies*

$$p_{k_1 \cdots k_l} \leqq \left(\frac{1}{d}\right)^l.$$

3. This is the only place we need assume that R is as large as N^2.

We need the following two lemmas to prove Lemma 4. Note that Lemma 5 states that the value of k_i for which the hypergeometric distribution is maximal is close to the expected value rp_i of the number of elements of color i obtained in r draws. If this optimal value were exactly rp_i, the proof of Lemma 4 would be substantially simpler.

Lemma 5[4] *The values of k_i in the maximal term of the hypergeometric distribution $p_{k_1 \cdots k_l}$ satisfy*

$$\frac{rp_i}{1 + l/n} - 1 < k_i < rp_i + (l-1)p_i + 1, \tag{2}$$

where $p_i = n_i/n, 1 \leqq i \leqq l$.

Proof. For any pair (i,j) of distinct indices we calculate the ratio

$$\frac{p \cdots k_i + 1, \cdots, k_j - 1 \cdots}{p_{k_1, \cdots, k_l}} = \frac{(n_i - k_i)\, k_j}{(k_i + 1)\left(n_j - k_j + 1\right)}.$$

A necessary condition for $p_{k_1,\cdots,k_l}$ to be maximal is that the numerator does not exceed the denominator, or $(n_i - k_i)k_j \leqq (k_i + 1)(n_j - k_j + 1)$. If we divide by n and rearrange this becomes

$$p_i k_j \leqq p_j k_i + p_j + \frac{k_i - k_j + 1}{n}. \tag{3}$$

If we sum (3) over all $j \neq i$ and use the identities $\Sigma p_i = 1$ and $\Sigma k_i = r$, then we obtain the left half of (2). Similarly, if we sum (3) over all $i \neq j$ we obtain the right-hand side of (2). ∎

Lemma 6 *For all $\varepsilon > 0$ there is a z_ε such that for all θ and for all $z \geqq z_\varepsilon|\theta|$*

$$\left(1 + \frac{\theta}{z}\right)^z \geqq (1+\varepsilon)^{-|\theta|}\, e^\theta.$$

Note that z_ε does not depend on θ.

Proof. From elementary calculus we have $\lim_{z\to\infty}(1 + \theta/z)^z = e^\theta$. Setting $\theta = 1$ and -1 we conclude $(1 + 1/z)^z \geqq (1 + \varepsilon)^{-1}\, e$ and $(1 - 1/z)^z \geqq (1 + \varepsilon)^{-1}\, e^{-1}$ for $z \geqq z_\varepsilon$. Setting $z = z'|\theta|$ we have $(1 + \theta/z)^z = (1 + \theta/(z'|\theta|))^{z'|\theta|} \geqq (1+\varepsilon)^{-|\theta|}\, e^\theta$ for $z' \geqq z_\varepsilon$; i.e., for $z \geqq z_\varepsilon|\theta|$. ∎

4. Feller [Fel68, p. 171, Exercise 28] states a similar result for the multinomial distribution. Our proof is suggested by Feller's hints.

Proof of Lemma 4. We have

$$p_{k_1\cdots k_l} = \left(\prod n_i!\right) r!\,(n-r)!/[n!\prod(k_i!(n_i-k_i)!)].$$

Stirling's approximation implies that $1/C_o \leqq \sqrt{2\pi m}\,(m/e)^m/m! \leqq C_0$ for some constant $C_0 \geqq 1$ and all $m \geqq 1$. Using this approximation for each factorial, and substituting $rp_i + \theta_i$ for $k_i, 1 \leqq i \leqq 1$, where $p_i = n_i/n$ and θ_i has been chosen to maximize (1), we obtain

$$p_{k_1\cdots k_l} \leqq ABC_0^{3l+3}, \tag{4}$$

where

$$A = \sqrt{\frac{\left(\prod n_i\right) r\,(n-r)}{(2\pi)^{l-1}\, n \prod (rp_i+\theta_i)\,((n-r)\,p_i-\theta_i)}} \tag{5}$$

and

$$B = \frac{\left(\prod n_i^{n_i}\right) r^r\,(n-r)^{n-r}}{n^n \prod \left[(rp_i+\theta_i)^{rp_i+\theta_i}\,((n-r)\,p_i-\theta_i)^{(n-r)p_i-\theta_i}\right]}. \tag{6}$$

For (5) and (6) we have used the identity $n_i - k_i = (n-r)p_i - \theta_i$. Notice that all occurrences of e cancel, since $\Sigma n_i = n$.

Since $\Sigma p_i = 1$ and $\Sigma k_i = r$, it follows that $\Sigma\,\theta_i = 0$. This fact can be used to verify that if B' is the number obtained by substituting 0 for the two occurrences of θ_i in the denominator (but not in the exponents) in the expression for B, then $B' = 1$. Thus if we multiply and divide the denominator of (6) by $\prod[(rp_i)^{rp_i+\theta_i}\,((n-r)p_i)^{(n-r)p_i-\theta_i}]$ we can simplify and obtain

$$B = \left[\prod_{i=1}^{l}\left(\left(1+\frac{\theta_i}{rp_i}\right)^{rp_i+\theta_i}\left(1-\frac{\theta_i}{(n-r)\,p_i}\right)^{(n-r)p_i-\theta_i}\right)\right]^{-1}.$$

Now we apply Lemma 6 and use the fact that $(1+\theta/z)^\theta \geqq 1$ for all $z > 0$ and all θ to obtain $B \leqq (1+\varepsilon)^{2\Sigma|\theta_i|}$, provided

$$rp_i \geqq z_\varepsilon\,|\theta_i| \quad \text{and} \quad (n-r)\,p_i \geqq z_\varepsilon\,|\theta_i|\,, \quad 1 \leqq i \leqq l. \tag{7}$$

By Lemma 5, we have $|\theta_i| \leqq lrp_i/(n+l)+(l-1)p_i+2$, so $|\theta_i| \leqq 2lp_i+2$. By assumption, we have $rp_i \geqq \beta, (n-r) \geqq r$ and $r \geqq 2l\beta$. Hence

$$(n-r)\,p_i \geqq rp_i \geqq \frac{\beta\,|\theta_i|}{4}, \tag{8}$$

so the provisos (7) are satisfied for $\beta \geqq 4z_\varepsilon$. Now summing the bound $|\theta_i| \leqq 2lp_i + 2$, we obtain $\Sigma|\theta_i| \leqq 4l$, so

$$B \leqq (1+\varepsilon)^{8l}. \tag{9}$$

It remains to estimate A from (5). We rewrite the product Π in the denominator as the product of five factors:

$$\prod(rp_i) \cdot \prod\left(1+\frac{\theta_i}{rp_i}\right) \cdot \prod n_i \cdot \left(1-\frac{r}{n}\right)^l \cdot \prod\left(1-\frac{\theta_i}{(n-r)\,p_i}\right).$$

To estimate the first factor Πrp_i, notice that $\Sigma rP_i = r$, and each $rp_i \geqq \beta$ by assumption. With these constraints, this product obtains its minimum when all but one of the factors are as small as possible (namely β). Thus

$$\prod(rp_i) \geqq \beta^{l-1}(r-(l-1)\beta) > \frac{1}{2}r\beta^{l-1}.$$

From (8), we have $1 + \theta_i/(rp_i) \geqq \frac{1}{2}$ for $\beta \geqq 8$, so $\Pi(1+\theta_i/(rp_i)) \geqq 2^{-l}$. The same bound applies to the fifth factor and (since $n \geqq 2r$) to the fourth. The third factor cancels with the numerator. Thus

$$A \leqq \left[(2\pi)^{l-1}\beta^{l-1}2^{-3l+1}\right]^{-1/2} \leqq \left(\frac{1}{c}\right)^l$$

for any c and $\beta \geqq \beta_C$. Lemma 4 follows from this, (4) and (9). ∎

4 Proof of the Main Theorem

As indicated earlier in the paper, we will follow the general argument used in the structured case. As in § 3, we again assume $R = R(N) = N^2$. We let T denote the time (that is, the depth) of a branching program, and let S denote the space (that is, the capacity $= \log_2$ # nodes). Since we must clearly (by the simplest adversary argument) have $T \geqq N$ and $S \geqq \log_2 T$, we have $S \geqq \log_2 N$. Let us restate the main theorem.

Theorem *Let τ be an R branching program for sorting N integers. Then $T \cdot S = \Omega(N^2/\log N)$.*

Proof. Letting $c = 4$, use Lemma 1 to obtain α for N sufficiently large. We will now consider τ in stages, where every stage represents $t = [\alpha N]$ steps.

For $1 \leqq i \leqq N/(2f(S))$, let P_i be the fraction of input sequences for which τ has output at least $2if(S)$ ranks by the end of the ith stage. We shall now prove

$$P_i \leqq i\left(\frac{1}{2}\right)^S. \tag{$*$}$$

For each node θ on the $(i \cdot t)$th level (= end of stage i) let $P_{i,\theta}$ be the fraction of input sequences which lead to θ and for which τ outputs at least $2f(S)$ ranks during the $i+1$st stage. If we expand the part of the $(i+1)$st stage that is rooted at θ into an R tree, we see by Lemma 1 that (regardless of what has happened in earlier stages) $P_{i,\theta} \leqq (\frac{1}{4})^S$. Since there are at most 2^S nodes θ at level $i \cdot t$, we have $P_{i+1} \leqq P_i + \Sigma_\theta P_{i,\theta} \leqq P_i + 2^S \cdot (\frac{1}{4})^S$, or $P_{i+1} \leqq P_i + (\frac{1}{2})^S$. This inequality holds for $0 \leqq i \leqq N/(2(f(S))$, if we define $P_0 = 0$. The inequality $(*)$ now follows by induction on i.

Recall $f(S) = S\lceil \log N \rceil$. If $2f(S) > N$ then $S > N/2\lceil \log N \rceil$, so $ST = \Omega(N^2/\log N)$ in this case. If $2f(S) \leqq N$, then we can set $i = i_0 = \lfloor N/(2f(S)) \rfloor$ in $(*)$ to obtain since $(S \geqq \log N)$ $P_{i_0} \leqq (N/(2f(S))) \cdot 1/N = 1/(2f(S)) < 1$. Hence at stage i_0 for some input, τ has output fewer than $2_{i_0} f(S) \leqq N$ ranks, so $T \geqq i_0 t = \lfloor N/(2f(S)) \rfloor \cdot \lfloor \alpha N \rfloor = \Omega(N^2/(S \log N))$ steps. ∎

5 Conclusion

In order to better appreciate the application of the main theorem, we offer the following example.

Let M be any machine (say, a unit cost RAM or vector machine with operations $+, -, \times, \div, \uparrow$) whose inputs are accessed from a random access *read-only* input device. We only insist that there is a bound on the number of inputs accessible on a given computation step. Choose any "fair" definition of space, e.g., space $= \max_j \sum_{i=1}^{t} \lceil \log(r_i^j + 1) \rceil$ where r_i^j is the contents of register i at time j and t is the largest register used. For such a machine the theorem yields $T \cdot S = \Omega(N^2/\log N)$. And, of course, the same result holds for multidimensional Turing machines, etc.

Although the lower bound $T \cdot S = \Omega(N^2/\log N)$ established in this paper for a general model of computation differs by a log factor from the lower bound for the structured case [BFK+79], the upper bounds for the structured case apply unchanged. This is because a "structured algorithm" is a $\{<, >\}$ branching program, and a comparison $x_i < x_j$ over the domain $[1, R]$ can be carried out on an R branching program in two time steps and $R+2$ nodes. However, in order to be sure that the time and space of the simulating program are of the same order as the time and space of the original program, it is necessary to assume $R = O(N^k)$ for some k. Under this assumption, the upper bound $T \cdot S = O(N^2 \log N)$, for $\Omega(\log N) \leqq S \leqq O(N)$ recently established by Frederickson [Fre80] for a unit cost "structured" random access machine with suitable instructions applies to an R branching program. (Frederickson's bound generalizes the one in [BFK+79]). It is worth noting that for "unstructured" (i.e., general) random access machines, the upper bound can be extended, using radix sort, to the case $T = O(N)$ and $S = O(N \log N)$.

We thus have a $\log^2 N$ discrepancy in the upper and lower bounds. We note that we can improve on the upper bounds when $R = N+O(N)$, say by finding the missing elements.

It seems to us, however, that the discrepancy in the bounds is far less important than the need to establish analogous results for a set-recognition problem; for example, determining if $X \cap Y = \emptyset$. At the present time such a time-space result has not yet been established for the structured comparison model. We believe that our results suggest that proofs for the structured model may provide a framework for the general model. However, it must be noted that the less constructive variant of branching programs for "silent sorting" mentioned in Borodin et al. [BFK+79, Conclusion] becomes trivial in the general setting.

In retrospect, we can see that our methods are quite "brute-force". In particular, we do not make an essential use of an adversary. Rather what we have is basically a counting argument. Moreover, we do not make full use of the fact that space is limited throughout the computation; we only use the fact that it is restricted at certain points of the computation. We suspect that the set recognition problems will entail a more sophisticated argument.

A more general view of time-space complexity is captured in Cook's class SC [Coo79, Coo80] (formerly PLOPS); that is, those problems for which there exist algorithms which run *simultaneously* in polynomial (sequential) time and $\log^k$ (for some k) space. Obviously, any problem (e.g., sorting, $X \cap Y = \emptyset$?, etc.) which is in log space, is also in SC. A central issue for computational complexity is to establish the conjecture (assuming it is true) that $\mathrm{P} \cap (\cup_k \mathrm{DSPACE}(\log^k)) \not\supseteq \mathrm{SC}$. Cook and Tompa (see Tompa [Tom78]) show that the structured branching program model (with either $\{=, \neq\}$ or $\{<, =, >\}$ as the allowable comparisons) may provide a sufficiently general setting for this conjecture.

Another important direction for future work lies in the related (but apparently different) question of size vs. depth. The recent work of Pippenger [Pip79], Ruzzo [Ruz79a], and Dymond and Cook [DC80], has focused attention on the stability and importance of the class NC; that is, those problems for which there are algorithms which run *simultaneously* in polynomial size (= sequential operations) and $\log^k$ depth (= parallel time). Again, it is a central issue in complexity to establish the conjecture $P \cap (\cup_k \text{parallel time}(\log^k)) \not\supseteq \mathrm{NC}$.

Motivated by the results of this paper, we would like to find a problem for which (say) size $\cdot$ depth $= \Omega(N^2)$. Sorting will not suffice since we can sort simultaneously in $\log^2$ depth and $N \log^2 N$ size using a Batcher sorting network. However, one is tempted to conjecture that any Boolean circuit for sorting which uses only $k \log N$ depth requires $cN^{1+\varepsilon}$ size where c and ε will depend upon k. The class of problems

which are computable by a log depth, $N \log^k N$ size circuit is a class of practical importance. We suspect that it will be difficult to prove that a given problem does not belong to this class.

Acknowledgment

We sincerely thank Romas Aleliunas and Patrick Dymond for their many helpful suggestions.

References

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

[Bor80] A. Borodin. February. 1980. *Structured vs General Models in Computational Complexity*. Presented at Internationales Symposium über Logik und Algorithmik zu Ehren von Professor Ernst Specker. Zurich, L'Enseignement Mathematique, to appear.

[BFK+79] A. Borodin, M. J. Fischer, D. G. Kirkpatrick, N. A. Lynch, and M. Tompa. October 29–31. 1979. A time–space tradeoff for sorting on non-oblivious machines. In *20th Annual Symposium on Foundations of Computer Science*. IEEE, San Juan, Puerto Rico, 319–327. DOI: https://doi.org/10.1109/SFCS.1979.4.

[Cob66] A. Cobham. 1966. *The Recognition Problem for the Set of Perfect Squares*. Conference Record, IEEE 7th Annual Symposium on Switching and Automata Theory, 78–87.

[Coo79] S. A. Cook. April 30–May 2. 1979. Deterministic CFL's are accepted simultaneously in polynomial time and log squared space. In *Proceedings of the 11th Annual ACM Symposium on Theory of Computing*. Atlanta, Georgia. ACM, 338–345. DOI: https://doi.org/10.1145/800135.804426.

[Coo80] S. A. Cook. February. 1980. *Towards a complexity theory of synchronous parallel computation.* Presented at Internationales Symposium über Logik und Algorithmik zu Ehren von Professor Ernst Specker, Zurich, L'Enseignement Mathematique, to appear.

[DC80] P. W. Dymond and S. A. Cook. October 13–15. 1980. Hardware complexity and parallel computation (preliminary version). In *21st Annual Symposium on Foundations of Computer Science*. Syracuse, New York. IEEE, 360–372. DOI: https://doi.org/10.1109/SFCS.1980.22.

[Fel68] W. Feller. 1968. *An Introduction to Probability Theory and its Applications*. I, John Wiley, New York.

[FM71] M. Fischer and A. Meyer. 1971. *Boolean Matrix Multiplication and Transitive Closure*. Conference Record, IEEE 12th Annual Symposium on Switching and Automata Theory, 129–131.

[Fre80] G. N. Frederickson. January. 1980. *Upper bounds for time-space trade-offs in sorting and selection*, Tech. Rep. CS-80-3, Dept. Computer Science, Pennsylvania State University.

[Hen65] F. Hennie. 1965. *Crossing Sequences and Off-Line Turing Machine Computations*. Conference Record IEEE Symposium on Switching Circuit Theory and Logical Design, 179–190.

[Knu72] D. E. Knuth. 1972. Mathematical analysis of algorithms. In C. V. Freeman (Ed.), *Proceedings of the IFIP Congress 71*, *Vol. 1*. North-Holland, Amsterdam, 19–27.

[Pip79] N. Pippenger. October 29–31. 1979. On simultaneous resource bounds (preliminary version). In *20th Annual Symposium on Foundations of Computer Science*. IEEE, San Juan, Puerto Rico, 307–311. DOI: https://doi.org/10.1109/SFCS.1979.29.

[Ruz79a] W. L. Ruzzo. October 29–31. 1979a. On uniform circuit complexity (extended abstract). In *20th Annual Symposium on Foundations of Computer Science*. IEEE, San Juan, Puerto Rico, 312–318. DOI: https://doi.org/10.1109/SFCS.1979.31.

[Tom78] M. Tompa. July. 1978. *Time-space tradeoffs for straight-line and branching programs.* Tech. Rep. 122/78, Dept. Computer Science, Univ. of Toronto.

Pebbles and Branching Programs for Tree Evaluation

Stephen A. Cook, Pierre McKenzie, Dustin Wehr, Mark Braverman, and Rahul Santhanam

We introduce the *tree evaluation problem*, show that it is in **LogDCFL** (and hence in **P**), and study its branching program complexity in the hope of eventually proving a superlogarithmic space lower bound. The input to the problem is a rooted, balanced d-ary tree of height h, whose internal nodes are labeled with d-ary functions on $[k] = \{1, \ldots, k\}$, and whose leaves are labeled with elements of $[k]$. Each node obtains a value in $[k]$ equal to its d-ary function applied to the values of its d children. The output is the value of the root. We show that the standard black pebbling algorithm applied to the binary tree of height h yields a deterministic k-way branching program with $O(k^h)$ states solving this problem, and we prove that this upper bound is tight for $h = 2$ and $h = 3$. We introduce a simple semantic restriction called *thrifty* on k-way branching programs solving tree evaluation problems and show that the same state bound of $\Theta(k^h)$ is tight for all $h \geq 2$ for deterministic thrifty programs. We introduce fractional pebbling for trees and show

Versions of parts of this article appeared in Braverman et al. [BCM+09a, BCM+09b].
Authors' addresses: S. Cook (corresponding author), Department of Computer Science, University of Toronto, Toronto, Canada; email: sacook@cs.toronto.edu; P. McKenzie, Université de Montréal, Montréal, Canada; D. Wehr and M. Braverman, Department of Computer Science, University of Toronto, Toronto, Canada; R. Santhanam, University of Edinburgh, UK.

Originally published in ACM Transactions on Computation Theory, Vol. 3, No. 2, Article 4, Publication date: January 2012.
Original DOI 10.1145/2077336.2077337 http://doi.acm.org/10.1145/2077336.2077337

that this yields nondeterministic thrifty programs with $\Theta(k^{h/2+1})$ states solving the Boolean problem "determine whether the root has value 1", and prove that this bound is tight for $h = 2, 3, 4$. We also prove that this same bound is tight for unrestricted nondeterministic k-way branching programs solving the Boolean problem for $h = 2, 3$.

Categories and Subject Descriptors: F.2.0 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity

General Terms: Theory

Additional Key Words and Phrases: Branching programs, logDCFL, log space, lower bounds

ACM Reference Format:
Cook, S., McKenzie, P., Wehr, D., Braverman, M., and Santhanam, R. 2012. Pebbles and branching programs for tree evaluation. ACM Trans. Comput. Theory 3, 2, Article 4 (January 2012), 43 pages.
DOI = 10.1145/2077336.2077337 http://doi.acm.org/10.1145/2077336.2077337

1 Introduction

What follows is a nondecreasing sequence of standard complexity classes between $\mathbf{AC}^0(6)$ and the polynomial hierarchy.

$$\mathbf{AC}^0(6) \subseteq \mathbf{NC}^1 \subseteq \mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{LogCFL} \subseteq \mathbf{AC}^1 \subseteq \mathbf{NC}^2 \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PH} \tag{1}$$

A problem in $\mathbf{AC}^0(6)$ is given by a uniform family of polynomial size bounded depth circuits with unbounded fan-in Boolean and mod 6 gates. As far as we can tell an $\mathbf{AC}^0(6)$ circuit cannot determine whether a majority of its input bits are ones, and yet we cannot provably separate $\mathbf{AC}^0(6)$ from any of the other classes in the sequence. This embarrassing state of affairs motivates this article (as well as much of the lower bound work in complexity theory).

We propose a candidate for separating **NL** from **LogCFL**. The *tree evaluation problem* $FT_d(h, k)$ is defined as follows. The input to $FT_d(h, k)$ is a balanced d-ary tree of height h, denoted T_d^h (see Figure 1). Attached to each internal node i of the tree is some explicit function $f_i : [k]^d \to [k]$ specified as k^d integers in $[k] = \{1, \ldots, k\}$. Attached to each leaf is a number in $[k]$. Each internal tree node takes a value in $[k]$ obtained by applying its attached function to the values of its children. The function problem $FT_d(h, k)$ is to compute the value of the root, and the Boolean problem $BT_d(h, k)$ is to determine whether this value is 1.

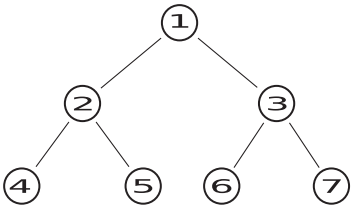


Figure 1 A height 3 binary tree T_2^3 with nodes numbered heap style.

It is not hard to show that a deterministic logspace-bounded polytime auxiliary pushdown automaton decides $BT_d(h,k)$, where d,h and k are input parameters. This implies by Sudborough [Sud78] that $BT_d(h,k)$ belongs to the class **LogDCFL** of languages logspace reducible to a deterministic context-free language. The latter class lies between **L** and **LogCFL**, but its relationship with **NL** is unknown (see Mahajan [Mah07] for a recent survey). We conjecture that $BT_d(h,k)$ does not lie in **NL**. A proof would separate **NL** and **LogCFL**, and hence (by (1)) separate $\mathbf{NC}^1$ and $\mathbf{NC}^2$.

Thus we are interested in proving superlogarithmic space upper and lower bounds (for fixed degree $d \geq 2$) for $BT_d(h,k)$ and $FT_d(h,k)$. Notice that for each constant $k = k_0 \geq 2$, $BT_d(h,k_0)$ is an easy generalization of the Boolean formula value problem for balanced formulas, and hence it is in $\mathbf{NC}^1$ and **L**. Thus it is important that k be an unbounded input parameter.

We use Branching Programs (BPs) as a nonuniform model of Turing machine space: A lower bound of $s(n)$ on the number of BP states implies a lower bound of $\Theta(\log s(n))$ on Turing machine space, but to go the other way, that is, to deduce BP size lower bounds from Turing machine space lower bounds, we would need to defeat a Turing machine supplied with an advice string for each input length. Thus BP state lower bounds are stronger than TM space lower bounds, but we do not know how to take advantage of the uniformity of TMs to get the supposedly easier lower bounds on TM space. In this article all of our lower bounds are nonuniform and all of our upper bounds are uniform.

In the context of branching programs we think of d and h as fixed, and we are interested in how the number of states required grows with k. To indicate this point of view we write the function problem $FT_d(h,k)$ as $FT_d^h(k)$ and the Boolean problem $BT_d(h,k)$ as $BT_d^h(k)$. For this it turns out that k-way BPs are a convenient model, since an input for $BT_d^h(k)$ or $FT_d^h(k)$ is naturally presented as a tuple of elements in $[k]$. Each nonfinal state in a k-way BP queries a specific element of the tuple, and branches k possible ways according to the k possible answers.

It is natural to assume that the inputs to Turing machines are binary strings, so 2-way BPs are a closer model of TM space than are k-way BPs for $k > 2$. But every 2-way BP is easily converted to a k-way BP with the same number of states, and every k-way BP can be converted to a 2-way BP with an increase of only a factor of k in the number of states, so for the purpose of separating **L** and **P** we may as well use k-way BPs.

Of course the number of states required by a k-way BP to solve the Boolean problem $BT_d^h(k)$ is at most the number required to solve the function problem $FT_d^h(k)$. In the other direction it is easy to see (Lemma 2.3) that $FT_d^h(k)$ requires at most a factor of k more states than $BT_d^h(k)$. From the point of view of separating **L** and **P** a factor of k is not important. Nevertheless it is interesting to compare the two numbers, and in some cases (Corollary 5.2) we can prove tight bounds for both: For deterministic BPs solving height 3 trees they differ by a factor of $\log k$ rather than k.

The best (i.e., fewest states) algorithms that we know for deterministic k-way BPs solving $FT_d^h(k)$ come from black pebbling algorithms for trees: If p pebbles suffice to pebble the tree T_d^h then $O(k^p)$ states suffice for a BP to solve $FT_d^h(k)$ (Theorem 3.4). This upper bound on states is tight (up to a constant factor) for trees of height $h = 2$ or $h = 3$ (Corollary 5.2), and we suspect that it may be tight for trees of any height.

There is a well-known generalization of black pebbling called black-white pebbling which naturally simulates nondeterministic algorithms. Indeed if p pebbles suffice to black-white pebble T_d^h then $O(k^p)$ states suffice for a nondeterministic BP to solve $BT_d^h(k)$. However the best lower bound we can obtain for nondeterministic BPs solving $BT_2^3(k)$ (see Figure 1) is $\Omega(n^{2.5})$, whereas it takes 3 pebbles to black-white pebble the tree T_2^3. This led us to rethink the upper bound, and we discovered that there is indeed a nondeterministic BP with $O(k^{2.5})$ states which solves $BT_2^3(k)$. The algorithm comes from a black-white pebbling of T_2^3 using only 2.5 pebbles: It places a half-black pebble on node 2, a black pebble on node 3, and adds a half-white pebble on node 2, allowing the root to be black-pebbled (see Figure 2).

This led us to the idea of fractional pebbling in general, a natural generalization of black-white pebbling. A fractional pebble configuration on a tree assigns two nonnegative real numbers $b(i)$ and $w(i)$ totalling at most 1, to each node i in the tree, with appropriate rules for removing and adding pebbles. The idea is to minimize the maximum total pebble weight on the tree during a pebbling procedure which starts and ends with no pebbles and has a black pebble on the root at some point.

It turns out that nondeterministic BPs nicely implement fractional pebbling procedures: If p pebbles suffice to fractionally pebble T_d^h then $O(k^p)$ states suffice

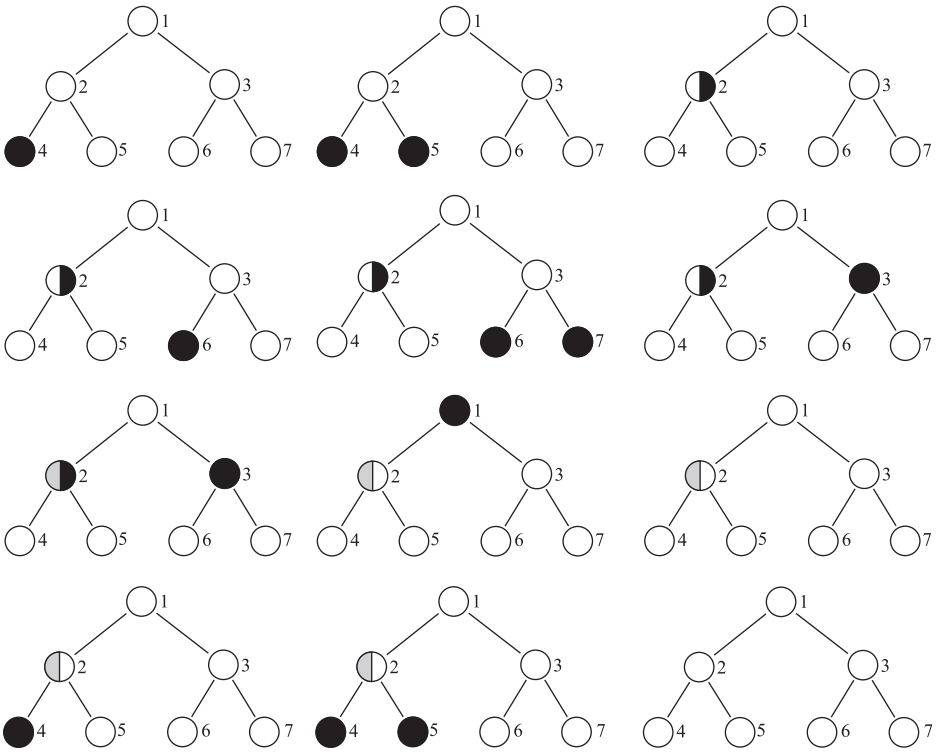


Figure 2 An optimal fractional pebbling sequence for the height 3 tree using 2.5 pebbles, all configurations included (except the empty starting configuration). The gray half-circle means the *white* value of that node is .5, whereas unshaded area means absence of pebble value. So for example in the seventh configuration, node 2 has black value .5 and white value .5, node 3 has black value 1, and the remaining nodes all have black and white value 0.

for a nondeterministic BP to solve $BT_d^h(k)$. The idea is that if node i has a fraction $b(i) + w(i)$ pebbles then the corresponding BP configuration remembers a fraction $b(i) + w(i)$ of the $\log k$ bits specifying the value of node i, where $b(i)$ bits are verified and $w(i)$ bits are conjectured. After much work we have not been able to improve upon this $O(k^p)$ upper bound for any $d, h \geq 2$. We prove it is optimal for trees of height 3 (Corollary 5.2).

We can prove that for fixed degree d the number of pebbles required to pebble (in any sense) the tree T_d^h grows as $\Theta(h)$, so the p in the preceding best-known upper bound of $O(k^p)$ states grows as $\Theta(h)$. This and the following fact motivate further study of the complexity of $FT_d^h(k)$.

Fact 1 A lower bound of $\Omega(k^{r(h)})$ for *any* unbounded function $r(h)$ on the number of states required to solve $FT_d^h(k)$ implies that $\mathbf{L} \neq \mathbf{LogCFL}$ (Theorem 3.1 and Corollary 3.3).

Proving tight bounds on the number of pebbles required to fractionally pebble a tree turns out to be much more difficult than for the case of whole black-white pebbling. However we can prove good upper and lower bounds. For binary trees of any height h we prove an upper bound of $h/2+1$ and a lower bound of $h/2-1$ (the upper bound is optimal for $h \leq 4$). These bounds can be generalized to d-ary trees (Theorem 4.4).

We introduce a natural semantic restriction on BPs which solve $BT_d^h(k)$ or $FT_d^h(k)$: A k-way BP is *thrifty* if it only queries the function $f(x_1, \ldots, x_d)$ associated with a node when $(x_1, \ldots, x_d)$ are the correct values of the children of the node.

It is not hard to see that the deterministic BP algorithms that implement black pebbling are thrifty. With some effort we were able to prove a converse (for binary trees): If p is the minimum number of pebbles required to black-pebble T_2^h then every deterministic thrifty BP solving $BT_2^h(k)$ (or $FT_2^h(k)$) requires at least k^p states. Thus any deterministic BP solving these problems with fewer states must query internal nodes $f_i(x,y)$ where (x,y) are not the values of the children of node i. For the decision problem $BT_2^h(k)$ there is indeed a nonthrifty deterministic BP improving on the bound by a factor of $\log k$ (Theorem 5.1 (16)), and this is tight for $h = 3$ (Corollary 5.2). But we have not been able to improve on thrifty BPs for solving any function problem $FT_d^h(k)$.

The nondeterministic BPs that implement fractional pebbling are indeed thrifty. However here the converse is far from clear: there is nothing in the definition of *thrifty* that hints at fractional pebbling. We have been able to prove that thrifty BPs cannot beat fractional pebbling for binary trees of height $h = 4$ or less, but for general trees this is open.

It is not hard to see that for black pebbling, fractional pebbles do not help. This may explain why we have been able to prove tight bounds for deterministic thrifty BPs for all binary trees, but only for trees of height 4 or less for nondeterministic thrifty BPs.

We pose the following as another interesting open question.

Thrifty Hypothesis Thrifty BPs are optimal among k-way BPs solving $FT_d^h(k)$.

Proving this for deterministic BPs would show $\mathbf{L} \neq \mathbf{LogDCFL}$, and for nondeterministic BPs would show $\mathbf{NL} \neq \mathbf{LogCFL}$. Disproving this would provide interesting new space-efficient algorithms and might point the way to new approaches for proving lower bounds.

The lower bounds mentioned before for unrestricted branching programs when the tree heights are small are obtained in two ways: first using the Nečiporuk method [Neč66] (or see Wegener [Weg00]), and second using a method that analyzes the state sequences of the BP computations. Using the state sequence method we have not yet beat the $\Omega(n^2)$ deterministic branching program size barrier (neglecting log factors) inherent to the Nečiporuk method for Boolean problems, but we can prove lower bounds for function problems which cannot be matched by the Nečiporuk method (Theorems 5.5, 5.6, 5.9, 5.10). For nondeterministic branching programs with states of unbounded outdegree, we show that both methods yield a lower bound of $\Omega(n^{3/2})$ states (neglecting logs) for the decision problem BT_2^3.

1.1 Summary of Contributions

— We introduce a family of computation problems $FT_d^h(k)$ and $BT_d^h(k)$, $d, h \geq 2$, which we propose as good candidates for separating **L** and **NL** from apparently larger complexity classes in (1). Our goal is to prove space lower bounds for these problems by proving state lower bounds for k-way branching programs which solve them. For $h = 3$ we can prove tight bounds for each $d \geq 2$ on the number of states required by k-way BPs to solve them, namely, (from Corollary 5.2);

$$\Theta(k^{(3/2)d-1/2}) \text{ for nondeterministic BPs solving } BT_d^3(k),$$
$$\Theta(k^{2d-1}/\log k) \text{ for deterministic BPs solving } BT_d^3(k),$$
$$\Theta(k^{2d-1}) \text{ for deterministic BPs solving } FT_d^3(k).$$

— We introduce a simple and natural restriction called *thrifty* on BPs solving $FT_d^h(k)$ and $BT_d^h(k)$. The best-known upper bounds for deterministic BPs solving $FT_d^h(k)$ and for nondeterministic BPs solving $BT_d^h(k)$ are realized by thrifty BPs (although deterministic nonthrifty BPs can save a factor of $\log k$ states over deterministic thrifty BPs solving the decision problem $BT_2^h(k)$). Proving even much weaker lower bounds than these upper bounds for unrestricted BPs would separate **L** from **LogCFL** (see Fact 1 earlier). We prove that for binary trees deterministic thrifty BPs cannot do better than implement black pebbling (this is far from obvious).[1]

— We formulate the Thrifty Hypothesis. Either a proof or a disproof would have interesting consequences.

1. In Wehr [Weh11] coauthor Wehr solved an open problem in Gál et al. [GKM08] by adapting our thrifty lower bound proof to prove an exponential lower bound on the size of semantic incremental branching programs solving GEN.

— We introduce *fractional pebbling* as a natural generalization of black-white pebbling for simulating nondeterministic space bounded computations. We prove almost tight lower bounds for fractionally pebbling binary trees (Theorem 4.4). The best-known upper bounds for nondeterministic BPs solving $FT_d^h(k)$ come from fractional pebbling, and these can be implemented by thrifty BPs. An interesting open question is to prove that nondeterministic thrifty BPs cannot do better than implement fractional pebbling. (We prove this for $h = 2, 3, 4$.)

— We use a "state sequence" method for proving size lower bounds for branching programs solving $FT_d^h(k)$ and $BT_d^h(k)$, and show that it improves on the Nečiporuk method for certain function problems.

The next major step is to prove good lower bounds for trees of height $h = 4$. If we can prove the Thrifty Hypothesis for deterministic BPs solving the function problem (and hence the decision problem) for trees of height 4, then we would beat the $\Omega(n^2)$ limitation mentioned before on Nečiporuk's method. See Section 6 (Conclusion) for this argument, and a comment about the nondeterministic case.

1.2 Relation to Previous Work

Taitslin [Tai05] proposed a problem similar to $BT_2^h(k)$ in which the functions attached to internal nodes are specific quasigroups, in an unsuccessful attempt to prove $\mathbf{NL} \neq \mathbf{P}$.

Gál et al. [GKM08] proved exponential lower bounds on the size of restricted n-way branching programs solving versions of the problem GEN. Like our problems $BT_d^h(k)$ and $FT_d^h(k)$, the best-known upper bounds for solving GEN come from pebbling algorithms.

As a concrete approach to separating $\mathbf{NC}^1$ from $\mathbf{NC}^2$, Karchmer et al. [KRW95] suggested proving that the circuit depth required to compose a Boolean function with itself h times grows appreciably with h. They proposed the *universal composition relation* conjecture, stating that an abstraction of the composition problem requires high communication complexity, as an intermediate goal to validate their approach. This conjecture was later proved in two ways, first [EIRS01] using innovative information-theoretic machinery and then [HW93] using a clever new complexity measure that generalizes the subadditivity property implicit in Nečiporuk's lower bound method [Neč66, Weg00]. Proving the conjecture thus cleared the road for the approach, yet no sufficiently strong unrestricted circuit lower bounds could be proved using it so far.

Edmonds et al. [EIRS01] noted that the approach would in fact separate $\mathbf{NC}^1$ from $\mathbf{AC}^1$. They also coined the name *Iterated Multiplexor* for the most general

computational problem considered in Karchmer et al. [KRW95], namely composing in a tree-like fashion a set of explicitly presented Boolean functions, one per tree node. Our problem $FT^h_d(k)$ can be considered as a generalization of the Iterated Multiplexor problem in which the functions map $[k]^d$ to $[k]$ instead of $\{0,1\}^d$ to $\{0,1\}$. This generalization allows us to focus on getting lower bounds as a function of k when the tree is fixed.

For time-restricted branching programs, Borodin et al. [BRS93] exhibited a family of Boolean functions that require exponential size to be computed by nondeterministic syntactic read-k times BPs. Later Beame et al. [BSSV03] exhibited such functions that require exponential size to be computed by randomized BPs whose computation time is limited to $o(n\sqrt{\log n/\log\log n})$, where n is the input length. However all these functions can be computed by polynomial size BPs when time is unrestricted.

In the present article we consider branching programs with no time restriction such as read-k times. However the smallest size deterministic BPs known to us that solve $FT^h_d(k)$ implement the black pebbling algorithm, and these BPs happen to be (syntactic) read-once.

1.3 Organization

The article is organized as follows. Section 2 defines the main notions used in this article, including branching programs and pebbling. Section 3 relates pebbling and branching programs to Turing machine space, noting in particular that a k-way BP size lower bound of $\Omega(k^{\text{function}(h)})$ for $BT^h_d(k)$ would show $\mathbf{L} \neq \mathbf{LogCFL}$. Section 4 proves upper and lower bounds on the number of pebbles required to black, black-white, and fractionally pebble the tree T^h_d. These pebbling bounds are exploited in Section 5 to prove upper bounds on the size of branching programs. BP lower bounds are obtained using the Nečiporuk method in Section 5.1. Alternative proofs to some of these lower bounds using the "state sequence method" are given in Section 5.2. An example of a function problem for which the state sequence method beats the Nečiporuk method is given in Theorems 5.5 and 5.9. Section 5.3 contains bounds for thrifty branching programs.

2 Preliminaries

We assume some familiarity with complexity theory, such as can be found in Goldreich [Gol08]. We write $[k]$ for $\{1,2,\ldots,k\}$. For $d,h \geq 2$ we use T^h_d to denote the balanced d-ary tree of height h.

Warning Here the *height* of a tree is the number of levels in the tree, as opposed to the distance from root to leaf. Thus T^2_2 has just 3 nodes.

We number the nodes of T_d^h as suggested by the heap data structure. Thus the root is node 1, and in general the children of node i are (when $d = 2$) nodes $2i, 2i+1$ (see Figure 1).

Definition 2.1 **Tree Evaluation Problems**

Given: The tree T_d^h with each nonleaf node i independently labeled with a function $f_i : [k]^d \to [k]$ and each leaf node independently labeled with an element from $[k]$, where $d, h, k \geq 2$.

Function evaluation problem $FT_d^h(k)$: Compute the value $v_1 \in [k]$ of the root 1 of T_d^h, where in general $v_i = a$ if i is a leaf labeled a and $v_i = f_i(v_{j_1}, \ldots, v_{j_d})$ if the children of i are $j_1, \ldots, j_d$.

Boolean problem $BT_d^h(k)$: Decide whether $v_1 = 1$.

2.1 Branching Programs

A family of branching programs serves as a nonuniform model of a Turing machine. For each input size n there is a BP B_n in the family which models the machine on inputs of size n. The states (or nodes) of B_n correspond to the possible configurations of the machine for inputs of size n. Thus for $s(n) \in \Omega(\log n)$, if the machine computes in space $s(n)$ then B_n has $2^{O(s(n))}$ states.

Many variants of the branching program model have been studied (see in particular the survey by Razborov [Raz91] and the book by Ingo Wegener [Weg00]). Our definition that follows is inspired by Wegener [Weg00, p. 239], by the k-way branching program of Borodin and Cook [BC82] and by its nondeterministic variant [BRS93, GKM08]. We depart from the latter, however, in two ways: nondeterministic branching program labels are attached to states rather than edges (because we think of branching program states as Turing machine configurations) and cycles in branching programs are allowed (because our lower bounds apply to this more general model[2]).

Definition 2.2 **Branching Programs**

A *nondeterministic k-way branching program* B computing a total function $g : [k]^m \to R$, where R is a finite set, is a directed rooted multigraph whose nodes are called *states*. Every edge has a label from $[k]$. Every state has a label from $[m]$, except $|R|$ *final* sink states consecutively labeled with the elements from R. An input $(x_1, \ldots, x_m) \in [k]^m$ activates, for each $1 \leq j \leq m$, every edge labeled x_j out of every

2. A BP with cycles can be simulated by an acyclic BP by at most squaring the number of states. Hence this distinction is not important for separating **L** and **P**.

state labeled j. A *computation* on input $\vec{x} = (x_1, \ldots, x_m) \in [k]^m$ is a directed path consisting of edges activated by $\vec{x}$ which begins with the unique start state (the root), and either it is infinite, or it ends in the final state labeled $g(x_1, \ldots, x_m)$, or it ends in a nonfinal state labeled j with no outedge labeled x_j (in which case we say the computation *aborts*). At least one such computation must end in a final state. The *size* of B is its number of states. B is *deterministic k-way* if every nonfinal state has precisely k outedges labeled $1, \ldots, k$. B is *binary* if $k = 2$.

We say that B solves a decision problem (relation) if it computes the characteristic function of the relation.

A k-way branching program computing the function $FT_d^h(k)$ requires as input k^d many k-ary arguments for each internal node i of T_d^h in order to specify the function f_i, together with one k-ary argument for each leaf. Thus in the notation of Definition 2.1, $FT_d^h(k)$: $[k]^m \to R$ where $R = [k]$ and $m = \frac{d^{h-1}-1}{d-1} \cdot k^d + d^{h-1}$. Also $BT_d^h(k)$: $[k]^m \to \{0,1\}$.

For fixed d, h we are interested in how the number of states required for a k-way branching program to compute $FT_d^h(k)$ and $BT_d^h(k)$ grows with k. We define $\#\mathsf{detFstates}_d^h(k)$ (respectively, $\#\mathsf{ndetFstates}_d^h(k)$) to be the minimum number of states required for a deterministic (respectively, nondeterministic) k-way branching program to solve $FT_d^h(k)$. Similarly we define $\#\mathsf{detBstates}_d^h(k)$ and $\#\mathsf{ndetBstates}_d^h(k)$ to be the number of states for solving $BT_d^h(k)$.

The next lemma shows that the function problem is not much harder to solve than the Boolean problem.

Lemma 2.3

$$\#\mathsf{detBstates}_d^h(k) \le \#\mathsf{detFstates}_d^h(k) \le (k-1) \cdot \#\mathsf{detBstates}_d^h(k)$$
$$\#\mathsf{ndetBstates}_d^h(k) \le \#\mathsf{ndetFstates}_d^h(k) \le (k-1) \cdot \#\mathsf{ndetBstates}_d^h(k)$$

Proof. The left inequalities are obvious. For the others, we can construct a branching program solving the function problem from a sequence of $k-1$ programs solving Boolean problems, where the ith program determines whether the value of the root node is i. ∎

Next we introduce thrifty programs, a restricted form of k-way branching programs for solving tree evaluation problems. Thrifty programs efficiently simulate pebbling algorithms, and implement the best-known upper bounds for $\#\mathsf{ndetBstates}_d^h(k)$ and $\#\mathsf{detFstates}_d^h(k)$, and are within a factor of $\log k$ of the best-known upper bounds for $\#\mathsf{detBstates}_d^h(k)$. In Section 5 we prove tight lower bounds for deterministic thrifty programs which solve $BT_d^h(k)$ and $FT_d^h(k)$.

Definition 2.4 **Thrifty Branching Program**
A deterministic k-way branching program which solves $FT_d^h(k)$ or $BT_d^h(k)$ is *thrifty* if during the computation on any input every query $f_i(\vec{x})$ to an internal node i of T_d^h satisfies the condition that $\vec{x}$ is the tuple of correct values for the children of node i. A nondeterministic such program is *thrifty* if for every input every computation which ends in a final state satisfies the aforesaid restriction on queries.

Note that the restriction in the previous definition is semantic, rather than syntactic. It somewhat resembles the semantic restriction used to define incremental branching programs in Gál et al. [GKM08]. However we are able to prove strong lower bounds using our semantic restriction, but in Gál et al. [GKM08] a syntactic restriction was needed to prove lower bounds.

2.2 One Function Is Enough

It turns out that the complexities of $FT_d^h(k)$ and $BT_d^h(k)$ are not much different if we require all functions assigned to internal nodes to be the same.[3] To denote this restricted version of the problems we replace F by $\hat{F}$ and B by $\hat{B}$. Thus $\hat{F}T_d^h(k)$ is the function problem for T_d^h when all node functions are the same, and $\hat{B}T_d^h(k)$ is the corresponding Boolean problem. To specify an instance of one of these new problems we need only give one copy of the table for the common node function $\hat{f}$, together with the values for the leaves.

Theorem 2.5 *Let $N = (d^h - 1)/(d - 1)$ be a constant denoting the number of nodes in the tree T_d^h. Any Nk-way branching program $\hat{B}$ solving $\hat{F}T_d^h(Nk)$ (respectively, $\hat{B}T_d^h(Nk)$) can be transformed to a k-way branching program B solving $FT_d^h(k)$ (respectively, $BT_d^h(k)$), where B has no more states than $\hat{B}$ and B is deterministic iff $\hat{B}$ is deterministic. Also for each $d \geq 2$ the decision problem $BT_d(h,k)$ is log space reducible to $\hat{B}T_d(h,k)$ (where h, k are input parameters).*

Proof. Given an instance I of $FT_d^h(k)$ (or $BT_d^h(k)$) we can find a corresponding instance $\hat{I}$ of $\hat{F}T_d^h(Nk)$ (or $\hat{B}T_d^h(Nk)$) by coding the set of all functions f_i associated with internal nodes i in I by a single function $\hat{f}$ associated with each node of $\hat{I}$. Here we represent each element of $[Nk]$ by a pair $\langle i, x\rangle$, where $i \in [N]$ represents a node in T_d^h and $x \in [k]$. We want to satisfy the following claim.

Claim *If a node i has a value x in I then node i has value $\langle i, x\rangle$ in $\hat{I}$.*

Thus if i is a leaf node, then we define the leaf value for node i in $\hat{I}$ to be $\langle i, x\rangle$, where x is the value of leaf i in I.

3. We thank Yann Strozecki, who posed this question.

We define the common internal node function $\hat{f}$ as follows. If nodes $i_1, \ldots, i_d$ are the children of node j in T_d^h, then

$$\hat{f}(\langle i_1, x_1\rangle, \ldots, \langle i_d, x_d\rangle) = \langle j, f_j(x_1, \ldots, x_d)\rangle. \tag{2}$$

The value of $\hat{f}$ is irrelevant (make it $\langle 1, 1\rangle$) if nodes $i_1, \ldots, i_d$ are not the children of j.

An easy induction on the height of a node i shows that the preceding claim is satisfied.

Note that the value x of the root node 1 in I is easily determined by the value $\langle 1, x\rangle$ of the root in $\hat{I}$. We specify that the pair $\langle 1, 1\rangle$ has value 1 in $[Nk]$, so I is a YES instance of the decision problem $BT_d^h(k)$ iff $\hat{I}$ is a YES instance of $\hat{B}T_d^h(Nk)$.

To complete the proof of the last sentence in the theorem we note that the number of bits needed to specify I is $\Theta(Nk^d \log k)$, and the number of bits to specify $\hat{I}$ is dominated by the number to specify $\hat{f}$, which is $O((Nk)^d \log(Nk))$. Thus the transformation from I to $\hat{I}$ is length-bounded by a polynomial in length of its argument, and it is not hard to see that it can be carried out in log space.

Now we prove the first part of the theorem. Given an Nk-way BP $\hat{B}$ solving $\hat{B}T_d^h(Nk)$ (respectively, $\hat{F}T_d^h(Nk)$) we can find a corresponding k-way BP B solving $BT_d^h(k)$ (respectively, $FT_d^h(k)$) as follows.

The idea is that on input instance I, B acts like $\hat{B}$ on input $\hat{I}$. Thus for each state $\hat{q}$ in $\hat{B}$ that queries a leaf node i, the corresponding state q in B queries i, and for each possible answer $x \in [k]$, B has an outedge labeled x corresponding to the edge from $\hat{q}$ labeled $\langle i, x\rangle$. If $\hat{q}$ queries $\hat{f}$ at arguments as in (2) (where $i_1, \ldots, i_d$ are the children of node j) then q queries $f_j(x_1, \ldots, x_d)$ and for each $x \in [k]$, q has an outedge labeled x corresponding to the edge from $\hat{q}$ labeled $\langle j, x\rangle$. If $i_1, \ldots, i_d$ are not the children of j, then the node q is not necessary in B, since the answer to the query is always the default $\langle 1, 1\rangle$.

In case $\hat{B}$ is solving the function problem $\hat{F}T_d^h(Nk)$ then each output state labeled $\langle 1, x\rangle$ is relabeled x in B (recall that the root of T_d^h is number 1). Any output state q labeled $\langle i, x\rangle$ where $i > 1$ will never be reached in B (since the value of the root node of $\hat{I}$ always has the form $\langle 1, x\rangle$) so q can be deleted. For any edge in $\hat{B}$ leading to q the corresponding edge in B can lead anywhere. ∎

Similarly to Theorem 2.5 it is easy to see that the problems $FT_d^h(k)$ and $BT_d^h(k)$ can be reduced to the case that the degree $d = 2$ by increasing the height h by a factor of $\lceil \log_2 d \rceil$ and increasing k to $k' = k^{d'}$, where $d' < d$. The idea is to replace each node v in T_d^h by a binary tree T_v of height $\lceil \log_2 d \rceil$ whose first d leaves correspond to the d children of v. The value of the root of T_v is that of v, and the value of a nonroot internal node u of T_v is the tuple of values of the leaves of T_v which are

descendants of u. The function f'_v which corresponds to the node v in the binary tree satisfies (assuming d is a power of 2)

$$f'_v(\langle x_1, \ldots, x_{d/2}\rangle, \langle x_{d/2+1}, \ldots, x_d\rangle) = f_v(x_1, \ldots, x_d).$$

The function associated with a nonroot internal node of T_v just concatenates tuples with appropriate padding with 1's.

One goal of this article is to draw attention to the tree evaluation problem and to encourage further attempts at showing $BT_d(h,k) \notin \mathbf{L}$. By the preceding paragraph this is equivalent to showing $BT_2(h,k) \notin \mathbf{L}$, and by Theorem 2.5 this is equivalent to showing $\hat{B}T_d(h,k) \notin \mathbf{L}$. Further our suggested method is to try proving for each fixed h a lower bound of $\Omega(k^{r(h)})$ on the number of states required for a k-way BP to solve $FT^h_d(k)$, where $r(h)$ is any unbounded function (see Corollary 3.3 to follow later). Again according to Theorem 2.5 (since N is a constant) technically speaking we may as well assume that all the node functions in the instance of $FT^h_d(k)$ are the same. However in practice this assumption is not helpful in provinga lower bound. For example Theorem 5.10 states that k^3 states are required for a deterministic k-way BP to solve $FT^3_2(k)$, and the proof assigns three different functions to the three internal nodes of the binary tree of height 3.

2.3 Pebbling

The pebbling game for DAGs (Directed Acyclic Graphs) was defined by Paterson and Hewitt [PH70] and was used as an abstraction for deterministic Turing machine space in Cook [Coo74]. Black-white pebbling was introduced in Cook and Sethi [CS76] as an abstraction of nondeterministic Turing machine space (see Nordström [Nor09] for a recent survey).

Here we define and use three versions of the pebbling game for DAGs with one root (i.e., one sink node). The first is a simple "black pebbling" game: A black pebble can be placed on any leaf (i.e., source node), and in general if all children of a node i (where a child of i is a node with an edge to i) have pebbles, then one of the pebbles on the children can be slid to i (this is a "black sliding move"). Any black pebble can be removed at any time. The goal is to pebble the root, using as few pebbles as possible.

The second version is "whole" black-white pebbling as defined in Cook and Sethi [CS76] with the restriction that we do not allow "white sliding moves". Thus if node i has a white pebble and each child of i has a pebble (either black or white) then the white pebble can be removed. (A white sliding move would apply if one of the children had no pebble, and the white pebble on i was slid to the empty child. We do not allow this.) A white pebble can be placed on any node at any time. The goal is to start and end with no pebbles, but to have a black pebble on the root at some time.

The third is a new game called *fractional pebbling*, which generalizes whole black-white pebbling by allowing the black and white pebble value of a node to be any real number between 0 and 1. However the total pebble value of each child of a node i must be 1 before the black value of i is increased or the white value of i is decreased. Figure 2 illustrates two configurations in an optimal fractional pebbling of the binary tree of height three using 2.5 pebbles.

Our motivation for choosing these definitions is that we want pebbling algorithms for trees to closely correspond to k-way branching program algorithms for the tree evaluation problem. A black pebble on a node means that the corresponding branching program state knows the value of the node, and a white pebble (applicable to nondeterministic BPs) means that state has a specific conjecture for the value of the node (which must later be verified). A fractional pebble means that the state knows or conjectures that fraction of the $\log k$ bits is the value.

We start by formally defining fractional pebbling, and then define the other two notions as restrictions on fractional pebbling.

Definition 2.6 **Pebbling**

A *fractional pebble configuration* on a rooted d-ary tree T is an assignment of a pair of real numbers $(b(i), w(i))$ to each node i of the tree, where

$$0 \leq b(i), w(i), \tag{3}$$

$$b(i) + w(i) \leq 1. \tag{4}$$

Here $b(i)$ and $w(i)$ are the *black pebble value* and the *white pebble value*, respectively, of i, and $b(i) + w(i)$ is the *pebble value* of i. The number of pebbles in the configuration is the sum over all nodes i of the pebble value of i. The legal pebble moves are as follows (always subject to maintaining the constraints (3), (4)): (i) For any node i, decrease $b(i)$ arbitrarily. (ii) For any node i, increase $w(i)$ arbitrarily. (iii) For every node i, if each child of i has pebble value 1, then decrease $w(i)$ to 0, increase $b(i)$ arbitrarily, and simultaneously decrease the black pebble values of the children of i arbitrarily.

A *fractional pebbling* of T using p pebbles is any sequence of (fractional) pebbling moves on nodes of T which starts and ends with every node having pebble value 0, and at some point the root has black pebble value 1, and no configuration has more than p pebbles.

A *whole black-white pebbling* of T is a fractional pebbling of T such that $b(i)$ and $w(i)$ take values in $\{0, 1\}$ for every node i and every configuration. A *black pebbling* is a black-white pebbling in which $w(i)$ is always 0.

Notice that rule (iii) does not quite treat black and white pebbles dually, since the pebble values of the children must each be 1 before any decrease of $w(i)$ is

allowed. A true dual move would allow increasing the white pebble values of the children so they all have pebble value 1 while simultaneously decreasing $w(i)$. In other words, we allow black sliding moves, but disallow white sliding moves. The reason for this (as mentioned before) is that nondeterministic branching programs can simulate the former, but not the latter. However white sliding moves are a natural dual to black sliding moves and we give a formal definition and examples in Section 4.3.

We use $\#\mathsf{pebbles}(T)$, $\#\mathsf{BWpebbles}(T)$, and $\#\mathsf{FRpebbles}(T)$ respectively to denote the minimum number of pebbles required to black pebble T, black-white pebble T, and fractional pebble T. Bounds for these values are given in Section 4. For example for $d = 2$ we have $\#\mathsf{pebbles}(T_2^h) = h$, $\#\mathsf{BWpebbles}(T_2^h) = \lceil h/2 \rceil + 1$, and $\#\mathsf{FRpebbles}(T_2^h) \leq h/2 + 1$. In particular $\#\mathsf{FRpebbles}(T_2^3) = 2.5$ (see Figure 2).

3 Connecting TMS, BPS, and Pebbling

Let $FT_d(h,k)$ be the same as $FT_d^h(k)$ except now the inputs vary with both h and k, and we assume the input to $FT_d(h,k)$ is a binary string X which codes h and k and codes each node function f_i for the tree T_d^h by a sequence of k^d binary numbers and each leaf value by a binary number in $[k]$, so X has length

$$|X| = \Theta(d^h k^d \log k). \tag{5}$$

The output is a binary number in $[k]$ giving the value of the root.

The problem $BT_d(h,k)$ is the Boolean version of $FT_d(h,k)$: The input is the same, and the instance is true iff the value of the root is 1.

Obviously $BT_d(h,k)$ and $FT_d(h,k)$ can be solved in polynomial time, but we can prove a stronger result.

Theorem 3.1 *The problem $BT_d(h,k)$ is in* **LogDCFL***, even when d is given as an input parameter.*

Proof. By Sudborough [Sud78] it suffices to show that $BT_d(h,k)$ is solved by some deterministic auxiliary pushdown automaton M in log space and polynomial time. The algorithm for M is to use its stack to perform a depth-first search of the tree T_d^h, where for each node i it keeps a partial list of the values of the children of i on its stack, until it obtains all d values, at which point it computes the value of i and pops its stack, adding that value to the list for the parent node.

Note that the length n of an input instance is about $d^h k^d \log k$ bits, so $\log n > d \log k$, so M has ample space on its work tape to write all d values of the children of a node i. ∎

The best-known upper bounds on branching program size for $FT_d^h(k)$ grow as $k^{\Omega(h)}$. The next result shows (Corollary 3.3) that any lower bound with a nontrivial

dependency on h in the exponent of k for deterministic (respectively, nondeterministic) BP size would separate **L** (respectively, **NL**) from **LogDCFL**.

Theorem 3.2 *For each $d \geq 2$, if $BT_d(h,k)$ is in* **L** *(respectively,* **NL***) then there is a constant c_d and a function $f_d(h)$ such that $\#\mathsf{detFstates}_d^h(k) \leq f_d(h)k^{c_d}$ (respectively, $\#\mathsf{ndetFstates}_d^h(k) \leq f_d(h)k^{c_d}$) for all $h,k \geq 2$.*

Proof. By Lemma 2.3 it suffices to prove this for $\#\mathsf{detBstates}_d^h(k)$ and $\#\mathsf{ndetBstates}_d^h(k)$ instead of $\#\mathsf{detFstates}_d^h(k)$ and $\#\mathsf{ndetFstates}_d^h(k)$. In general a Turing machine which can enter at most C different configurations on all inputs of a given length n can be simulated (for inputs of length n) by a binary (and hence k-ary) branching program with C states. Each Turing machine using space $O(\log n)$ has at most n^c possible configurations on any input of length $n \geq 2$, for some constant c. By (5) the input for $BT_d(h,k)$ has length $n = \Theta(d^h k^d \log k)$, so there are at most $(d^h k^d \log k)^{c'}$ possible configurations for a log space Turing machine solving $BT_d(h,k)$, for some constant c'. So we can take $f_d(h) = d^{c'h}$ and $c_d = c'(d+1)$. ∎

Corollary 3.3 *Fix $d \geq 2$ and any unbounded function $r(h)$. If $\#\mathsf{detFstates}_d^h(k) \in \Omega(k^{r(h)})$ then $BT_d(h,k) \notin \boldsymbol{L}$. If $\#\mathsf{ndetFstates}_d^h(k) \in \Omega(k^{r(h)})$ then $BT_d(h,k) \notin$* **NL**.

The next result connects pebbling upper bounds with upper bounds for thrifty branching programs.

Theorem 3.4 *(i) If T_d^h can be black pebbled with p pebbles, then deterministic thrifty branching programs with $O(k^p)$ states can solve $FT_h(k)$ and $BT_d^h(k)$.*

(ii) If T_d^h can be fractionally pebbled with p pebbles then nondeterministic thrifty branching programs can solve $BT_d^h(k)$ with $O(k^p)$ states.

Proof. Consider the sequence $C_0, C_1, \ldots C_\tau$ of pebble configurations for a black pebbling of T_d^h using p pebbles. We may as well assume that the root is pebbled in configuration C_τ, since all pebbles could be removed in one more step at no extra cost in pebbles. We design a thrifty branching program B for solving $FT_d^h(k)$ as follows. For each pebble configuration C_t, program B has k^p states; one state for each possible assignment of a value from $[k]$ to each of the p pebbles. Hence B has $O(k^p)$ states, since τ is a constant independent of k. Consider an input I to $FT_d^h(k)$, and let v_i be the value in $[k]$ which I assigns to node i in T_d^h (see Definition 2.1). We design B so that on I the computation of B will be a state sequence $\alpha_0, \alpha_1, \ldots, \alpha_\tau$, where the state α_t assigns to each pebble the value v_i of the node i that it is on. (If a pebble is not on any node, then its value is 1.)

For the initial pebble configuration no pebbles have been assigned to nodes, so the initial state of B assigns the value 1 to each pebble. In general if B is in a state

α corresponding to configuration C_t, and the next configuration C_{t+1} places a pebble j on node i, then the state α queries the node i to determine v_i, and moves to a new state which assigns v_i to the pebble j and assigns 1 to any pebble which is removed from the tree. Note that if i is an internal node, then all children of i must be pebbled at C_t, so the state α "knows" the values $v_{j_1}, \ldots, v_{j_d}$ of the children of i, so α queries $f_i(v_{j_1}, \ldots, v_{j_d})$.

When the computation of B reaches a state α_τ corresponding to C_τ, then α_τ determines the value of the root (since C_τ has a pebble on the root), so B moves to a final state corresponding to the value of the root.

The argument for the case of whole black-white pebbling is similar, except now the value for each white pebble represents a guess for the value v_i of the node it is on. If the pebbling algorithm places a white pebble j on a node at some step, then the corresponding state of B nondeterministically moves to any state in which the values of all pebbles except j are the same as before, but the value of j can be any value in $[k]$. If the pebbling algorithm removes a white pebble j from a node i, then the corresponding state has a guess v'_i for the value of i, and either i is a leaf, or all children of i must be pebbled. The corresponding state of B queries i to determine its true value v_i. If $v_i \neq v'_i$ then the computation aborts (i.e., all outedges from the state have label v'_i). Otherwise B assigns j the value 1 and continues.

When B reaches a state α corresponding to a pebble configuration C_t for which the root has a black pebble j, then α knows whether or not the tentative value assigned to the root is 1. All future states remember whether the tentative value is 1. If the computation successfully (without aborting) reaches a state α_τ corresponding to the final pebble configuration C_τ, then B moves to the final state corresponding to output 1 or output 0, depending on whether the tentative root value is 1.

Now we consider the case in which $C_0, \ldots, C_\tau$ represents a fractional pebbling computation. If $b(i)$, $w(i)$ are the black and white pebbled values of node i in configuration C_t, then a state α of B corresponding to C_t will remember a fraction $b(i)+w(i)$ of the $\log k$ bits specifying the value v_i of the node i, where the fraction $b(i)$ of bits are verified, and the fraction $w(i)$ of bits are conjectured. In general these numbers of bits are not integers, so they are rounded up to the next integer. This rounding introduces at most two extra bits for each node in T_d^h, for a total of at most $2T$ extra bits, where T is the number of nodes in T_d^h. Since the sum over all nodes of all pebble values is at most p, the total number of bits that need to be remembered for a given pebble configuration is at most $p \log k + 2T$, where T is a constant. Associated with each step in the fractional pebbling there are $2^{p \log k + 2T} = O(k^p)$ states in the branching program, one for each setting of these bits. These bits can

be updated for each of the three possible fractional pebbling moves (i), (ii), (iii) in Definition 2.6 in a manner similar to that for whole black-white pebbling.

It is easy to see that in all cases the branching programs described satisfy the thrifty requirement that an internal node is queried only at the correct values for its children (or, in the black-white and fractional cases, the program aborts if an incorrect query is made because of an incorrect guess for the value of a white-pebbled node). ∎

Corollary 3.5 $\#\mathsf{detFstates}_d^h(k) = O(k^{\#\mathsf{pebbles}(T_d^h)})$ *and* $\#\mathsf{ndetFstates}_d^h(k) = O(k^{\#\mathsf{FRpebbles}(T_d^h)})$.

4 Pebbling Bounds

4.1 Previous Results

We start by summarizing what is known about whole black and black-white pebbling numbers as defined at the end of Definition 2.6 (i.e., we allow black sliding moves but not white sliding moves).

The following are minor adaptations of results and techniques that have been known since work of Loui [Lou79], auf der Heide [adHei79], and Lengauer and Tarjan [LT80] in the late '70s. They considered pebbling games where sliding moves were either disallowed or permitted for both black and white pebbles, in contrast to our results that follow.

We always assume $h \geq 2$ and $d \geq 2$.

Theorem 4.1 $\#\mathsf{pebbles}(T_d^h) = (d-1)h - d + 2$.

Proof. For $h = 2$ this gives $\#\mathsf{pebbles}(T_d^2) = d$, which is obviously correct. In general we show $\#\mathsf{pebbles}(T_d^{h+1}) = \#\mathsf{pebbles}(T_d^h) + d - 1$, from which the theorem follows.

The following pebbling strategy gives the upper bound: Let the root be node 1 and the children be $2 \ldots d+1$. Pebble the nodes $2 \ldots d+1$ in order using the optimal number of pebbles for T_d^{h-1}, leaving a black pebble at each node. Note that for the black pebble game, the complexity of pebbling in the game where a pebble remains on the root is the same as for the game where the root has a black pebble on it at some point. The maximum number of pebbles at any point on the tree is $d - 1 + \#\mathsf{pebbles}(T_d^{h-1})$. Now slide the black pebble from node 1 to the root, and then remove all pebbles.

For the lower bound, consider the time t at which the children of the root all have black pebbles on them. There must be a final time t' before t at which one of the sub-trees rooted at $2, 3, \ldots, d+1$ had $\#\mathsf{pebbles}(T_d^h)$ pebbles on it. This is because pebbling any of these subtrees requires at least $\#\mathsf{pebbles}(T_d^h)$ pebbles, by definition. At time t', all the other subtrees must have at least 1 black pebble each

on them. If not, then there is a subtree T which does not, and it would have to be pebbled before time t, which contradicts the definition of t'. Thus at timet', there are at least $\#\mathsf{pebbles}(T_d^h) + d - 1$ pebbles on the tree. ∎

Theorem 4.2 *For $d = 2$ and d odd:*

$$\#\mathsf{BWpebbles}(T_d^h) = \lceil (d-1)h/2 \rceil + 1. \tag{6}$$

For d even:

$$\#\mathsf{BWpebbles}(T_d^h) \leq \lceil (d-1)h/2 \rceil + 1. \tag{7}$$

When d is odd, this number is the same as when white sliding moves are allowed.

Proof. We divide the proof into three parts: Part I proves (6) when d is odd, Part II proves (7) when d is even (which implies the upper bound for (6) when $d = 2$), and Part III proves the lower bound in (6) when $d = 2$.

Part I. We show (6) when d is odd.

For $h = 2$ this gives $\#\mathsf{BWpebbles}(T_d^2) = d$, which is obviously correct. In general for odd d we show

$$\#\mathsf{BWpebbles}(T_d^{h+1}) = \#\mathsf{BWpebbles}(T_d^h) + (d-1)/2 \tag{8}$$

from which the theorem follows for this case.

For the upper bound for the left-hand side, we strengthen the induction hypothesis by asserting that during the pebbling there is a *critical time* at which the root has a black pebble and there are at most $\#\mathsf{BWpebbles}(T_d^h) - (d-1)/2$ pebbles on the tree (counting the pebble on the root). This can be made true when $h = 2$ by removing all the pebbles on the leaves after the root is pebbled.

To pebble the tree T_d^{h+1}, note that we are allowed $(d-1)/2$ extra pebbles over those required to pebble T_d^h. Start by placing black pebbles on the left-most $(d-1)/2$ children of the root, and removing all other pebbles. Now go through the procedure for pebbling the middle principal subtree, stopping at the critical time, so that there is a black pebble on the middle child of the root and at most $\#\mathsf{BWpebbles}(T_d^h) - (d-1)/2$ pebbles on the middle subtree. Now place white pebbles on the remaining $(d-1)/2$ children of the root, slide a black pebble to the root, and remove all black pebbles on the children of the root. This is the critical time for pebbling T_d^{h+1}: Note that there are at most $\#\mathsf{BWpebbles}(T_d^h)$ pebbles on the tree (we removed the black pebble on the root of the middle subtree).

Now remove the pebble on the root and remove all pebbles on the middle subtree by completing its pebbling (keeping the $(d-1)/2$ white pebbles on the children

in place). Finally remove the remaining $(d-1)/2$ white pebbles one by one, simply by pebbling each subtree, and removing the white pebble at the root of the subtree instead of black-pebbling it.

To prove the lower bound for the left-hand side of (8), we strengthen the induction hypothesis so that now a black-white pebbling allows white sliding moves, and the root may be pebbled by either a black pebble or a white pebble. (Note that for the base case the tree T_d^2 still requires d pebbles.) Consider such a pebbling of T_d^{h+1} which uses as few moves as possible. Consider a time t at which all children of the root have pebbles on them (i.e., just before the root is black pebbled or just after a white pebble on the root is removed). For each child i, let t_i be a time at which the tree rooted at i has $\#\mathsf{BWpebbles}(T_d^h)$ pebbles on it. We may assume

$$t_2 < t_3 < \ldots < t_{d+1}.$$

Let $m = (d+3)/2$ be the middle child. If $t_m < t$ then each of the $(d-1)/2$ subtrees rooted at i for $i < m$ has at least one pebble on it at time t_m, since otherwise the effort made to place $\#\mathsf{BWpebbles}(T_d^h)$ pebbles on it earlier is wasted. Hence (8) holds for this case. Similarly if $t_m > t$ then each of the $(d-1)/2$ subtrees rooted at i for $i > m$ has at least one pebble on it at time t_m, since otherwise the effort to place T_d^h pebbles on it later is wasted, so again (8) holds.

Part II. We prove (7) for even degree d.

$$\#\mathsf{BWpebbles}(T_d^h) \leq \lceil (d-1)h/2 \rceil + 1$$

For $h = 2$ the formula gives $\#\mathsf{BWpebbles}(T_d^2) = d$, which is obviously correct. For $h = 3$ the formula gives $\#\mathsf{BWpebbles}(T_d^3) = (3/2)d$, which can be realized by black-pebbling $d/2 + 1$ of the root's children and white-pebbling the rest. In general it suffices to prove the following recurrence.

$$\#\mathsf{BWpebbles}(T_d^{h+2}) \leq \#\mathsf{BWpebbles}(T_d^h) + d - 1 \tag{9}$$

We strengthen the induction hypothesis by asserting that during the pebbling T_d^h of there is a *critical time* at which the root has a black pebble and there are at most $\#\mathsf{BWpebbles}(T_d^h) - (d-1)$ pebbles on the tree (counting the pebble on the root). This is easy to see when $h = 2$ and $h = 3$.

We prove the recurrence as follows. We want to pebble T_d^{h+2} using $d-1$ more pebbles than is required to pebble T_d^h. Let us call the children of the root $c_1, \ldots, c_d$. We start by placing black pebbles on $c_1, \ldots, c_{d/2}$. We illustrate how to do this by showing how to place a black pebble on $c_{d/2}$ after there are black pebbles on nodes

$c_1, \ldots, c_{d/2-1}$. At this point we still have $d/2$ extra pebbles left among the original $d-1$. Let us assign the names $c'_1, \ldots, c'_d$ to the children of $c_{d/2}$. Use the $d/2$ extra pebbles to put black pebbles on $c'_1, \ldots, c'_{d/2}$. Now run the procedure for pebbling the subtree rooted at $c'_{d/2+1}$ up to the critical time, so there is a black pebble on $c'_{d/2+1}$. Now place white pebbles on the remaining $d/2-1$ children of $c_{d/2}$, slide a black pebble up to $c_{d/2}$, remove the remaining black pebbles on the children of $c_{d/2}$, and complete the pebbling procedure for the subtree rooted at $c'_{d/2+1}$, so that subtree has no pebbles. Now remove the white pebbles on the remaining $d/2-1$ children of $c_{d/2}$.

At this point there are black pebbles on nodes $c_1, \ldots, c_{d/2}$, and no other pebbles on the tree. We now place a black pebble on $c_{d/2+1}$ as follows. Let us assign the names $c''_1, \ldots, c''_d$ to the children of $c_{d/2+1}$. Use the remaining $d/2-1$ extra pebbles to place black pebbles on $c''_1, \ldots, c''_{d/2-1}$. Now run the pebble procedure on the subtree rooted at $c''_{d/2}$ up to the critical time, so $c''_{d/2}$ has a black pebble. Now place white pebbles on the remaining $d/2$ children of $c_{d/2+1}$, slide a black pebble up to $c_{d/2+1}$, remove the remaining black pebbles on the children of $c_{d/2+1}$, place white pebbles on the remaining $d/2-1$ children of the root, slide a black pebble up to the root, and remove the remaining black pebbles from the children of the root.

This is now the critical time for the procedure pebbling T_d^{h+2}. There is a black pebble on the root, $d/2-1$ white pebbles on the children of the root, $d/2$ white pebbles on the children of $c_{d/2+1}$, and at most $\#\mathsf{BWpebbles}(T_d^h) - d$ pebbles on the subtree rooted at $c''_{d/2}$ (we've removed the black pebble on $c''_{d/2}$), making a total of at most $\#\mathsf{BWpebbles}(T_d^h)$ pebbles on the tree.

Now remove the black pebble from the root and complete the pebble procedure for the subtree rooted at $c''_{d/2}$ to remove all pebbles from that subtree. There remain $d/2-1$ white pebbles on the children of the root and $d/2$ white pebbles on the children of $c_{d/2+1}$, making a total of $d-1$ white pebbles. Now remove each of the white pebbles on the children of $c_{d/2+1}$ by pebbling each of these subtrees in turn. Finally we can remove each of the remaining $d/2-1$ white pebbles on the children of the root by a process similar to the one used to place $d/2$ black pebbles on the children of the root at the beginning of the procedure (we now in effect have one more pebble to work with).

Part III. Finally we give the lower bound for the case $d = 2$.

$$\#\mathsf{BWpebbles}(T_2^h) \geq \lceil h/2 \rceil + 1$$

Clearly 2 pebbles are required for the tree of height 2, and it is easy to show that 3 pebbles are required for the height 3 tree.

In general it suffices to show that the binary tree T of height $h+2$ requires at least one more pebble than the binary tree of height h. Suppose otherwise, and consider a pebbling of T that uses the minimum number of pebbles required for the tree of height h, and assume that the pebbling is as short as possible. Let t_1 be a time when the root has a black pebble. For $i = 3, 4, 5$ there must be a time t_i when all the pebbles are on the subtree rooted at node i. This is because node i must be pebbled at some point, and if the pebble is white then right after the white pebble is removed we could have placed a black pebble in its place (since we do not allow white sliding moves).

Suppose that $\{t_1, t_3, t_4, t_5\}$ are ordered such that

$$t_{i_1} < t_{i_2} < t_{i_3} < t_{i_4}.$$

Then t_1 cannot be either t_{i_3} or t_{i_4} since otherwise at time t_{i_2} there are no pebbles on the subtree rooted at node i_1 and hence its earlier pebbling was wasted (since the root has yet to be pebbled). Similarly if t_1 is either t_{i_1} or t_{i_2} then at time t_{i_3} there are no pebbles on the subtree rooted at i_4, and since the root has already been pebbled the later pebbling of this subtree is wasted. ∎

4.2 Results for Fractional Pebbling

The concept of fractional pebbling is new. Determining the minimum number p of pebbles required to fractionally pebble T_d^h is important since $O(k^p)$ is the best-known upper bound on the number of states required by a nondeterministic BP to solve $FT_d^h(k)$ (see Theorem 3.4). It turns out that proving fractional pebbling lower bounds is much more difficult than proving whole black-white pebbling lower bounds. We are able to get exact fractional pebbling numbers for the binary tree of height 4 and less, but the best general lower bound comes from a nontrivial reduction to a paper by Klawe [Kla85] which proves bounds for the pyramid graph. This bound is within $d/2 + 1$ pebbles of optimal for degree d trees (at most 2 pebbles from optimal for binary trees).

Our proof of the exact value of $\#\mathsf{FRpebbles}(T_2^4) = 3$ led us to conjecture that any nondeterministic BP computing $BT_2^4(k)$ requires $\Omega(k^3)$ states. In Section 5 we provide evidence for that conjecture by proving that any nondeterministic *thrifty* BP requires $O(k^3)$ states. The lower bound for height 3 and any degree follows from the lower bound of $\Omega(k^{\frac{3}{2}d-\frac{1}{2}})$ states for nondeterministic branching programs computing $BT_d^3(k)$ (Corollary 5.2).

We start by presenting a general result showing that fractional pebbling can save at most a factor of two over whole black-white pebbling for any DAG (Directed Acyclic Graph). (Here the pebbling rules for a DAG are the same as for a tree, where

we require that every sink node (i.e., every "root") must have a whole black pebble at some point.) We will not use this result, but it does provide a simple proof of weaker lower bounds than those given in Theorem 4.4 that follows shortly.

Theorem 4.3 *If a DAG D has a fractional pebbling using p pebbles, then it has a black-white pebbling using at most $2p$ pebbles.*

Proof. Given a sequence P of fractional pebbling moves for a DAG D in which at most p pebbles are used, we define a corresponding sequence P' of pebbling moves in which at most $2p$ pebbles are used. The sequence P' satisfies the following invariant with respect to P.

(♠) A node v has a black pebble (respectively, a white pebble) on it at time t with respect to P' iff $b(v) \geq 1/2$ (respectively, $w(v) > 1/2$) at time t with respect to P.

An important consequence of this invariant is that if at time t in P node v satisfies $b(v) + w(v) = 1$ then at time t in P' node v is pebbled.

We describe when a pebble is placed or removed in P'. At the beginning, there are no pebbles on any nodes. P' simulates P as follows. Assume there is a certain configuration of pebbles on D, placed according to P' after time $t-1$; we describe how P's move at time t is reflected in P'. If in the current move of P, $b(v)$ (respectively, $w(v)$) increases to $1/2$ or greater (respectively, greater than $1/2$) for some node v, then the current pebble, if any, on v, is removed and a black pebble (respectively, a white pebble) is placed on v in P'. Note that this is always consistent with the pebbling rules. If in the current configuration of P' there is a black (respectively, white) pebble on a vertex v, and in the current move of P, $b(v)$ (respectively, $w(v)$) falls below $1/2$, then the pebble on v is removed. Again, this is always consistent with the pebbling rules for the black-white pebble game and the fractional black-white pebble game. For all other kinds of moves of P, the configuration in P' does not change.

If P is a valid sequence of fractional pebbling moves, then P' is a valid sequence of pebbling moves. We argue that the cost of P' is at most twice the cost of P, and that if there is a point at which the root has black pebble value 1 with respect to P, then there is a point at which the root is black-pebbled in P'. These facts together establish the theorem.

To demonstrate these facts, we simply observe that the invariant (♠) holds by induction on the time t for the simulation we defined. This implies that at any point t, the number of pebbles on D with respect to P' is at most the number of nodes v for which $b(v)+w(v) \geq 1/2$ with respect to P, and is therefore at most twice the total value of pebbles with respect to P at time t. Hence the cost of pebbling D using P' is at most twice the cost of pebbling D using P. Also, if there is a time t at

which the root r has black pebble value 1 with respect to P, then $b(r) \geq 1/2$ at time t, so there is a black pebble on r with respect to P' at time t. ■

The next result presents our best-known bounds for fractionally pebbling trees T_d^h.

Theorem 4.4

$$(d-1)h/2 - d/2 \leq \#\mathsf{FRpebbles}(T_d^h) \leq (d-1)h/2 + 1$$
$$\#\mathsf{FRpebbles}(T_d^3) = (3/2)d - 1/2$$
$$\#\mathsf{FRpebbles}(T_2^4) = 3$$

We divide the proof into several parts. First we prove the upper bound.

$$\#\mathsf{FRpebbles}(T_d^h) \leq (d-1)h/2 + 1$$

Proof. Let A_h be the algorithm for height $h \geq 2$. It is composed of two parts, B_h and C_h. B_h is run on the empty tree, and finishes with a black pebble on the root and $(d-1)(h-2)$ white half pebbles below the root (and of these $(d-1)(h-3)$ lie below the rightmost child of the root). Next, the black pebble on the root is removed. Then C_h is run on the result, and finishes with the empty tree. B_h and C_h both use $(d-1)h/2 + 1$ pebbles.

A'_h is the same as A_h except that it finishes with a black half pebble on the root. It does this in the most straight-forward way, by leaving a black half pebble after the root is pebbled, and so it uses $(d-1)h/2 + 1.5$ pebbles for all $h \geq 3$.

B_2: Pebble the tree of height 2 using d black pebbles.

B_h, $h > 2$: Run A'_{h-1} on node 2 using $(d-1)(h-1)/2 + 1.5$ pebbles, and then on node 3 (if $3 \leq d$) using a total of $(d-1)(h-1)/2 + 2$ pebbles (counting the half pebble on node 2), and so on for nodes $2, 3 \ldots, d$. So $(d-1)(h-1)/2+1+(d-1)/2 = (d-1)h/2 + 1$ pebbles are used when A'_{h-1} is run on node d. Next run B_{h-1} on node $d+1$, using $(d-1)(h-1)/2+1$ pebbles on the subtree rooted at $d+1$, for $(d-1)h/2+1$ pebbles in total (counting the black half pebbles on node $2, \ldots, d$). The result is a black pebble on node $d+1$, $(d-1)(h-3)$ white half pebbles under $d+1$, and from earlier $d-1$ black half pebbles on $2, \ldots, d$, for a total of $(d-1)(h-2)/2 + 1$ pebbles. Add a white half pebble to each of $2, \ldots, d$, then slide the black pebble from $d+1$ onto the root. Remove the black half pebbles from $2, \ldots, d$. Now there are $(d-1)(h-2)$ white half pebbles under the root, and a black pebble on the root.

C_2: The tree of height 2 is empty, so return.

C_h: The tree has no black pebbles and $(d-1)(h-2)$ white half pebbles. Note that if a sequence can pebble a tree with p pebbles, then essentially the same sequence can be used to remove a white half pebble from the root with $p + .5$ pebbles. C_h

runs C_{h-1} on node $d+1$, resulting in a tree with only a half white pebble on each of $2,\ldots,d$. This takes $(d-1)h/2+1$ pebbles. Then A_{h-1} is run on each of $2,\ldots,d$ in turn, to remove the white half pebbles. The first such call of A_{h-1} is the most expensive, using $(d-1)(h-1)/2+1+(d-1)/2=(d-1)h/2+1$ pebbles. ∎

As noted earlier, the tight lower bound for height 3 and any degree:

$$\#\mathsf{FRpebbles}(T_d^3) \geq (3/2)d - 1/2$$

follows from the asymptotically tight lower bound of $\Omega(k^{\frac{3}{2}d-\frac{1}{2}})$ states for nondeterministic branching programs computing $BT_d^3(k)$ (Corollary 5.2). We do, however, have a direct proof of $\#\mathsf{FRpebbles}(T_2^3) \geq 5/2$.

Proof. Assume to the contrary that there is a fractional pebbling with fewer than 2.5 pebbles. It follows that no nonleaf node i can ever have $w(i) \geq 0.5$, since the children of i must each have pebble value 1 in order to decrease $w(i)$. Since there must be some time t_1 during the pebbling sequence such that both the nodes 2 and 3 (the two children of the root) have pebble value 1, it follows that at time t_1, $b(2) > 0.5$ and $b(3) > 0.5$. Hence for $i = 2,3$ there is a largest $t_i \leq t_1$ such that node i is black-pebbled at time t_i and $b(i) > 0.5$ during the time interval $[t_i, t_1]$. (By "black-pebbled" we mean at time $t_i - 1$ both children of i have pebble value 1, so that at time t_i the value of $b(i)$ can be increased.)

Assume without loss of generality that $t_2 < t_3$. Then at time $t_3 - 1$ both children of node 3 have pebble value 1 and $b(2) > 0.5$, so the total pebble value exceeds 2.5. ∎

Before we prove the lower bound for all heights, which we do not believe is tight, we prove one more tight lower bound.

$$\#\mathsf{FRpebbles}(T_2^4) \geq 3$$

Proof. Let $C_0, C_1, \ldots, C_m$ be the sequence of pebble configurations in a fractional pebbling of the binary tree of height 4. We say that C_t is the configuration at time t. Thus C_0 and C_m have no pebbles, and there is a first time t_1 such that C_{t_1+1} has a black pebble on the root. In general we say that step t in the pebbling is the move form C_t to C_{t+1}. In particular, if an internal node i is black-pebbled at step t then both children of i have pebble value 1 in C_t and node i has a positive black pebble value in C_{t+1}.

Note that if any configuration C_t has a whole white pebble on some internal node then both children must have pebble value 1 to remove that pebble, so some configuration will have at least pebble value 3, which is what we are to prove. Hence

we may assume that no node in any C_t has white pebble value 1, and hence every node must be black-pebbled at some step.

For each node i we associate a critical time t_i such that i is black-pebbled at step t_i and hence the children of i each have pebble value 1 in configuration C_{t_i}. The time t_1 associated with the root (as earlier) is the first step at which the root is black-pebbled, and hence nodes 2 and 3 each have pebble value 1 in C_{t_1}. In general if t_i is the critical time for internal node i, and j is a child of i, then the critical time t_j for j is the largest $t < t_i$ such that j is black-pebbled at step t.

Sibling Assumption We may assume without loss of generality (by applying an isomorphism to the tree) that if i and j are siblings and $i < j$ then $t_i < t_j$.

In general the critical times for a path from root to leaf form a descending chain. In particular

$$t_7 < t_3 < t_1.$$

For each $i > 1$ we define b_i and w_i to be the black and white pebble values of node i at the critical time of its parent. Thus for all $i > 1$

$$b_i + w_i = 1. \tag{10}$$

Now let p be the maximum pebble value of any configuration C_t in the pebbling. Our task is to prove that $p \geq 3$.

After the critical time of an internal node i the white pebble values of its two children must be removed. When the first one is removed both white values are present along with pebble value 1 on two children, so

$$w_{2i} + w_{2i+1} + 2 \leq p.$$

In particular for $i = 1, 3$ we have

$$w_2 + w_3 + 2 \leq p, \tag{11}$$
$$w_6 + w_7 + 2 \leq p. \tag{12}$$

Now we consider two cases, depending on the order of t_2 and t_7.

Case I. $t_2 < t_7$. Then by the Sibling Assumption, at time t_7 (when node 7 is black-pebbled) we have

$$b_2 + b_6 + 2 \leq p. \tag{13}$$

Now if we also suppose that w_6 is not removed until after t_1 (CASE IA) then when the first of w_2, w_6 is removed we have

$$w_2 + w_6 + 2 \leq p$$

so adding this equation with (13) and using (10) we see that $p \geq 3$ as required.

However if we suppose that w_6 is removed before t_1 (CASE IB) (but necessarily after $t_2 < t_3$) then we have

$$b_2 + b_3 + w_6 + 2 \leq p.$$

then we can add this to (11) to again obtain $p \geq 3$.

Case II. $t_7 < t_2$. Then $t_6 < t_7 < t_2 < t_3$ so at time t_2 we have

$$b_6 + b_7 + 2 \leq p$$

so adding this to (12) we again obtain $p \geq 3$. ∎

To prove the general lower bound, we need the following lemma.

Lemma 4.5 *For every finite DAG there is an optimal fractional pebbling in which all pebble values are rational numbers. (This result is robust independent of various definitions of pebbling; for example with or without sliding moves, and whether or not we require the root to end up pebbled.)*

Proof. Consider an optimal fractional pebbling algorithm. Let the variables $b_{v,t}$ and $w_{v,t}$ stand for the black and white pebble values of node v at step t of the algorithm.

Claim *We can define a set of linear inequalities with 0-1 coefficients which suffice to ensure that the pebbling is legal.*

For example, all variables are nonnegative, $b_{v,t} + w_{b,t} \leq 1$, initially all variables are 0, and finally the nodes have the values that we want, node values remain the same on steps in which nothing is added or subtracted, and if the black value of a node is increased at a step then all its children must be 1 in the previous step, etc.

Now let p be a new variable representing the maximum pebble value of the algorithm. We add an inequality for each step t that says the sum of all pebble values at step t is at most p.

Any solution to the linear programming problem: Minimize p subject to all of the preceding inequalities gives an optimal pebbling algorithm for the graph. But every LP program with rational coefficients has a rational optimal solution (if it has any optimal solution). ∎

Now we can prove the lower bound for all heights.

$$\#\mathsf{FRpebbles}(T_d^h) \geq (d-1)(h-1)/2 - 1/2 = (d-1)h/2 - d/2 \tag{14}$$

Remark 4.6 We conjecture that the upper bound given in Theorem 4.4 is tight. It seems like proving this should not be much harder than proving the lower bound for black-white pebbling T_d^h. However we have not even been able to prove the weaker lower bound (14) directly. The present proof derives the lower bound from Klawe's result (Theorem 4.8).

Proof. The degree d and height h of the tree are fixed throughout this proof, so we will write just T instead of T_d^h.

The high-level strategy for the proof is as follows. We transform T into a DAG G such that a lower bound for $\#\mathsf{BWpebbles}(G)$ gives a lower bound for $\#\mathsf{FRpebbles}(T)$. To analyze $\#\mathsf{BWpebbles}(G)$, we use a result by Klawe [Kla85], who shows that for any DAG H that satisfies a certain "niceness" property (Definition 4.12), $\#\mathsf{BWpebbles}(H)$ can be given in terms of $\#\mathsf{pebbles}(H)$ (and the relationship is tight to within an additive constant less than one). This helps since the black pebbling cost is typically easier to analyze. In our case, G does not satisfy the niceness property as-is, but just by removing some edges from G (which cannot increase the black or black/white pebbling cost), we get a new DAG G' that is nice. We then compute $\#\mathsf{pebbles}(G')$ exactly, which by Klawe's result yields a lower bound on $\#\mathsf{BWpebbles}(G') \leq \#\mathsf{BWpebbles}(G)$, and hence on $\#\mathsf{FRpebbles}(T)$.

We first motivate the construction G and show that the whole black-white pebbling number of G is related to the fractional pebbling number of T.

We will use Lemma 4.5 to show that the following "discretized" fractional pebbling cost is almost the same as the fractional pebbling cost $\#\mathsf{FRpebbles}$ when the parameter c is large enough.

Definition 4.7 **Discretized Fractional Pebbling**
For positive integer c, let $\#\mathsf{FRpebbles}_c(H)$ be the cost of fractionally pebbling H when only the following moves are allowed.

- For any node v, decrease $b(v)$ or increase $w(v)$ by $1/c$.
- For any node v, including leaf nodes, if all the children of v have value 1, then increase $b(v)$ or decrease $w(v)$ by $1/c$.

By Lemma 4.5, we can assume all pebble values are rational, and if we choose c large enough it is not a restriction that pebble values can only be changed by $1/c$. Sliding moves are not allowed in the discretized game, but it is easy to see that

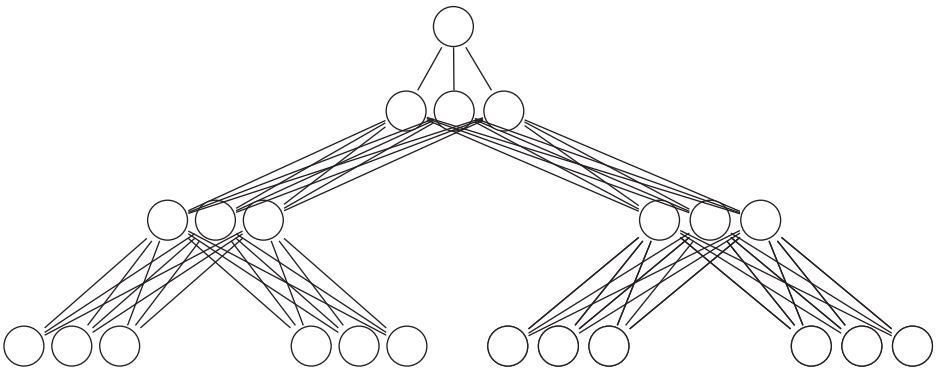


Figure 3 G for the height 3 binary tree with $c = 3$.

increases the cost by at most 1 (compared to fractional pebbling with black sliding moves). Hence we have the following fact.

Fact 2 $\#\mathsf{FRpebbles}(T) \geq \#\mathsf{FRpebbles}_c(T) - 1$ for sufficiently large c.

Now let c be an arbitrary positive integer. We show how to construct $G = G_c$.[4] We will split up each node of T into c nodes, so that the discretized fractional pebbling game on T corresponds to the whole black-white pebbling game on G.[5] Specifically, the cost of the whole black-white pebble game on the new graph will be exactly c times the cost of the discretized game on T.

The idea is to use c whole-pebble-taking nodes to "simulate" each fractional-pebble-taking node of T. For example, if $c = 20$ and we have a configuration of T where node u has black value $4/20$ and white value $6/20$, then in the corresponding configuration of G, 4 (respectively, 6) of the 20 nodes dedicated to simulating u are whole black (respectively, white) pebbled. More precisely, in place of each node v of T, G has c nodes $v[1], \ldots, v[c]$; having any $c' \leq c$ of those nodes pebbled simulates v having value c'/c in the discretized fractional pebbling game. In place of each edge (u, v) of T is a copy of the complete bipartite graph (U, V), where U contains nodes $u[1] \ldots u[c]$ and V contains nodes $v[1] \ldots v[c]$. To clarify: if u is a parent of v in the tree, then all the edges go from V to U in the corresponding complete bipartite graph. Finally, a new "root" is added at height $h + 1$ with edges from each of the c nodes at height h.[6]

4. We don't write the subscript since c is fixed throughout the argument.

5. For an example, see Figure 3.

6. The reason for this is quite technical: Klawe's definition of pebbling is slightly different from ours in that it requires that the root remain pebbled. Adding a new root forces there to be a time

So every node in G at height $h-1$ and lower has c parents, and every internal (i.e., nonleaf) node except for the root has dc children. By construction we get:[7]

Fact 3 $\#\mathsf{FRpebbles}_c(T) \geq (\#\mathsf{BWpebbles}(G)-1)/c$.

From Facts 2 and 3, our goal follows if we can show

$$\#\mathsf{BWpebbles}(G) \geq c((d-1)(h-1)+1)/2+1.$$

For that we will use Theorem 4.8 (from Klawe [Kla85]), stated next. The statement of the theorem depends on Klawe's definition of *nice* DAGs (Definition 4.12), which is stated later when we finally get around to proving that a DAG is nice (Proposition 4.12.1).

Theorem 4.8 [Kla85]
If H is a nice DAG, then

$$\#\mathsf{BWpebbles}(H) \geq \lfloor \#\mathsf{pebbles}(H)/2 \rfloor + 1.$$

G is not nice in Klawe's sense. We will delete some edges from G to produce a nice DAG G' and then we will analyze $\#\mathsf{pebbles}(G')$. Note that deleting edges cannot increase the black-white pebbling cost, and so we have the next fact.

Fact 4 $\#\mathsf{BWpebbles}(G') \leq \#\mathsf{BWpebbles}(G)$.

The following definition will help in explaining the construction of G' as well as for specifying and proving properties of certain paths.

Definition 4.9 For $u \in G$, let $T(u)$ be the node in T from which u was generated (i.e., $T(u)[i] = u$ for some $i \leq c$). For $v, v' \in T$, we say $v <_T v'$ if v is visited before v' in an inorder traversal of T. For $u, u' \in G$, we say $u <_G u'$ if $T(u) <_T T(u')$ or if for some $v \in T$, $i < j \leq c$ have $u = v[i]$, $u' = v[j]$.

G' is obtained from G by removing $c-1$ edges from each internal node except the root, as follows (for an example, see Figure 4). For each internal node v of T, consider the corresponding nodes $v[1], v[2], \ldots, v[c]$ of G. Remove the edges from $v[i]$ to its $i-1$ smallest and $c-i$ largest children. So in the end each internal node of G' except the root has $c(d-1)+1$ children.

By Theorem 4.8 (with $H = G'$) and Fact 4, it now remains to lower bound $\#\mathsf{pebbles}(G')$ (Proposition 4.9.1 with c even), and then show that G' is nice (Proposition 4.12.1).

---

when all c of the height h nodes, which represent the root of T, are pebbled. This affects the relationship between $\#\mathsf{FRpebbles}_c(T)$ and $\#\mathsf{BWpebbles}(G)$ very slightly, as indicated by Fact 3.

7. For clarification, we note that $\#\mathsf{BWpebbles}(G)/c \geq \#\mathsf{FRpebbles}_c(T)$.

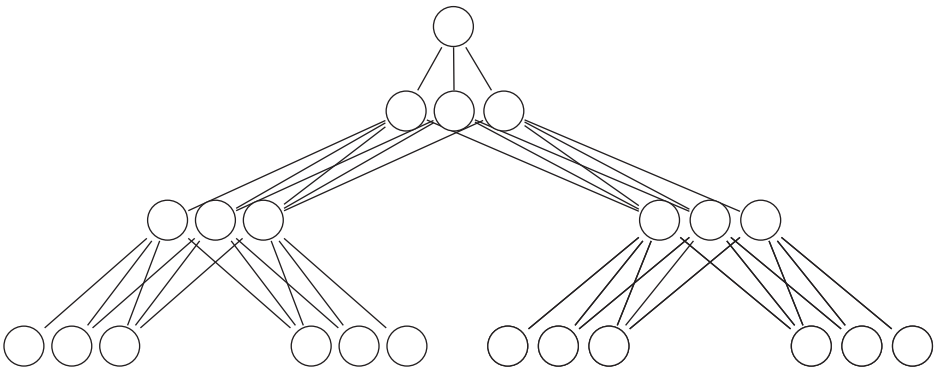


Figure 4 G' for the height 3 binary tree with $c = 3$.

Proposition 4.9.1 $\#pebbles(G') = c((d-1)(h-1)+1)$.

It will be convenient to rewrite the expression as $(c-1)+c(d-1)(h-1)+1$. The upper bound is attained using a simple recursive algorithm similar to that used for T_d^h (Theorem 4.1).

For the lower bound, consider the earliest time t when all paths from a leaf to the root are blocked (a path is blocked if at least one of its nodes is pebbled). Figure 5 is an example of the type of pebbling configuration that we are about to analyze. The last pebble placed must have been placed at a leaf, since otherwise $t-1$ would be an earlier time when all paths from a leaf to the root are blocked. Let P_{BN} be a newly blocked path from a leaf to the root (the *BottleNeck* path). Consider the set

$$S = \{u \in G' \mid u \text{ is a child of a node in } P_{\text{BN}} \text{ and } u \notin P_{\text{BN}}\}$$

of size $(c-1)+c(d-1)(h-1)$.[8]

Claim 1 *There is a set of pairwise node-disjoint paths $\{P_u\}_{u \in S}$ such that for every $u \in S$, P_u is a path from a leaf to u and P_u does not intersect P_{BN}.*

Assuming Claim 1 holds, we get that at time $t-1$, for every $u \in S$ there must be at least one pebble on P_u, since otherwise there would still be an open path from a leaf to the root at time t. Also counting the leaf node that is pebbled at time t gives $(c-1)+c(d-1)(h-1)+1$ pebbles. Hence, to complete the proof of Proposition 4.9.1, it just remains to prove Claim 1, and for that task we will benefit from a couple more simple definitions:

8. S has $c-1$ nodes at height h and $c(d-1)$ nodes at heights $1, \ldots, h-1$. See Figure 5, where the pebbled nodes are the nodes in S plus one leaf node not in S.

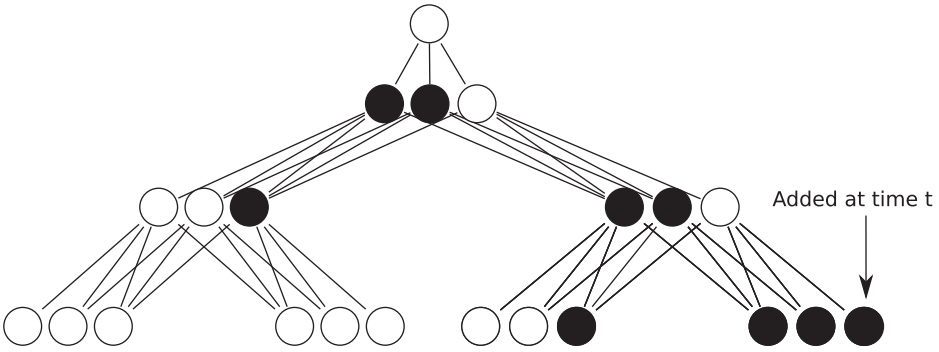


Figure 5 A possible black pebbling bottleneck of G' for the height 3 binary tree with $c = 3$.

Definition 4.10 For $u \in G'$, the left-most (respectively, right-most) path to u is the unique path from some leaf to u that is determined, starting at u, by moving to the smallest (respectively, largest) child at every level.[9]

Definition 4.11 For any path P from a leaf to a height l node u, for $l' \leq l$ let $P(l')$ be the height l' node on P.

Recall the ordering on the nodes of G (which we extend to G') from Definition 4.9. For each $u \in S$ at height l, if u is less than (respectively, greater than) $P_{\mathrm{BN}}(l)$ then make P_u the left-most (respectively, right-most) path to u. Intuitively, we are choosing P_u so that it moves away from P_{BN} as quickly as possible. Now we need to show that the paths $\{P_u\}_{u \in S} \cup \{P_{\mathrm{BN}}\}$ are pairwise node-disjoint. The following fact is clear from the definition of G'.

Fact 5 For any $u, v \in G'$, if $u < v$ then the smallest child of u is not a child of v, and the largest child of v is not a child of u.

First we show that P_u and P_{BN} are node-disjoint for every $u \in S$. The following lemma will help now and in the proof of Proposition 4.12.1.

Lemma 4.11.1 *For $u, v \in G'$ with $u < v$, if there is no path from u to v or from v to u, then the left-most path to u does not intersect any path to v from a leaf, and the right-most path to v does not intersect any path to u from a leaf.*

Proof. Suppose otherwise and let P'_u be the left-most path to u, and P'_v a path to v that intersects P'_u. Since there is no path between u and v, there is a height l, one

9. Equivalently: if u is at height l then the left-most (respectively, right-most) path to u is the length l path $u_1, \ldots, u_l$ such that $u_l = u$ and u_i is the smallest (respectively, largest) child of u_{i+1} for all $i \in \{1, \ldots, l-1\}$.

greater than the height where the two paths first intersect, such that $P'_u(l)$, $P'_v(l)$ are defined and $P'_u(l) < P'_v(l)$. But then from Fact 5 $P'_u(l-1) \neq P'_v(l-1)$, a contradiction. The proof for the second part of the lemma is similar. ∎

That P_u and P_{BN} are disjoint follows from using Lemma 4.11.1 on u and the sibling of u in P_{BN}.

Next we show that for distinct $u, v \in S$, the paths P_u and P_v do not intersect. Let us first show that P_u does not contain v, and by symmetry we will have that P_v does not contain u. Suppose for the sake of contradiction that P_u contains v, and without loss of generality assume P_u is the left-most path to u (the other case is symmetric). Since $u \neq v$, there must be a height $l \leq \mathrm{height}(u)$ such that $P_u(l-1) = v$ and $P_u(l)$ is a parent of v. From the definition of S, we know $P_{\mathrm{BN}}(l)$ is also a parent of v. Since we assumed P_u is the left-most path to u, it must be that $P_u(l) < P_{\mathrm{BN}}(l)$. But then Fact 5 tells us that v cannot be a child of $P_{\mathrm{BN}}(l)$, a contradiction. So we have shown that P_v does not contain u, and by symmetry P_u does not contain v. Now suppose that P_u and P_v intersect at some node other than u or v. Then there is a height l, one greater than the height where they first intersect, such that $P_u(l) \neq P_v(l)$. Now, observe that P_u and P_v are both left-most paths or both right-most paths, since otherwise in order for them to intersect they would need to cross P_{BN} (which we showed does not happen). But then from Fact 5 $P_u(l-1) \neq P_v(l-1)$, a contradiction.

That completes the proof of Claim 1 and hence of Proposition 4.9.1. We now just need to prove Proposition 4.12.1 and then apply Klawe's Theorem 4.8.

Definition 4.12 A DAG H is *nice* if the following conditions hold.

(1) If u_1 and u_2 are sibling nodes[10] in H then the cost of black pebbling u_1 is equal to the cost of black pebbling u_2.

(2) If u_1 and u_2 are siblings, then there is no path from u_1 to u_2 or from u_2 to u_1.

(3) If $u, u_1, \ldots, u_m$ are nodes none of which has a path to any of the others, then there are node-disjoint paths $P_1, \ldots, P_m$ such that P_i is a path from a leaf to u_i and there is no path between u and any node in P_i.

Proposition 4.12.1 *G' is nice.*

Proof. Property 2 is obviously satisfied.

For Property 1, the argument used to give the black pebbling lower bound of $(c-1) + c(d-1)(h-1) + 1$ can be used to give a lower bound of $c(d-1)(h'-1) + 1$

10. That is, they have a parent in common.

for the cost of black pebbling any node at height $h' \leq h$.[11] Moreover that bound is easily shown to be tight.

For Property 3, we can choose P_i to be the left-most (respectively, right-most) path from u_i if u_i is less than (respectively, greater than) u. We then use Lemma 4.11.1 on each pair of nodes in $\{u, u_1, \ldots, u_m\}$. ∎

4.3 White Sliding Moves

In the definition of fractional pebbling (Definition 2.6) we allow black sliding moves but not white sliding moves. To allow white sliding moves we would add a clause.

(4) For every internal node i, decrease $w(i)$ to 0 and increase the white pebble value of each child of i so that each child has total pebble value 1.

We did not include this move in the original definition because a nondeterministic k-way BP solving $FT_d^h(k)$ or $BT_d^h(k)$ does not naturally simulate it. The natural way to simulate such a move would be to verify the conjectured value of node i (conjectured when the white pebble was placed on i) by comparing it with $f_i(v_{j_1}, \ldots, v_{j_d})$, where $j_1, \ldots, j_d$ are the children of i. But this would require the BP to remember a $(d+1)$-tuple of values, whereas potentially only d pebbles are involved.

White sliding moves definitely reduce the number of pebbles required to pebble some trees. For example the binary tree T_2^3 can easily be pebbled with 2 pebbles using white sliding moves, but requires 2.5 pebbles without (Theorem 4.4). The next result shows that 8/3 pebbles suffice for pebbling T_2^4 with white sliding moves, whereas 3 pebbles are required without (Theorem 4.4).

Theorem 4.13 *The binary tree of height 4 can be pebbled with* 8/3 *pebbles using white sliding moves.*

Proof. The height 3 binary tree can be pebbled with 2 pebbles. Use that sequence on node 2, but leave one third of a black pebble on node 2. That takes 7/3 pebbles. Put black pebbles on nodes 12 and 13. Slide one third of a black pebble up to node 6. Remove the pebbles on nodes 12 and 13. Put black pebbles on nodes 14 and 15; this is the first configuration with 8/3 pebbles. Slide the pebble on node 14 up to node 7. Remove the pebble from 15. Put 2/3 of a white pebble on node 6. Slide the black pebble on node 7 up to node 3. Remove one third of a black pebble from node 6. Put 2/3 of a white pebble on node 2; the resulting configuration has 8/3 pebbles. Slide the black pebble on node 3 up to the root. Remove all black pebbles. At this point there is 2/3 of a white pebble on both node 2 and node 6. Put a black pebble on node 12 and one-third of a black pebble on node 13, another bottleneck. Slide the

11. The only change is in the size of the set of nodes S. For the root, which is at height $h+1$, S has size $(c-1) + c(d-1)(h-1)$, whereas for a height $h' \leq h$ node, S has size $c(d-1)(h'-1)$.

2/3 white pebble on node 6 down to node 13. Remove the pebbles from nodes 12 and 13. Finally, use 8/3 pebbles to remove the 2/3 white pebble from node 2. ■

5 Branching Program Bounds

In this section we prove tight bounds (up to a constant factor) for the number of states required for both deterministic and nondeterministic k-way branching programs to solve the Boolean problems $BT_d^h(k)$ for all trees of height $h = 2$ and $h = 3$. (The bound is obviously $\Theta(k^d)$ for trees of height 2, because there are $d + k^d$ input variables.) For every height $h \geq 2$ we prove upper bounds for deterministic thrifty programs which solve $FT_d^h(k)$ (Theorem 5.1, (15)), and show that these bounds are optimal for degree $d = 2$ even for the Boolean problem $BT_d^h(k)$ (Theorem 5.11). We prove upper bounds for nondeterministic thrifty programs solving $BT_d^h(k)$ in general, and show that these are optimal for binary trees of height 4 or less (Theorems 5.1 and 5.15).

For the nondeterministic case our best BP upper bounds for every $h \geq 2$ come from fractional pebbling algorithms via Theorem 3.4. For the deterministic case our best bounds for the function problem $FT_d^h(k)$ come from black pebbling via the same theorem, although we can improve on them for the Boolean problem $BT_2^h(k)$ by a factor of $\log k$ (for $h \geq 3$).

Theorem 5.1 **BP Upper Bounds**
For all $h, d \geq 2$

$$\#\mathsf{detFstates}_d^h(k) = O(k^{(d-1)h-d+2}), \tag{15}$$

$$\#\mathsf{detBstates}_d^h(k) = O(k^{(d-1)h-d+2}/\log k), \quad \textit{for } h \geq 3, \tag{16}$$

$$\#\mathsf{ndetBstates}_d^h(k) = O(k^{(d-1)(h/2)+1}). \tag{17}$$

The first and third bounds are realized by thrifty programs.

Proof. The first and third bounds follow from Theorem 3.4 (which states that pebbling upper bounds give rise to upper bounds for the size of thrifty BPs) and from Theorems 4.1 and 4.4 (which give the required pebbling upper bounds).

To prove (16) we use a branching program which implements the following algorithm. Here we have a parameter m, and choosing $m = \lceil \log k^{d-1} - \log\log k^{d-1} \rceil$ suffices to show $\#\mathsf{detBstates}_d^h(k) = O(k^{(d-1)(h-1)+1}/\log k^{d-1})$, from which (16) follows. We estimate the number of states required up to a constant factor.

(1) Compute v_2 (the value of node 2 in the heap ordering), using the black pebbling algorithm for the principal left subtree. This requires $k^{(d-1)(h-2)+1}$ states. Divide the k possible values for v_2 into $\lceil k/m \rceil$ blocks of size m.

(2) Remember the block number for v_2, and compute $v_3, \ldots, v_{d+1}$. This requires $k/m \times k^{d-2} \times k^{(d-1)(h-2)+1} = k^{(d-1)(h-1)+1}/m$ states.

(3) Remember $v_3, \ldots, v_{d+1}$ and the block number for v_2. Compute $f_1(a, v_3, \ldots, v_{d+1})$ for each of the m possible values a for v_2 in its block number, and keep track of the set of a's for which $f_1 = 1$. This requires $k^{d-1} \times k/m \times m \times 2^m = k^d 2^m$ states.

(4) Remember just the set of possible a's (within its block) from before (there are 2^m possibilities). Compute v_2 again and accept or reject depending on whether v_2 is in the subset. This requires $k^{(d-1)(h-2)+1} 2^m$ states.

The total number of states has order the maximum of $k^{(d-1)(h-1)+1}/m$ and $k^{(d-1)(h-2)+1}2^m$, which is at most

$$k^{(d-1)(h-1)+1}/(\log k^{d-1} - \log\log k^{d-1})$$

for $m = \log k^{d-1} - \log\log k^{d-1}$. ∎

We combine the preceding upper bounds with the Nečiporuk lower bounds in Section 5.1, Figure 6, to obtain the following.

Corollary 5.2 **Tight bounds for height 3 trees**
For all $d \geq 2$

$$\#\mathsf{detFstates}_d^3(k) = \Theta(k^{2d-1})$$
$$\#\mathsf{detBstates}_d^3(k) = \Theta(k^{2d-1}/\log k)$$
$$\#\mathsf{ndetBstates}_d^3(k) = \Theta(k^{(3/2)d-1/2}).$$

Model	Lower bound for $FT_d^h(k)$	Lower bound for $BT_d^h(k)$
Deterministic k-way branching program	$\boxed{\frac{d^{h-2}-1}{4(d-1)^2} \cdot k^{2d-1}}$	$\boxed{\frac{d^{h-2}-1}{3(d-1)^2} \cdot \frac{k^{2d-1}}{\log k}}$
Deterministic binary branching program	$\frac{d^{h-2}-1}{5(d-1)^2} \cdot k^{2d} = \Omega(n^2/(\log n)^2)$	$\frac{d^{h-2}-1}{4d(d-1)} \cdot \frac{k^{2d}}{\log k} = \Omega(n^2/(\log n)^3)$
Nondeterministic k-way BP	$\frac{d^{h-2}-1}{2d-2} \cdot k^{\frac{3d}{2}-\frac{1}{2}}\sqrt{\log k}$	$\boxed{\frac{d^{h-2}-1}{2d-2} \cdot k^{\frac{3d}{2}-\frac{1}{2}}}$
Nondeterministic binary BP	$\frac{d^{h-2}-1}{2d-2} \cdot k^{\frac{3d}{2}}\sqrt{\log k} = \Omega(n^{3/2}/\log n)$	$\frac{d^{h-2}-1}{2d-2} \cdot k^{\frac{3d}{2}} = \Omega(n^{3/2}/(\log n)^{3/2})$

Figure 6 Size bounds for k large enough, expressed in terms of $n = \Omega(k^d \log k)$ in the binary cases, obtained by applying the Nečiporuk method. Rectangles indicate optimality in k when $h = 3$ (Corollary 5.2). Improving any entry to $\Omega(k^{\text{unbounded } f(h)})$ would prove $\mathbf{L} \subsetneq \mathbf{P}$ (Corollary 3.3).

5.1 The Nečiporuk Method

By applying the Nečiporuk method to a k-way branching program B computing a function $f : [k]^m \to R$, we mean the following well-known steps [Neč66] (see Wegener [Weg00]).

(1) Upper bound the number $N(s, v)$ of (syntactically) distinct branching programs of type B having s nonfinal states, each labeled by one of v variables.

(2) Pick a partition $\{V_1, \ldots, V_p\}$ of $[m]$.

(3) For $1 \leq i \leq p$, lower bound the number $r_{V_i}(f)$ of restrictions $f_{V_i} : [k]^{|V_i|} \to R$ of f obtainable by fixing values of the variables in $[m] \backslash V_i$.

(4) Then size $(B) \geq |R| + \sum_{1 \leq i \leq p} s_i$, where $s_i = \min\{s : N(s, |V_i|) \geq r_{V_i}(f)\}$.

The Nečiporuk method still yields the strongest explicit binary branching program size lower bounds known today, namely $\Omega\left(\frac{n^2}{(\log n)^2}\right)$ for the deterministic case [Neč66] and $\Omega\left(\frac{n^{3/2}}{\log n}\right)$ for the nondeterministic case ([Pud87a], see Razborov [Raz91]). It is known that the previous lower bounds are the best that can be obtained using the Nečiporuk method. For the deterministic case this is stated with proof hints in Wegener [Weg87, p. 422]. An argument for the nondeterministic case is made in Beame and McKenzie [BM12].

Theorem 5.3 *Applying the Nečiporuk method yields Figure 6.*

Remark 5.4 Our $\Omega(n^{3/2}/(\log n)^{3/2})$ binary nondeterministic BP lower bound for the $BT_d^h(k)$ problem and in particular for $BT_2^3(k)$ applies to BP "state-size" defined here as the number of *states* in the BP. By comparison, Pudlak's $\Omega(n^{3/2}/\log n)$ lower bound [Pud87a, Raz91] (for a different Boolean function) applies to the "edge-size" of the closely related switching and rectifier network model, where "edge-size" is defined as the number of (labeled) *edges* in the network. Because switching and rectifier networks can also use unlabeled edges, any k-way nondeterministic BP with state-size S can be simulated by a network of edge-size at most kS (regardless of the BP outdegree). Pudlak's $\Omega(n^{3/2}/\log n)$ bound thus applies as well to the number of states in a binary nondeterministic BP computing his function, and his bound is the best that the Nečiporuk method can achieve [BM12].

Proof of Theorem 5.3. We have $N_{\text{det}}^{k\text{-way}}(s, v) \leq v^s \cdot (s + |R|)^{sk}$ for the number of deterministic BPs and $N_{\text{nondet}}^{k\text{-way}}(s, v) \leq v^s \cdot (|R|+1)^{sk} \cdot (2s)^{sk}$ for nondeterministic BPs having s nonfinal states, each labeled with one of v variables. To see the latter bound, note that edges labeled $i \in [k]$ can connect a state S to zero or one state among the final states and can connect S independently to any number of states among the nonfinal states.

The only decision to make when applying the Nečiporuk method is the choice of the partition of the input variables. Here every entry in Figure 6 is obtained using the same partition (with the understanding that a k-ary variable in the partition is replaced by $\log k$ binary variables when we treat 2-way branching programs).

We will only partition the set V of k-ary $FT_d^h(k)$ or $BT_d^h(k)$ variables that pertain to internal tree nodes other than the root (we will neglect the root and leaf variables). Each internal tree node has $d-1$ siblings and each sibling involves k^d variables. By a *litter* we will mean any set of d k-ary variables that pertain to precisely d such siblings. We obtain our partition by writing V as a union of

$$k^d \cdot \Sigma_{i=0}^{h-3} d^i = k^d \cdot \frac{d^{h-2}-1}{d-1}$$

litters. (Specifically, each litter can be defined as

$$\{f_i(j_1,j_2,\ldots,j_d),f_{i+1}(j_1,j_2,\ldots,j_d),\ldots,f_{i+d-1}(j_1,j_2,\ldots,j_d)\}$$

for some $1 \le j_1,j_2,\ldots,j_d \le k$ and some d siblings $i, i+1,\ldots,i+d-1$.)

Consider such a litter L. We claim that $|R|^{k^d}$ distinct functions $f_L : [k]^d \to R$ can be induced by setting the variables outside of L, where $|R| = k$ in the case of $FT_d^h(k)$ and $|R| = 2$ in the case of $BT_d^h(k)$. Indeed, to induce any such function, fix the "descendants of the litter L" to make each variable in L relevant to the output; then, set the variables pertaining to the immediate ancestor node ν of the siblings forming L to the appropriate k^d values, as if those were the final output desired; finally, set all the remaining variables in a way such that the values in ν percolate from ν to the root.

It remains to do the calculations. We illustrate two cases. Similar calculations yield the other entries in Figure 6.

Nondeterministic k-way branching programs computing $FT_d^h(k)$. Here $|R| = k$. In a correct program, the number s of states querying one of the d litter L variables must satisfy

$$k^{k^d} \le N_{\text{nondet}}^{k\text{-way}}(s,d) \le d^s \cdot (k+1)^{sk} \cdot (2^s)^{sk} \le s^s \cdot k^{2sk} \cdot (2^s)^{sk}$$

since $d \le s$ (because $FT_d^h(k)$ depends on all its variables), and thus

$$k^d \log k \le s(\log s + 2k \log k) + s^2 k.$$

Suppose to the contrary that $s < (k^{\frac{d-1}{2}}\sqrt{\log k})/2$. Then

$$\begin{aligned} s(\log s + 2k\log k) + s^2 k &< s\left(\frac{d-1}{2}\log k + \frac{\log\log k}{2} + 2k\log k\right) + s^2 k \\ &< s(sk) + s^2 k < k^d \log k \end{aligned}$$

for large k and all $d \geq 2$, a contradiction. Hence $s \geq (k^{\frac{d-1}{2}}\sqrt{\log k})/2$. Since this holds for every litter, recalling step 4 in the Nečiporuk method as described prior to Theorem 5.3, the total number of states in the program is at least

$$k + k^d \cdot \frac{d^{h-2}-1}{d-1} \cdot \left(k^{\frac{d-1}{2}}\sqrt{\log k}\right)/2 \geq \frac{d^{h-2}-1}{2d-2} \cdot k^{\frac{3d}{2}-\frac{1}{2}}\sqrt{\log k}.$$

Nondeterministic binary (i.e., 2-way) branching programs deciding $BT_d^h(k)$. Here $|R| = 2$. When the program is binary, the d variables in the litter L become $d \log k$ Boolean variables. The number s of states querying one of these $d \log k$ variables then satisfies

$$2^{k^d} \leq N_{\text{nondet}}^{\text{2-way}}(s, d\log k) \leq (d\log k)^s \cdot (2+1)^{2s} \cdot (2^s)^{2s} < (s\log k)^s \cdot 2^{4s+2s^2}$$

since $d \leq s$ and thus

$$k^d \leq s\log s + s\log\log k + 4s + 2s^2 \leq 3s^2 + 5s\log\log k.$$

It follows that $s \geq k^{\frac{d}{2}}/2$. Hence the total number of states in a binary nondeterministic program deciding $BT_d^h(k)$ is at least

$$k^d \cdot \frac{d^{h-2}-1}{d-1} \cdot \frac{k^{d/2}}{2} \geq \frac{d^{h-2}-1}{2(d-1)} \cdot k^{\frac{3d}{2}} = \frac{d^{h-2}-1}{2(d-1)} \cdot \frac{(k^d\log k)^{3/2}}{(\log k)^{3/2}} = \Omega(n^{3/2}/(\log n)^{3/2}),$$

where $n = \Theta(k^d \log k)$ is the length of the binary encoding of $BT_d^h(k)$. ∎

The next two results together with Theorems 5.9 and 5.10 show limitations on the Nečiporuk method that are not necessarily present in the state sequence method. We include these to support our hope that the latter method and its generalizations have the potential to break the quadratic limitation in proving lower bounds using the Nečiporuk method.

Let $Children_d^h(k)$ have the same input as $FT_d^h(k)$ with the exception that the root function is deleted. The output is the tuple $(v_2, v_3, \ldots, v_{d+1})$ of values for the children of the root. $Children_d^h(k)$ can be computed by a k-way deterministic BP with $O(k^{(d-1)h-d+2})$ states using the same black pebbling method which yields the bound (15) in Theorem 5.1.

Theorem 5.5 *For any $d, h \geq 2$, the best k-way deterministic BP size lower bound attainable for $Children_d^h(k)$ by applying the Nečiporuk method is $\Omega(k^{2d-1})$.*

Proof. The function $Children_d^h(k) : [k]^m \to R$ has $m = \Theta(k^d)$. Any partition $\{V_1, \ldots, V_p\}$ of the set of k-ary input variables thus has $p = O(k^d)$. Claim: for each i,

the best attainable lower bound on the number of states querying variables from V_i is $O(k^{d-1})$.

Consider such a set V_i, $|V_i| = v \geq 1$. Here $|R| = k^d$, so the number $N_{\text{det}}^{k\text{-way}}(s, v)$ of distinct deterministic BPs having s nonfinal states querying variables from V_i satisfies

$$N_{\text{det}}^{k\text{-way}}(s, v) \geq 1^s \cdot (s + |R|)^{sk} \geq (1 + k^d)^{sk} \geq k^{dsk}.$$

Hence the estimate used in the Nečiporuk method to upper bound $N_{\text{det}}^{k\text{-way}}(s, v)$ will be at least k^{dsk}. On the other hand, the number of functions $f_{V_i} : [k]^v \to R$ obtained by fixing variables outside of V_i cannot exceed $k^{O(k^d)}$ since the number of variables outside V_i is $\Theta(k^d)$. Hence the best lower bound on the number of states querying variables from V_i obtained by applying the method will be no larger than the smallest s verifying $k^{ck^d} \leq k^{dsk}$ for some c depending on d and k. This proves our claim since then this number is at most $s = O(k^{d-1})$. ∎

Let $SumMod_d^h(k)$ have the same input as $FT_d^h(k)$ with the exception that the root function is preset to the sum modulo k. In other words the output is $v_2 + v_3 + \cdots + v_{d+1} \bmod k$.

Theorem 5.6 *The best k-way deterministic BP size lower bound attainable for $SumMod_2^3(k)$ by applying the Nečiporuk method is $\Omega(k^2)$.*

Proof. The function $SumMod_2^3(k) : [k]^m \to R$ has $m = \Theta(k^2)$. Consider a set V_i in any partition $\{V_1, \ldots, V_p\}$ of the set of k-ary input variables, $|V_i| = v$. Here $|R| = k$, so the number $N_{\text{det}}^{k\text{-way}}(s, v)$ of distinct deterministic BPs having s nonsink states querying variables from V_i satisfies

$$N_{\text{det}}^{k\text{-way}}(s, v) \geq 1^s \cdot (s + |R|)^{sk} \geq (1 + k)^{sk} \geq k^{sk}.$$

If V_i contains a leaf variable, then perhaps the number of functions induced by setting variables complementary to V_i can reach the maximum k^{k^2}. Nečiporuk would conclude that k states querying the variables from such a V_i are necessary. Note that there are at most 4 sets V_i containing a leaf variable (hence a total of $4k$ states required to account for the variables in these 4 sets). Now suppose that V_i does not contain a leaf variable. Then setting the variables complementary to V_i can either induce a constant function (there are k of those), or the sum of a constant plus a variable (there are at most $k \cdot |V_i|$ of those) or the sum of two of the variables (there are at most $|V_i|^2$ of those). So the maximum number of induced functions is $|V_i|^2 = O(k^4)$. The number of states querying variables from V_i is found by Nečiporuk to be $s \geq 4/k$. In other words $s = 1$. So for any of the at least $p - 4$

sets in the partition not containing a leaf variable, the method gets one state. Since $p - 4 = O(k^2)$, the total number of states accounting for all the V_i is $O(k^2)$. ■

5.2 The State Sequence Method

Here we give alternative proofs for some of the lower bounds given in Section 5.1. These proofs are more intricate than the Nečiporuk proofs but they do not suffer a priori from a quadratic limitation. The method also yields stronger lower bounds for $Children_2^4(k)$ and $SumMod_2^3(k)$ (Theorems 5.9 and 5.10) than those obtained by applying Nečiporuk's method (Theorems 5.5 and 5.6).

Theorem 5.7 $\#\mathsf{ndetBstates}_2^3(k) \geq k^{2.5}$ *for sufficiently large k.*

Proof. Consider an input I to $BT_2^3(k)$. We number the nodes in T_2^3 as in Figure 1, and let v_j^I denote the valueof node j under input I. We say that a state in a computation on input I *learns* v_j^I if that state queries $f_j^I(v_{2j}^I, v_{2j+1}^I)$ (recall $2j, 2j+1$ are the children of node j).

Definition **Learning Interval**

Let B be a k-way nondeterministic BP that solves $BT_2^3(k)$. Let $\mathcal{C} = \gamma_0, \gamma_1, \cdots, \gamma_T$ be a computation of B on input I. We say that a state γ_i in the computation is *critical* if one or more of the following holds.

(1) $i = 0$ or $i = T$.
(2) γ_i learns v_2^I and there is an earlier state which learns v_3^I with no intervening state that learns v_2^I.
(3) γ_i learns v_3^I and no earlier state learns v_3^I unless an intervening state learns v_2^I.

We say that a subsequence $\gamma_i, \gamma_{i+1}, \cdots \gamma_j$ is a *learning interval* if γ_i and γ_j are consecutive critical states. The interval is *type 3* if γ_i learns v_3^I, and otherwise the interval is *type 2*.

The reason for the assymetry in the previous definition is that the initial state γ_0 of B may learn neither v_2^I nor v_3^I, in which case the initial learning interval is type 2. Since the tree T_2^3 has a symmetry which interchanges nodes 2 and 3, we may assume without loss of generality that γ_0 does not query the function f_3 and hence it does not learn v_3^I no matter what the input I. Thus a type 2 learning interval begins with γ_0 and/or a state which learns v_2^I, and never learns v_3^I until the last state. A type 3 learning interval begins with a state which learns v_3^I and never learns v_2^I until the last state.

Now let B be as earlier, and for $j \in \{2,3\}$ let Γ_j be the set of all states of B which make a query of the form $f_j(x,y)$ for some $x,y \in [k]$. We will prove the theorem by showing that for large k

$$|\Gamma_2| + |\Gamma_3| > k^2\sqrt{k}. \tag{18}$$

For $r,s \in [k]$ let $F_{yes}^{r,s}$ be the set of inputs I to B whose four leaves are labeled r, s, r, s respectively, whose middle node functions f_2^I and f_3^I are identically 1 except $f_2^I(r,s) = v_2^I$ and $f_3^I(r,s) = v_3^I$, and $f_1^I(v_2^I, v_3^I) = 1$ (so $v_1^I = 1$). Thus each such I is a "YES input", and should be accepted by B.

Note that for fixed r, s, each member I of $F_{yes}^{r,s}$ is uniquely specified by a triple

$$(v_2^I, v_3^I, f_1^I) \text{ where } f_1^I(v_2^I, v_3^I) = 1 \tag{19}$$

and we assume $f_1^I : [k] \times [k] \to \{0,1\}$, so $F_{yes}^{r,s}$ has exactly $k^2(2^{k^2-1})$ members.

For $j \in \{2,3\}$ and $r,s \in [k]$ let $\Gamma_j^{r,s}$ be the subset of Γ_j consisting of those states which query $f_j(r,s)$. Then Γ_j is the disjoint union of $\Gamma_j^{r,s}$ over all pairs (r,s) in $[k]\times[k]$. Hence to prove (18) it suffices to show

$$|\Gamma_2^{r,s}| + |\Gamma_3^{r,s}| > \sqrt{k} \tag{20}$$

for large k and all r, s in $[k]$. We will show this by showing

$$(|\Gamma_2^{r,s}| + 1)(|\Gamma_3^{r,s}| + 1) \geq k/2 \tag{21}$$

for all $k \geq 2$. (Note that given the product, the sum is minimized when the summands are equal.)

For each input I in $F_{yes}^{r,s}$ we associate a fixed accepting computation $\mathcal{C}(I)$ of B on input I.

Now fix $r,s \in [k]$. For $a,b \in [k]$ and $f : [k] \times [k] \to \{0,1\}$ with $f(a,b) = 1$ we use (a,b,f) to denote the input I in $F_{yes}^{r,s}$ it represents as in (19).

To prove (21), the idea is that if it is false, then as I varies through all inputs (a,b,f) in $F_{yes}^{r,s}$ there are too few states learning $v_2^I = a$ and $v_3^I = b$ to verify that $f(a,b) = 1$. Specifically, we can find a, b, f, g such that $f(a,b) = 1$ and $g(a,b) = 0$, and by cutting and pasting the accepting computation $\mathcal{C}(a,b,f)$ with accepting computations of the form $\mathcal{C}(a,b',g)$ and $\mathcal{C}(a',b,g)$ we can construct an accepting computation of the "NO input" (a,b,g).

We may assume that the branching program B has a unique initial state γ_0 and a unique accepting state δ_{ACC}.

For $j \in \{2,3\}$, $a,b \in [k]$ and $f : [k] \times [k] \to \{0,1\}$ with $f(a,b) = 1$ define $\varphi_j(a,b,f)$ to be the set of all state pairs (γ,δ) such that there is a type j learning interval in

$\mathcal{C}(a,b,f)$ which begins with γ and ends with δ. Note that if $j=2$ then $\gamma \in (\Gamma_2^{r,s} \cup \{\gamma_0\})$ and $\delta \in (\Gamma_3^{r,s} \cup \{\delta_{ACC}\})$, and if $j=3$ then $\gamma \in \Gamma_3^{r,s}$ and $\delta \in (\Gamma_2^{r,s} \cup \{\delta_{ACC}\})$.

To complete the definition, define $\varphi_j(a,b,f) = \emptyset$ if $f(a,b) = 0$.

For $j \in \{2,3\}$ and $f : [k] \times [k] \to \{0,1\}$ we define a function $\varphi_j[f]$ from $[k]$ to sets of state pairs as

$$\varphi_2[f](a) = \bigcup_{b \in [k]} \varphi_2(a,b,f) \subseteq S_2,$$
$$\varphi_3[f](b) = \bigcup_{a \in [k]} \varphi_3(a,b,f) \subseteq S_3,$$

where $S_2 = (\Gamma_2^{r,s} \cup \{\gamma_0\}) \times (\Gamma_3^{r,s} \cup \{\delta_{ACC}\})$ and $S_3 = \Gamma_3^{r,s} \times (\Gamma_2^{r,s} \cup \{\delta_{ACC}\})$.

For each f the function $\varphi_j[f]$ can be specified by listing a k-tuple of subsets of S_j, and hence there are at most $2^{k|S_j|}$ distinct such functions as f ranges over the 2^{k^2} Boolean functions on $[k] \times [k]$, and hence there are at most $2^{k(|S_2|+|S_3|)}$ pairs of functions $(\varphi_2[f], \varphi_3[f])$. If we assume that (21) is false, we have $|S_2| + |S_3| < k$. Hence by the pigeonhole principle there must exist distinct Boolean functions f, g such that $\varphi_2[f] = \varphi_2[g]$ and $\varphi_3[f] = \varphi_3[g]$.

Since f and g are distinct we may assume that there exist a, b such that $f(a,b) = 1$ and $g(a,b) = 0$. Since $\varphi_2[f](a) = \varphi_2[g](a)$, if (γ, δ) are the endpoints of a type 2 learning interval in $\mathcal{C}(a,b,f)$ there exists b' such that (γ, δ) are the endpoints of a type 2 learning interval in $\mathcal{C}(a,b',g)$ (and hence $g(a,b') = 1$). Similarly, if (γ, δ) are endpoints of a type 3 learning interval in $\mathcal{C}(a,b,f)$ there exists a' such that (γ, δ) are the endpoints of a type 3 learning interval in $\mathcal{C}(a',b,g)$.

Now we can construct an accepting computation for the "NO input" (a,b,g) from $\mathcal{C}(a,b,f)$ by replacing each learning interval beginning with some γ and ending with some δ by the corresponding learning interval in $\mathcal{C}(a,b',g)$ or $\mathcal{C}(a',b,g)$. (The new accepting computation has the same sequence of critical states as $\mathcal{C}(a,b,f)$.) This works because a type 2 learning interval never queries v_3 and a type 3 learning interval never queries v_2.

This completes the proof of (21) and the theorem. ∎

Theorem 5.8 *Every deterministic branching program that solves $BT_2^3(k)$ has at least $k^3/\log k$ states for sufficiently large k.*

Proof. We modify the proof of Theorem 5.7. Let B be a deterministic BP which solves $BT_2^3(k)$, and for $j \in \{2,3\}$ let Γ_j be the set of states in B which query f_j (as before). It suffices to show that for sufficiently large k

$$|\Gamma_2| + |\Gamma_3| \geq k^3/\log k. \tag{22}$$

For $r, s \in [k]$ we define the set $F^{r,s}$ to be the same as $F^{r,s}_{yes}$ except that we remove the restriction on f^I_1. Hence there are exactly $k^2 2^{k^2}$ inputs in $F^{r,s}$.

As before, for $j \in \{2,3\}$, Γ_j is the disjoint union of $\Gamma^{r,s}$ for $r, s \in [k]$. Thus to prove (22) it suffices to show that for sufficiently large k and all r, s in $[k]$

$$|\Gamma^{r,s}_2| + |\Gamma^{r,s}_3| \geq k/\log k. \tag{23}$$

We may assume there are unique start, accepting, and rejecting states γ_0, δ_{ACC}, δ_{REJ}. Fix $r, s \in [k]$.

For each root function $f : [k] \times [k] \to \{0,1\}$ we define the functions

$$\psi_2[f] : [k] \times (\Gamma^{r,s}_2 \cup \{\gamma_0\}) \to (\Gamma^{r,s}_3 \cup \{\delta_{ACC}, \delta_{REJ}\})$$
$$\psi_3[f] : [k] \times \Gamma^{r,s}_3 \to (\Gamma^{r,s}_2 \cup \{\delta_{ACC}, \delta_{REJ}\})$$

by $\psi_2[f](a,\gamma) = \delta$ if δ is the next critical state after γ in a computation with input (a,b,f) (this is independent of b), or $\delta = \delta_{REJ}$ if there is no such critical state. Similarly $\psi_3[f](b,\delta) = \gamma$ if γ is the next critical state after δ in a computation with input (a,b,f) (this is independent of a), or $\delta = \delta_{REJ}$ if there is no such critical state.

Claim *The pair of functions* $(\psi_2[f], \psi_3[f])$ *is distinct for distinct* f.

For suppose otherwise. Then there are f, g such that $\psi_2[f] = \psi_2[g]$ and $\psi_3[f] = \psi_3[g]$ but $f(a,b) \neq g(a,b)$ for some a, b. But then the sequences of critical states in the two computations $C(a,b,f)$ and $C(a,b,g)$ must be the same, and hence the computations either accept both (a,b,f) and (a,b,g) or reject both. So the computations cannot both be correct.

Finally we prove (23) from the claim. Let $s_2 = |\Gamma^{r,s}_2|$ and let $s_3 = |\Gamma^{r,s}_3|$, and let $s = s_2 + s_3$. Then the number of distinct pairs (ψ_2, ψ_3) is at most

$$(s_3+2)^{k(s_2+1)}(s_2+2)^{ks_3} \leq (s+2)^{k(s+1)}$$

and since there are 2^{k^2} functions f we have

$$2^{k^2} \leq (s+2)^{k(s+1)}$$

so taking logs, $k^2 \leq k(s+1)\log(s+2)$ so $k/\log(s+2) \leq s+1$, and (23) follows. ∎

Recall from Theorem 5.5 that applying the Nečiporuk method to $Children^4_2(k)$ yields a nonoptimal $\Omega(k^3)$ size lower bound and from Theorem 5.6 that applying it to $SumMod^3_2(k)$ yields a nonoptimal $\Omega(k^2)$ lower bound. The next two results improve on these bounds using the state sequence method. The new lower bounds match the upper bounds given by the pebbling method used to prove (15) in Theorem 5.1.

Theorem 5.9 *Any deterministic k-way BP for $Children_2^4(k)$ has at least $k^4/2$ states.*

Proof. Let E_{4true} be the set of all inputs I to $Children_2^4(k)$ such that $f_2^I = f_3^I = +_k$ (addition mod k), and for $i \in \{4,5,6,7\}$ f_i^I is identically 0 except for $f_i^I(v_{2i}^I, v_{2i+1}^I)$.

Let B be a branching program as in the theorem. For each $I \in E_{4true}$ let $\mathcal{C}(I)$ be the computation of B on input I.

For $r, s \in [k]$ let $E_{4true}^{r,s}$ be the set of inputs I in E_{4true} such that for $i \in \{4,5,6,7\}$, $v_{2i}^I = r$ and $v_{2i+1}^I = s$. Then for each pair r, s each input I in $E_{4true}^{r,s}$ is completely specified by the quadruple $v_4^I, v_5^I, v_6^I, v_7^I$, so $|E_{4true}^{r,s}| = k^4$.

For $r, s \in [k]$ and $i \in \{4,5,6,7\}$ let $\Gamma_i^{r,s}$ be the set of states of B that query $f_i(r,s)$, and let

$$\Gamma^{r,s} = \Gamma_4^{r,s} \cup \Gamma_5^{r,s} \cup \Gamma_6^{r,s} \cup \Gamma_7^{r,s}. \tag{24}$$

The theorem follows from the following claim.

Claim 1 $|\Gamma^{r,s}| \geq k^2/2$ *for all* $r, s \in [k]$.

To prove Claim 1, suppose to the contrary for some r, s.

$$|\Gamma^{r,s}| < k^2/2 \tag{25}$$

We associate a pair

$$T(I) = (\gamma^I, v_i^I)$$

with I as follows: γ^I is the last state in the computation $\mathcal{C}(I)$ that is in $\Gamma^{r,s}$ (such a state clearly exists), and $i \in \{4,5,6,7\}$ is the node queried by γ^I. (Here v_i^I is the value of node i).

We also associate a second triple $U(I)$ with each input I in $E_{4true}^{r,s}$ as follows.

$$U(I) = \begin{cases} (v_4^I, v_5^I, v_3^I) & \text{if } \gamma^I \text{ queries node 4 or 5} \\ (v_6^I, v_7^I, v_2^I) & \text{otherwise} \end{cases}$$

Claim 2 *As I ranges over $E_{4true}^{r,s}$, $U(I)$ ranges over at least $k^3/2$ triples in $[k]^3$.*

To prove Claim 2, consider the subset E' of inputs in $E_{4true}^{r,s}$ whose values for nodes 4,5,6,7 have the form a, b, a, c for arbitrary $a, b, c \in [k]$. For each such I in E' an adversary trying to minimize the number of triples $U(I)$ must choose one of the two triples $(a, b, a +_k c)$ or $(a, c, a +_k b)$. There are a total of k^3 distinct triples of each of the two forms, and the adversary must choose at least half the triples from

one of the two forms, so there must be at least $k^3/2$ distinct triples of the form $U(I)$. This proves Claim 2.

On the other hand by (25) there are fewer than $k^3/2$ possible values for $T(I)$. Hence there exist inputs $I,J \in E^{r,s}_{4true}$ such that $U(I) \neq U(J)$ but $T(I) = T(J)$. Since $U(I) \neq U(J)$ but $v^I_i = v^J_i$ (where i is the node queried by $\gamma^I = \gamma^J$) it follows that either $v^I_2 \neq v^J_2$ or $v^I_3 \neq v^J_3$, so I and J give different values to the function $Children^4_2(k)$. But since $T(I) = T(J)$ if follows that the two computations $\mathcal{C}(I)$ and $\mathcal{C}(J)$ are in the same state $\gamma^I = \gamma^J$ the last time any of the nodes $\{4,5,6,7\}$ is queried, and the answers $v^I_i = v^J_i$ to the queries are the same, so both computations give identical outputs. Hence one of them is wrong. ∎

Theorem 5.10 *Any deterministic k-way BP for $SumMod^3_2(k)$ requires at least k^3 states.*

Proof. We adapt the previous proof. Now $E^{r,s}$ is the set of inputs I to $SumMod^3_2(k)$ such that for $i \in \{2,3\}$, f^I_i is identically one except possibly for $f^I_i(r,s)$, and $v^I_4 = v^I_6 = r$ and $v^I_5 = v^I_7 = s$. Note that an input to $E^{r,s}$ can be specified by the pair (v^I_2, v^I_3), so $E^{r,s}$ has exactly k^2 elements. Define

$$\Gamma^{r,s} = \Gamma^{r,s}_2 \cup \Gamma^{r,s}_3.$$

Now we claim that an input I in $E^{r,s}$ can be specified by the pair (γ^I, v^I_i), where γ^I is the last state in the computation $\mathcal{C}(I)$ that is in $\Gamma^{r,s}$, and $i \in \{2,3\}$ is the node queried by γ^I.

The claim holds because (γ^I, v^I_i) determines the output of the computation, which in turn (together with v^I_i) determines v^I_j, where j is the sibling of i.

From the claim it follows that $|\Gamma^{r,s}| \geq k$ for all $r,s \in [k]$, and hence there must be at least $k3$ states in total. ∎

5.3 Thrifty Lower Bounds

Recall (Definition 2.4) that a thrifty branching program can only query $f_i(\vec{x})$ if $\vec{x}$ is the correct vector of values for the children of node i.

Theorem 5.11 next shows that the upper bound given in Theorem 5.1 (15) is optimal for deterministic thrifty programs solving the function problem $FT^h_d(k)$ for $d = 2$ and all $h \geq 2$. Theorem 5.15 shows that the upper bound of k^3 given in Theorem 5.1 (17) is optimal for nondeterministic thrifty programs solving the Boolean problem $BT^h_d(k)$ for $d = 2$ and $h = 4$ (it is optimal for $h \leq 3$ by Theorem 5.2). We have not been able to extend this last result to $h > 4$.

Theorem 5.11 *For any h,k, every deterministic thrifty branching program solving $BT^h_2(k)$ has at least k^h states.*

Fix a deterministic thrifty BP B that solves $BT_2^h(k)$. Let E be the inputs to B. Let Vars be the set of k-valued input variables (so $|E| = k^{|\mathsf{Vars}|}$). Let Q be the states of B. If i is an internal node then the i variables are $f_i(a,b)$ for $a,b \in [k]$, and if i is a leaf node then there is just one i variable l_i. We sometimes say "f_i variable" just as an in-line reminder that i is an internal node. For $q \in Q$ let $\mathsf{var}(q)$ be the input variable that q queries. Let node be the function that maps each variable X to the node i such that X is an i variable, and each state q to $\mathsf{node}(\mathsf{var}(q))$. When it is clear from the context that q is on the computation path of an input I, we just say "q queries i" instead of "q queries the thrifty i variable of I".

Fix an input I, and let P be its computation path. If q is a state on P we say that I visits q. Let n be the number of nodes in the tree. We will choose n states on P as *critical states* for I, one for each node. Note that I must visit a state that queries the root (i.e., queries the thrifty root variable of I), since otherwise the branching program would make a mistake on an input J that is identical to I except[12] $f_1^J(v_2^I, v_3^I) := 1 - f_1^I(v_2^I, v_3^I)$; hence $J \in BT_2^h(k)$ iff $I \notin BT_2^h(k)$. So, we can choose the root critical state for I to be the last state on P that queries the root. The remainder of the definition relies on the following small lemma.

Lemma 5.12 *For any input J and internal node i, if J visits a state q that queries i, then for each child j of i, there is an earlier state on the computation path of J that queries j.*

Proof. Suppose otherwise, and without loss of generality assume the previous statement is false for $j = 2i$. For every $a \neq v_{2i}^J$ there is an input J_a that is identical to J except $v_{2i}^{J_a} = a$. But the computation paths of J_a and J are identical up to q, so J_a queries a variable $f_i(a,b)$ such that $b = v_{2i+1}^{J_a}$ and $a \neq v_{2i}^{J_a}$, which contradicts the thrifty assumption. ∎

Now we can complete the definition of the critical states of I. For i an internal node, if q is the node i critical state for I then the node $2i$ (respectively, $2i+1$) critical state for I is the last state on P before q that queries $2i$ (respectively, $2i+1$).

We say that a collection of nodes is a *minimal cut* of the tree if every path from root to leaf contains exactly one of the nodes. Now we assign a black pebbling sequence to each state on P, such that the set of pebbled nodes in each configuration is a minimal cut of the tree or a subset of some minimal cut (and once it becomes a minimal cut, it remains so), and any two adjacent configurations are either identical, or else the later one follows from the earlier one by a valid pebbling move. (Here we allow the removal of the pebbles on the children of a node i as part

12. We assume the root function $f_1 : [k] \times [k] \to \{0,1\}$.

of the move that places a pebble on i.) This assignment can be described inductively by starting with the last state on P and working backwards. Note that implicitly we will be using the following fact.

Fact 6 For any input I, if j is a descendant of i then the node j critical state for I occurs earlier on the computation path of I than the node i critical state for I.

The pebbling configuration for the output state has just a black pebble on the root. Assume we have defined the pebbling configurations for q and every state following q on P, and let q' be the state before q on P. If q' is not critical, then we make its pebbling configuration be the same as that of q. If q' is critical then it must query a node i that is pebbled in q. The pebbling configuration for q' is obtained from the configuration for q by removing the pebble from i and adding pebbles to $2i$ and $2i+1$ (if i is an internal node; otherwise you only remove the pebble from i).

Now consider the last critical state in the computation path P^I that queries a height 2 node (i.e., a parent of leaves). We use r^I to denote this state and call it the *supercritical state* of I. The pebbling configuration associated with r^I is called the bottleneck configuration, and its pebbled nodes are called *bottleneck nodes*. The two children of $\mathsf{node}(r^I)$ must be bottleneck nodes, and the bottleneck nodes form a minimal cut of the tree. The path from the root to $\mathsf{node}(r^I)$ is the *bottleneck path*, and by Fact 6 it cannot contain any bottleneck nodes. Since the bottleneck nodes form a minimal cut, each of the $h-1$ nodes on the bottleneck path has one or more distinct bottleneck nodes as descendants, and $\mathsf{node}(r^I)$ has two such descendants, namely its two children. Hence there must be at least h bottleneck nodes.

Here is the main property of the pebbling sequences that we need.

Fact 7 For any input I, if nonroot node i with parent j is pebbled at a state q on P^I, then the node j critical state q' of I occurs later on P^I, and there is no state (critical or otherwise) between q and q' on P^I that queries i.

Let R be the states that are supercritical for at least one input. Let E_r be the inputs with supercritical state r. Now we can state the main lemma.

Lemma 5.13 *For every $r \in R$, there is an surjective function from $[k]^{|\mathsf{Vars}|-h}$ to E_r.*

The lemma gives us that $|E_r| \leq k^{|\mathsf{Vars}|-h}$ for every $r \in R$. Since $\{E_r\}_{r\in R}$ is a partition of E, there must be at least $|E|/k^{|\mathsf{Vars}|-h} = k^h$ sets in the partition, that is, there must be at least k^h supercritical states. So the theorem follows from the lemma.

Proof. Fix $r \in R$ and let $D := E_r$. Let $i_{\mathsf{sc}} := \mathsf{node}(r)$. Since r is thrifty for every I in D, there are values $v^D_{2i_{\mathsf{sc}}}$ and $v^D_{2i_{\mathsf{sc}}+1}$ such that $v^I_{2i_{\mathsf{sc}}} = v^D_{2i_{\mathsf{sc}}}$ and $v^I_{2i_{\mathsf{sc}}+1} = v^D_{2i_{\mathsf{sc}}+1}$ for every I in D. The surjective function of the lemma is computed by a procedure InterAdv that takes as input a $[k]$-string (the advice), tries to interpret it as the code of an input in D, and when successful outputs that input. We want to show that for every $I \in D$ we can choose $\mathsf{adv}^I \in [k]^{|\mathrm{Vars}|-h}$ such that InterAdv$(\mathsf{adv}^I) \downarrow = I$ (i.e., the procedure terminates and returns I).

The idea is that the procedure InterAdv traces the computation path P starting from state r, using the advice string adv^I when necessary to answer queries made by each state q along the path. By the thrifty property, the procedure can "learn" the values a, b of the children of $i = \mathsf{node}(q)$ (if i is an internal node) from the query $f_i(a, b)$ of q. Each such child that has not been queried earlier in the trace saves one advice value for the future. By Fact 7 the parent of each of the h bottleneck nodes will be queried before the node itself, making a total savings of at least h values in the advice string. After the trace is completed, the remaining advice values complete the specification of the input $I \in E_r$.

In more detail, for each input I in D we consider the execution of the procedure using the advice string adv^I tailored for I. We maintain a current state q, a partial function v^* from nodes to $[k]$, and a set of nodes U_{L} (the L stands for "learned"). Once we have added a node to U_{L}, we never remove it, and once we have added $v^*(i) := a$ to the definition of v^*, we never change $v^*(i)$. We have reached q by following the same computation path that input I follows starting from r. So initially $q = r$. In general, if $v^*(i) = a \downarrow$ (i.e., $v^*(i)$ is defined and has value a) for some a then we have determined this either from reading some element of adv^I or by querying the parent of i and using the thrifty property. Initially v^* is undefined everywhere. As the procedure goes on, we may often have to use an element of the advice in order to set a value of v^*; however, by exploiting the properties of the critical state sequences, when given the complete advice adv^I for I there will be at least h nodes U^I_{L} that we "learn" without directly using the advice. Such an opportunity arises when we visit a state that queries some variable $f_i(b_1, b_2)$ and we have not yet committed to a value for at least one of $v^*(2i)$ or $v^*(2i+1)$ (if both then, we learn two nodes). When this happens, we add that child or children of i to U_{L}. So initially U_{L} is empty. There is a loop in the procedure InterAdv that iterates until $|U_{\mathsf{L}}| = h$. Note that the children of i_{sc} will be learned immediately. Let $v^*(D)$ be the inputs in D consistent with v^*, that is, $I \in v^*(D)$ iff $I \in D$ and $v^I_i = v^*(i)$ for every $i \in \mathsf{Dom}(v^*)$.

Following is the complete pseudocode for InterAdv. We also state the most important of the invariants that are maintained.

Procedure InterAdv($\vec{a} \in [k]^*$):

1: $q := r$, $U_L := \emptyset$, $v^* :=$ undefined everywhere.
2: **Loop Invariant:** If N elements of $\vec{a}$ have been used, then $|\mathsf{Dom}(v^*)| = N + |U_L|$.
3: **while** $|U_L| < h$ **do**
4: $i := \mathsf{node}(q)$
5: **if** i is an internal node and $2i \notin \mathsf{Dom}(v^*)$ or $2i+1 \notin \mathsf{Dom}(v^*)$ **then**
6: let b_1, b_2 be such that $\mathsf{var}(q) = f_i(b_1, b_2)$.
7: **if** $2i \notin \mathsf{Dom}(v^*)$ **then**
8: $v^2(2i) := b_1$ and $U_L := U_L + 2i$.
9: **end if**
10: **if** $2i+1 \notin \mathsf{Dom}(v^*)$ and $|U_L| < h$ **then**
11: $v^*(2i+1) := b_2$ and $U_L := U_L + (2i+1)$.
12: **end if**
13: **end if**
14: **if** $i \notin \mathsf{Dom}(v^*)$ **then**
15: let a be the next unused element of $\vec{a}$
16: $v^*(i) := a$.
17: **end if**
18: $q :=$ the state reached by taking the edge out of q labeled $v^*(i)$.
19: **end while**
20: let $\vec{b}$ be the next $|\mathsf{Vars}| - |\mathsf{Dom}(v^*)|$ unused elements of $\vec{a}$.
21: let $I_1, \ldots, I_{|v^*(D)|}$ be the inputs in $v^*(D)$ sorted according to some globally fixed order on E.
22: if $\vec{b}$ is the t-largest string in the lexicographical ordering of $[k]^{|\mathsf{Vars}|-|\mathsf{Dom}(v^*)|}$, and $t \leq |v^*(D)|$, then return I_t.[13]

Note that the algorithm may hang at line 18 if q is a terminal state. This can only happen if the advice string $\vec{a}$ does not correspond to any input in D.

If the loop finishes, then there are at most $|E|/|k^{\mathsf{Dom}(v^*)}| = k^{|\mathsf{Vars}|-|\mathsf{Dom}(v^*)|}$ inputs in $v^*(D)$. So for each of the inputs I enumerated on line , there is a way of setting $\vec{a}$ so that I will be chosen on line 22.

Recall we are trying to show that for every I in D there is a string $\mathsf{adv}^I \in [k]^{|\mathsf{Vars}|-h}$ such that InterAdv($\vec{a}$) $\downarrow = I$. This is easy to see under the assumption that there is

13. See after this code for argument that $|v^*(D)| \leq k^{|\mathsf{Vars}|-|\mathsf{Dom}(v^*)|}$.

such a string that makes the loop finish while maintaining the loop invariant; since the loop invariant ensures we have used $|\mathsf{Dom}(v^*)| - h$ elements of advice when we reach line , and since line is the last time when the advice is used, in all we use at most $|\mathsf{Vars}| - h$ elements of advice. To remove that assumption, first observe that for each I, we can set the advice to some adv^I so that $I \in v^*(D)$ is maintained when InterAdv is run on $\vec{a}^I$. Moreover, for that adv^I, we will never use an element of advice to set the value of a bottleneck node of I, and I has at least h bottleneck nodes. Note, however, that this does not necessarily imply that U_{L}^I (the h nodes U_{L} we obtain when running InterAdv on adv^I) is a subset of the bottleneck nodes of I. Finally, note that we are of course implicitly using the fact that no advice elements are "wasted"; each is used to set a different node value. ∎

Corollary 5.14 *For any h, k, every deterministic thrifty branching program solving $BT_2^h(k)$ has at least $\sum_{2\le l\le h} k^l$ states.*

Proof. The previous theorem only counts states that query height 2 nodes. The same proof is easily adapted to show there are at least k^{h-l+2} states that query height l nodes, for $l = 2, \ldots, h$. ∎

Theorem 5.15 *Every nondeterministic thrifty branching program solving $BT_2^4(k)$ has $\Omega(k^3)$ states.*

Proof. As in the proof of the Theorem 5.7 we restrict attention to inputs I in which the function f_i associated with each internal node i satisfies $f_i(x, y) = 1$ except possibly when x, y are the values of its children. For $r, s \in [k]$ let $E^{r,s}$ be the set of all such inputs I such that for all $j \in \{4, 5, 6, 7\}$, $v_{2j}^I = r$ and $v_{2j+1}^I = s$ (i.e., each pair of sibling leaves have values r, s), and f_1 is identically 1 (so I is a YES instance). Thus I is determined by the values of its 6 middle nodes $\{2, 3, 4, 5, 6, 7\}$, so

$$|E^{r,s}| = k^6.$$

Let B be a nondeterministic thrifty branching program that solves $BT_2^4(k)$, and let Γ be the set of states of B which query one of the nodes 4, 5, 6, 7. We will show $|\Gamma| = \Omega(k^3)$.

For $r, s \in [k]$ let $\Gamma^{r,s}$ be the set of states of Γ that query $f_j(r, s)$ for some $j \in \{4, 5, 6, 7\}$. We will show

$$|\Gamma^{r,s}| + 1 \ge k/\sqrt{3}. \tag{26}$$

Since Γ is the disjoint union of $\Gamma^{r,s}$ for all $r, s \in [k]$, it will follow that $|\Gamma| = \Omega(k^3)$ as required.

For each $I \in E^{r,s}$ let $\mathcal{C}(I)$ be an accepting computation of B on input I. Let t_1^I be the first time during $\mathcal{C}(I)$ that the root f_1 is queried. Let γ^I be be the last state in $\Gamma^{r,s}$ before t_1^I in $\mathcal{C}(I)$ (or the initial state γ_0 if there is no such state) and let δ^I be the first state in $\Gamma^{r,s}$ after t_1^I (or the ACCEPT state δ_{acc} if there is no such state).

We associate with each $I \in E^{r,s}$ a tuple

$$U(I) = (u, \gamma^I, \delta^I, x_1, x_2, x_3, x_4),$$

where $u \in \{1,2,3\}$ is a tag, and x_1, x_2, x_3, x_4 are in $[k]$ and are chosen so that $U(I)$ uniquely determines I (by determining the values of all 6 middle nodes). Specifically, $x_1 = v_i^I$, where i is the node queried by γ^I (or $i = 4$ if $\gamma^I = \gamma_0$).

Let $\mathcal{S}(I)$ denote the segment of the computation $\mathcal{C}(I)$ between γ^I and δ^I (not counting the action of the last state δ^I). This segment always queries the root $f_1(v_2, v_3)$, but does not query any of the nodes 4, 5, 6, 7 except γ^I may query node i.

Next we partition $E^{r,s}$ into three sets $E_1^{r,s}, E_2^{r,s}, E_3^{r,s}$ according to which of the nodes v_2, v_3 that $\mathcal{S}(I)$ queries. (The tag u tells us that I lies in set $E_u^{r,s}$.)

Let node $j \in \{2,3\}$ be the parent of node i (where i is defined earlier) and let $j' \in \{2,3\}$ be the sibling of j.

- $E_1^{r,s}$ consists of those inputs I for which $\mathcal{S}(I)$ queries neither v_2 nor v_3.
- $E_2^{r,s}$ consists of those inputs I for which $\mathcal{S}(I)$ queries $v_{j'}$.
- $E_3^{r,s}$ consists of those inputs I for which $\mathcal{S}(I)$ queries v_j but not $v_{j'}$.

To complete the definition of $U(I)$ we need only specify the meaning of x_2, x_3, x_4. The idea is that the segment $\mathcal{S}(I)$ will determine (using the definition of thrifty) the values of (at least) two of the six middle nodes, and x_1, x_2, x_3, x_4 will specify the remaining four values. We require that x_1, x_2, x_3, x_4 must specify the value of any node (except the root) that is queried during the segment, but the state that queries the node determines the values of its children.

In case the tag $u = 1$, the computation queries $f_1(v_2, v_3)$, and hence determines v_2, v_3, so x_1, x_2, x_3, x_4 specify the four values v_4, v_5, v_6, v_7.

In case $u = 2$, the computation queries $f_{j'}$ at the values of its children, so x_1, x_2, x_3, x_4 do not specify the values of these children, but instead specify v_2, v_3.

In case $u = 3$, x_1, x_2, x_3, x_4 do not specify the value of the sibling of node i and do not specify $v_{j'}$, but do specify v_j and the values of the other level 2 nodes.

Claim *If $I, J \in E^{r,s}$ and $U(I) = U(J)$, then $I = J$.*

Inequality (26) (and hence the theorem) follows from the claim, because if $|\Gamma^{r,s}| + 1 < k/\sqrt{3}$ then there would be fewer than k^6 choices for $U(I)$ as I ranges over the k^6 inputs in $E^{r,s}$.

To prove the claim, suppose $U(I) = U(J)$ but $I \neq J$. Then we can define an accepting computation of input I which violates the definition of thrifty. Namely follow the computation $\mathcal{C}(I)$ up to γ^I. Now follow the segment of $\mathcal{C}(J)$ between γ^I and δ^I, and complete the computation by following $\mathcal{C}(I)$. Notice that the segment of $\mathcal{C}(J)$ never queries any of the nodes 4, 5, 6, 7 except for v_i, and $U(I) = U(J)$ (together with the definition of $E^{r,s}$) specifies the values of the other nodes that it queries. However, since $I \neq J$, this segment of $\mathcal{C}(J)$ with input I will violate the definition of thrifty while querying at least one of the three nodes v_1, v_2, v_3. ∎

6 Conclusion

The Thrifty Hypothesis states that thrifty branching programs are optimal among k-way BPs solving $FT_d^h(k)$ (the tree evaluation problem for balanced degree d trees of height h). This implies that the black pebbling method is optimal for the deterministic case. Proving this would separate **L** from **P** (Corollary 3.3). Even disproving the hypothesis would be interesting, since it would show that one can improve upon this obvious application of pebbling.

Open Problem 1 Prove or disprove the Thrifty Hypothesis.

Corollary 5.2 gives tight lower bounds for $FT_d^h(k)$ for trees of height 3, thus proving the Thrifty Hypothesis for this case. The next important step is to extend these bounds to height 4 trees. The upper bound given in Theorem 5.1 (15) for the height 4 function problem $FT_d^4(k)$ for deterministic BPs is $O(k^{3d-2})$. If we could match this with a similar lower bound when $d = 4$ (e.g., by using a variation of the state sequence method in Section 5.2) this would yield $\Omega(k^{10})$ states for the function problem and hence (by Lemma 2.3) $\Omega(k^9)$ states for the Boolean problem $BT_4^4(k)$. Since the input length $n = O(k^4 \log k)$, this would break the Nečiporuk $\Omega(n^2)$ barrier for branching programs (see Section 5.1).

Open Problem 2 Establish the complexity of deterministic branching programs solving the tree evaluation problem for height 4 trees.

For nondeterministic BPs, the upper bound given by Theorem 5.1 for the Boolean problem for height 4 trees is $O(k^{2d-1})$. This comes from the upper bound on fractional pebbling given in Theorem 4.4, which we suspect is optimal. For $h = 4$ and degree $d = 3$, the corresponding lower bound for nondeterministic BPs for $BT_3^4(k)$ would be $\Omega(k^5)$. Since the input length $n = O(k^3 \log k)$, a proof would break the Nečiporuk $\Omega(n^{3/2})$ barrier for nondeterministic BPs.

Open Problem 3 Establish the complexity of nondeterministic branching programs solving the tree evaluation problem for height 4 trees.

The next two problems seem to be more accessible than the first three.

Open Problem 4 Improve Theorem 4.4 to get exact bounds on the number of pebbles required to fractionally pebble T_d^h, preferably with a direct proof.

The preceding problem is important since we conjecture that fractional pebbling algorithms yield optimal nondeterministic thrifty algorithms for tree evaluation (and indeed optimal with "thrifty" omitted).

Open Problem 5 Generalize Theorem 5.15 to get good lower bounds for nondeterministic thrifty BPs solving $BT_2^h(k)$ for $h > 4$.

The proof of Theorem 5.11, which states that deterministic thrifty BPs require at least k^h states to solve $BT_2^h(k)$, is taken from Wehr [Weh10]. That paper also proves the same lower bound for the more general class of "less-thrifty" BPs, which are allowed to query $f_i(a, b)$ provided that either (a, b) correctly specify the values of both children of i, or neither a nor b is correct.

Wehr [Weh10] also calculates $(k + 1)^h$ as the exact number of states required to solve $FT_2^h(k)$ using the black pebbling method, and proves that this number cannot be beaten by any k-way deterministic BP when $h = 2$. In fact, we have not been able to beat this BP upper bound by even one state, for any h and any k using any method.

Open Problem 6 Prove or disprove the hypothesis that for all $h \geq 2$ and for all sufficiently large k, every deterministic BP solving $FT_2^h(k)$ requires at least $(k + 1)^h$ states.[14]

In Wehr [Weh11], Wehr generalizes the tree evaluation problem to the DAG evaluation problem DE_G^k for each rooted DAG G and $k \geq 2$, and proves that every thrifty deterministic BP solving DE_G^k has at least k^p states, where p is the black pebble cost of G (i.e., the minimum number of pebbles required to black-pebble the root of G). This generalizes our Theorem 5.11, which applies to the case that G is a balanced binary tree. (Wehr uses his result to prove an exponential lower bound on the size of semantic incremental branching programs solving GEN, answering an open question in Gál et al. [GKM08].)

This suggests generalizing the thrifty hypothesis to rooted DAGs. This would imply that for each rooted DAG G with black pebble cost p, every deterministic k-way BP solving DE_G^k has $\Omega(k^p)$ states, where the constant implied by Ω depends on G. The obvious deterministic k-way BPs coming from the black pebbling algorithm for G that uses the minimum number of pebbles p have $\Theta(k^p)$ states. We do not know how to do better than this for any G.

14. Wehr [Weh12] has shown that $(k+1)^h$ is exactly optimal among deterministic thrifty read-once BPs solving $FT_2^h(k)$. The upper bound comes from black pebbling.

Open Problem 7 Find a rooted DAG G and a family $\langle B_k \rangle$ of deterministic k-way BPs, where B_k solves DE_G^k and has $o(k^p)$ states.

Acknowledgments

James Cook played a helpful role in the early parts of this research.

References

[adHei79] F. M. auf der Heide. 1979. *A Comparison between Two Variations of a Pebble Game on Graphs*. Master's thesis. Fakultät für Mathematik, Universität Bielefeld.

[BM12] P. Beame and P. McKenzie. 2012. A note on Nečiporuk's method for nondeterministic branching programs. http://www.cs.washington.edu/homes/beame/papers/neci.pdf.

[BSSV03] P. Beame, M. Saks, X. Sun, and E. Vee. 2003. Time-space tradeoff lower bounds for randomized computation of decision problems. *J. ACM* 50, 154–195. DOI: https://doi.org/10.1145/636865.636867.

[BC82] A. Borodin and S. A. Cook. 1982. A time–space tradeoff for sorting on a general sequential model of computation. *SIAM J. Comput.* 11, 2, 287–297. DOI: https://doi.org/10.1137/0211022.

[BRS93] A. Borodin, A. Razborov, and R. Smolensky. 1993. On lower bounds for read-k-times branching programs. *Comput. Complex.* 3, 1–18. DOI: https://doi.org/10.1007/BF01200404.

[BCM+09a] M. Braverman, S. A. Cook, P. McKenzie, R. Santhanam, and D. Wehr. December 15–17. 2009a. Fractional pebbling and thrifty branching programs. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2009*. IIT Kanpur, India. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 109–120. DOI: https://doi.org/10.4230/LIPIcs.FSTTCS.2009.2311.

[BCM+09b] M. Braverman, S. A. Cook, P. McKenzie, R. Santhanam, and D. Wehr. August 24–28. 2009b. Branching programs for tree evaluation. In *Mathematical Foundations of Computer Science 2009, 34th International Symposium, MFCS 2009, Proceedings*. Novy Smokovec, High Tatras, Slovakia. Lecture Notes in Computer Science, Vol. 5734. Springer, 175–186. DOI: https://doi.org/10.1007/978-3-642-03816-7_16.

[Coo74] S. A. Cook. 1974. An observation on time–storage trade off. *J. Comput. Syst. Sci.* 9, 3, 308–316. DOI: https://doi.org/10.1016/S0022-0000(74)80046-2.

[CS76] S. A. Cook and R. Sethi. 1976. Storage requirements for deterministic polynomial time recognizable languages. *J. Comput. Syst. Sci.* 13, 1, 25–37. DOI: https://doi.org/10.1016/S0022-0000(76)80048-7.

[EIRS01] J. Edmonds, R. Impagliazzo, S. Rudich, and J. Sgall. 2001. Communication complexity towards lower bounds on circuit depth. *Comput. Complex.* 10, 3, 210–246.

[GKM08] A. Gál, M. Koucký, and P. McKenzie. 2008. Incremental branching programs. *Theory Comput. Syst.* 43, 2, 159–184.

[Gol08] O. Goldreich. 2008. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press.

[GLT80] J. R. Gilbert, T. Lengauer, and R. E. Tarjan. 1980. The pebbling problem is complete in polynomial space. *SIAM J. Comput.* 9, 3, 513–524. DOI: https://doi.org/10.1137/0209038.

[HW93] J. Håstad and A. Wigderson. 1993. Composition of the universal relation. In *Proceedings of the Advances in Computational Complexity Theory, AMS-DIMACS 13*, 119–134.

[KRW95] M. Karchmer, R. Raz, and A. Wigderson. 1995. Super-Logarithmic depth lower bounds via direct sum in communication complexity. *Comput. Complex.* 5, 191–204.

[Kla85] M. M. Klawe. 1985. A tight bound for black and white pebbles on the pyramid. *J. ACM* 32, 1, 218–228. DOI: https://doi.org/10.1145/2455.214115.

[LT80] T. Lengauer and R. E. Tarjan. 1980. The space complexity of pebble games on trees. *Inf. Process. Lett.* 10, 4–5, 184–188. DOI: https://doi.org/10.1016/0020-0190(80)90136-2.

[Lou79] M. Loui. 1979. *The Space Complexity of Two Pebble Games on Trees*. Technical Report LCS 133. MIT, Cambridge, Massachussetts.

[Mah07] M. Mahajan. 2007. Polynomial size log depth circuits: between $\mathbf{NC}^1$ and $\mathbf{AC}^1$. *Bull. EATCS* 91, 30–42.

[Neč66] È. Nečiporuk. 1966. On a boolean function. *Doklady of the Academy of the USSR* 169, 4, 765–766. (English translation in *Soviet Mathematics Doklady 7*, 4, 999–1000.)

[Nor09] J. Nordström. 2009. New wine into old wineskins: A survey of some pebbling classics with supplemental results. http://people.csail.mit.edu/jakobn/research/.

[PH70] M. S. Paterson and C. E. Hewitt. 1970. Comparative schematology. In *Record of the Project MAC Conference on Concurrent Systems and Parallel Computation*. ACM, 119–127. DOI: https://doi.org/10.1145/1344551.1344563.

[Pud87] P. Pudlák. 1987. The hierarchy of boolean circuits. *Comput. Artif. Intell.* 6, 5, 449–468.

[Raz91] A. Razborov. 1991. Lower bounds for deterministic and nondeterministic branching programs. In *Proceedings of the 8th International Symposium on Fundamentals of Computation Theory*. 47–60. DOI: https://doi.org/10.1007/3-540-54458-5_49.

[Sud78] I. H. Sudborough. 1978. On the tape complexity of deterministic context-free languages. *J. ACM* 25, 3, 405–414. DOI: https://doi.org/10.1145/322077.322083.

[Tai05] M. Taitslin. 2005. An example of a problem from PTIME and not in NLogSpace. In *Proceedings of the Tver State University*. *Appl. Math.* 6, 12, 5–22.

[Weg87] I. Wegener. 1987. *The Complexity of Boolean Functions*. Wiley-Teubner Series in Computer Science. B. G. Teubner & John Wiley, Stuttgart.

[Weg00] I. Wegener. 2000. *Branching Programs and Binary Decision Diagrams*. SIAM Monographs on Discrete Mathematics and Applications. SIAM, Philadelphia, PA.

[Weh10] D. Wehr. 2010. Pebbling and branching programs solving the tree evaluation problem. MSc research paper, Department of Computer Science, University of Toronto. February. arXiv:1002.4676.

[Weh11] D. Wehr. 2011. Lower bound for deterministic semantic-incremental branching programs solving GEN. arXiv:1101.2705.

[Weh12] D. Wehr. 2012. Exact size of the smallest min-depth branching programs solving the tree evaluation problem. http://www.cs.toronto.edu/ wehr/ research_docs/thrifty_and_read-once_exact_size.pdf.

PART V

THE BERKELEY NOTES

Cook's Berkeley Notes

Bruce M. Kapron

"A survey of classes of primitive recursive functions" gives a summary of material presented by Steve Cook in the University of California, Berkeley, course Math 290, Sect. 14, January 1967. The notes were carefully typed up and copies were distributed at the University, but they were never formally published.

The notes present a survey of subrecursive function classes, and classes of relations based on these classes, with a particular focus on low levels of Grzergorczk's hierarchy of primitive recursive functions [Grz53]. While much of the survey is presented in a style more familiar from recursion theory—in particular the use of inductive definitions of classes of functions and relations—there is also a focus on how these classes correspond to classes characterized by resource-bounded Turing machines (and related models). This latter focus is already present in contemporary work, such as Cobham's inductive characterization of the poly-time computable functions [Cob65] and Ritchie's of those computable in linear space [Rit63].

Most significantly, with respect to the P vs NP problem, Cook discusses the relationship between Cobham's class $\mathcal{L}$ of polynomial time functions and Bennett's class, denoted by $\mathcal{L}^+$ in the notes, of *extended positive rudimentary functions*, defined in his (unpublished) thesis [Ben62]. In particular, the following facts are observed or proved by Cook:

1. The class of relations corresponding to $\mathcal{L}^+$ is equal to that obtained by closing the relations in $\mathcal{L}$ under bounded existential quantification.
2. $\mathcal{L}^+$ corresponds to those functions Turing-machine computable in nondeterministic polynomial time.
3. $\mathcal{L} \subseteq \mathcal{L}^+$

Finally, he conjectures that the inclusion (3) is proper but observes that it "may be difficult to prove".

While Cobham's class $\mathcal{L}$ is quite familiar in current complexity theory and corresponds to the class FP of functions computable in polynomial time[1], Bennett's class is not as well-known and requires some elucidation. As described in Section 1 of the notes, for any class of relations, there is an associated class of functions, namely those whose graph is in the class of relations and that are bounded by a function "from some appropriate class." In this way, $\mathcal{L}^+$ is defined to be the class of functions that are poly-bounded and whose associated relation is extended positive rudimentary. We will not define this latter notion as it is quite technical and not needed for the current discussion. In particular, from (1) we know that this class is equal to the bounded existential closure of P, which is in fact equal to NP, characterized as those problems for which the existence of a solution may be verified in polynomial time.

We may now consider (2). While there is some ambiguity here as the notes do not give a formal definition, if we use the fact that in Bennett's thesis and in the Berkeley notes the functions considered are total, Cook's notion of nondeterministic function computability corresponds to the class NPSV_t of total single-valued functions computable in nondeterministic polynomial time, a class later defined formally by Book, Long, and Selman [BLS84]. So the claim in (2) is that the functions in NPSV_t are exactly those that are poly-bounded and have a graph that is in NP, a fact that may be easily verified.

In summary, (1) and (2) give an equivalence between the extended positive rudimentary relations and the class NP and between the associated class of functions and the class now known as NPSV_t. A missing link here is a characterization of the class NP via nondeterministic polynomial time Turing machines. This is most likely due to the focus on computability of (total) functions. Rather than giving a separate machine characterization for deciding relations (or, in more modern terminology, language recognition), in the notes this notion is reduced to computability of the characteristic function of the relation, as described in Section 1. Recall that the characteristic function of a relation takes the value 1 for an input when the relation holds for this input and 0 otherwise. However, the class of relations having a characteristic function in NPSV_t is *not* equal to NP. A characterization of NP may be obtained through the use of *partial* characteristic functions that, rather than taking the value 0, are undefined in case the relation does not hold for given inputs. With this modification, we can show that NP is the class of relations corresponding to

1. Note that notation FP is ambiguous and may (say in the setting of search complexity) denote those *multi-valued* functions for which there is a polynomial time algorithm that for any input returns some correct value.

NPSV, the class of partial single-valued functions computable in nondeterministic polynomial time.

We are left with the question of how Cook's conjecture relates to the P vs NP problem. Given the mappings between classes given above, (3) states that $\mathsf{FP} \subseteq \mathsf{NPSV}_t$, and the conjecture is that this inclusion is proper. We can use the fact that $\mathsf{NPSV}_t = \mathsf{FP}^{\mathsf{NP}\cap\mathsf{coNP}}$ [HHN+93], where the latter denotes the class of functions that are computable in polynomial time using an oracle for a problem in $\mathsf{NP} \cap \mathsf{coNP}$. This means that if $\mathsf{P} = \mathsf{NP}$, then $\mathsf{FP} = \mathsf{NPSV}_t$ since in this case $\mathsf{NP} \cap \mathsf{coNP} = \mathsf{P}$. So Cook's conjecture certainly implies that $\mathsf{P} \neq \mathsf{NP}$, and in fact implies the ostensibly stronger conjecture that $\mathsf{P} \neq \mathsf{NP} \cap \mathsf{coNP}$. Furthermore, if $\mathsf{FP} = \mathsf{FP}^{\mathsf{NP}\cap\mathsf{coNP}}$, then $\mathsf{P} = \mathsf{P}^{\mathsf{NP}\cap\mathsf{coNP}}$ because the assumption implies that the characteristic function of any language in $\mathsf{P}^{\mathsf{NP}\cap\mathsf{coNP}}$ is in FP, and so the language itself is in P. Now by a result of Brassard [Bra79], $\mathsf{P}^{\mathsf{NP}\cap\mathsf{coNP}} = \mathsf{NP} \cap \mathsf{coNP}$, so $\mathsf{FP} = \mathsf{FP}^{\mathsf{NP}\cap\mathsf{coNP}}$ implies $\mathsf{P} = \mathsf{NP} \cap \mathsf{coNP}$, and we can conclude that Cook's conjecture is in fact equivalent to the conjecture that $\mathsf{P} \neq \mathsf{NP} \cap \mathsf{coNP}$.

The fact that we obtain this sort of equivalence is again due the setting of total functions. In the partial case, it is known that $\mathsf{PF} = \mathsf{NPSV}$ iff $\mathsf{P} = \mathsf{NP}$, where PF is the class of partial functions computable in polynomial time [SMB83].

It is interesting to note that around the same time, Edmonds [Edm65a, Edm65b] addressed the question of the equality of P and $\mathsf{NP} \cap \mathsf{coNP}$ from the perspective of obtaining "good algorithms" for combinatorial optimization problems having "good characterizations," although he so did in a somewhat informal setting.

Although the notes do not directly address the P vs NP problem, they still introduce a number of steps of great importance to the subsequent theory, most notably the question of the power of nondeterminism for polynomial-time Turing machines, as well as the connection between the characterization of NP in terms of efficient verification of solutions and nondeterministic polynomial time computability (modulo the missing link described above.)

Apart from their relevance to the P vs NP problem, the notes also contain another well-known question relevant to subsequent work in complexity. Among the open questions appearing in Section 6 is whether the *context-sensitive relations* are closed under complementation. As Kuroda [Kur64] had shown that these correspond to the relations decidable in nondeterministic linear space, the question was resolved by the closure of nondeterministic space under complement, a result obtained independently by Immerman [Imm88] and Szelepcsényi [Sze88].

A Survey of Classes of Primitive Recursive Functions

Stephen A. Cook

1 Basic Notions

All functions considered here take tuples of non-negative integers into non-negative integers. We use the notation $\underline{x}$ for $x_1, \ldots, x_p$, where p is usually not specified.

Relations vs. functions If $\mathcal{F}$ is a class of functions, then $\mathcal{R}_{\mathcal{F}}$, the $\mathcal{F}$-relations, is the class of relations whose characteristic function is in $\mathcal{F}$. Conversely, if $\mathcal{R}$ is a class of relations, then it is possible to associate a class $\mathcal{F}$ of functions with $\mathcal{R}$ by saying $f \in \mathcal{F}$ iff first the relation $y = f(\underline{x})$ is in $\mathcal{R}$, and second f is bounded by a function from some appropriate class. For example, in case $\mathcal{R}$ is the class of constructive arithmetic relations the appropriate class of bounding functions turns out to be the class of polynomials. If we start with a class $\mathcal{F}$ of functions which includes the function $x = y$ and is closed under substitution and limited minimalization, then the class of functions associated with $\mathcal{R}_{\mathcal{F}}$ is again precisely $\mathcal{F}$, provided the class of bounding functions is chosen to be cofinal (see below) with $\mathcal{F}$. On the other hand, suppose we start with a class $\mathcal{R}$ of relations which includes the identity relation and is closed under explicit transformation and the Boolean operations. If we pass to the associated class $\mathcal{F}$ of functions using any bounding class which includes the constant function $\mathcal{I}$, then $\mathcal{R}_{\mathcal{F}}$ is precisely $\mathcal{R}$.

Cofinal Classes Two classes $\mathcal{F}_0$ and $\mathcal{F}_1$ are cofinal if for every $f \in \mathcal{F}_i$ there is a $g \in \mathcal{F}_{i-1}$ such that $f(\underline{x}) \leq g(\underline{x})$ for all $\underline{x}$ $(i = 0, 1)$.

Unpublished. Appears as a preprint https://eccc.weizmann.ac.il/report/2017/001/

Explicit Transformation A class $\mathcal{F}$ of functions is closed under explicit transformation if, whenever $g \in \mathcal{F}$, there is an $f \in \mathcal{F}$ such that $f(\underline{x}) = g(\underline{t})$ holds identically, where each t_i is either an x_j or a constant. For example, perhaps $f(x_1, x_2, x_3) = g(x_3, 2, x_3, x_1)$. Similarly for classes of relations.

Substitution $\mathcal{F}$ is closed under substitution if it is closed under both explicit transformation and composition.

Boolean Operations The three Boolean operations are negation (complementation), finite conjunction (intersection), and finite disjunction (union). These apply to relations.

Bounded (i.e. limited) quantification The two operations $\exists_{\leq}$ and $\forall_{\leq}$ apply to relations, and are defined as follows: $(\exists_{\leq} R)(\underline{x}, y)$ holds iff $R(\underline{x}, z)$ holds for some $z \leq y$, and $(\forall_{\leq} R)(\underline{x}, y)$ holds iff $R(\underline{x}, z)$ holds for all $z \leq y$.

Bounded (i.e. limited) recursion f is defined from g, h, k by limited recursion provided the following holds for all $\underline{x}, y$.

$$f(\underline{x}, 0) = g(\underline{x})$$
$$f(\underline{x}, y+1) = g(\underline{x}, f(\underline{x}, y), y)$$
$$f(\underline{x}, y) \leq k(\underline{x}, y)$$

***m*-adic notation** (We always assume $m \geq 2$ when speaking of m-adic notation). The m-adic notation for the positive integer n is the unique string $d_k d_{k-1} \ldots d_0$ of digits from the alphabet $\{1, 2, \ldots, m\}$ such that

$$n = \sum_{l=0}^{k} d_i m^i.$$

The m-adic notation for 0 is the empty string. Switching back and forth from m-adic to m-ary (radix) notation involves very little computation. m-adic (as opposed to m-ary) notation sets up a one-one correspondence between strings and non-negative integers. The ordering induced on strings by their m-adic value is the one determined first by length, and among strings of the same length, the ordering is lexicographical.

Bounded (i.e. limited) recursion on notation f is defined from $g, h_1, \ldots, h_m$, and k by limited recursion on (m-adic) notation provided the following hold for all $\underline{x}, y$.

$$f(\underline{x}, 0) = g(\underline{x})$$
$$f(\underline{x}, y * i) = h_i(\underline{x}, f(\underline{x}, y), y), i = 1, 2, \ldots, m$$
$$f(\underline{x}, y) \leq k(\underline{x}, y)$$

Here $*$ is the m-adic concatenation function.

Subpart quantification The two operations $\exists^m$ $\forall^m$ apply to relations and are defined as follows. $(\exists^m R)(\underline{x}, y)$ holds iff $R(\underline{x}, z)$ holds for some z whose m-adic notation is a consecutive substring (possibly all or empty) of the m-adic notation for y, and $(\forall^m R)(\underline{x}, y)$ holds iff $R(\underline{x}, z)$ holds for all such z.

2 The Grzegorczyk Hierarchy

(See Grzegorczyk [Grz53]) This is a sequence $\mathcal{E}^0 \subseteq \mathcal{E}^1 \subseteq \ldots$ of classes of functions whose union is precisely the class of primitive recursive functions. First let us define a sequence $\xi_0(x,y), \xi_1(x,y), \ldots$ of functions by

$$\xi_0(x,y) = y+1$$

$$\xi_{n+1}(x,0) = \begin{cases} x & \text{if } n = 0 \\ 0 & \text{if } n = 1 \\ 1 & \text{if } n > 1 \end{cases} \quad \left[\begin{array}{l}\text{DEFINE SUCH THAT FUNCTION IS} \\ \text{ALWAYS THERE}\end{array}\right]$$

$$\xi_{n+1}(x,y+1) = \xi_n(x, \xi_{n+1}(x,y)).$$

Note that $\xi_1(x,y) = x+1$, $\xi_2(x,y) = xy$ and $\xi_3(x,y) = x^y$. Then $\mathcal{E}^n$ can be defined as the least class including the functions $x+1$ and ξ_n, and closed under substitution and limited recursion. Grzegorczyk showed that $\mathcal{E}^3$ is just $\mathcal{E}$, the class of elementary functions of Kalmár. Each class is closed under limited minimalization and (at least for $n \geq 2$) limited recursion on notation, and the class of $\mathcal{E}^n$ relations is closed under the Boolean operations and bounded quantification for all n. $\mathcal{E}^{n+1}$ contains $\mathcal{E}^n$ properly for all n, and the class of $\mathcal{E}^{n+1}$ relations contains the class or $\mathcal{E}^n$ relations, at least for $n \geq 2$.

Theorem 1 was proved by Ritchie [Rit63], and theorem 2 was stated by Cobham [Cob65] and can be proved by Ritchie's methods. The theorems provide interesting characterizations for the functions of $\mathcal{E}^n$ in terms of their computation time and storage requirements.

Notation If Z is a Turing machine which computes a function $f(\underline{x})$ in m-adic notation, $\tau_Z(\underline{x})$ (the tape function for Z) is the number of tape squares used by Z in evaluating f at $\underline{x}$, and $\sigma(\underline{x})$ (the time function for Z) is the number of steps required by Z to evaluate f at $\underline{x}$.

Theorem 1 *$f(\underline{x}) \in \mathcal{E}^2$ iff there are constants C_1 and C_2 and a Turing machine Z which computes f in m-adic notation such that*

$$\tau_Z(\underline{x}) \leq C_1(\sum_i \ell(x_i)) + C_2 \qquad *$$

for all $\underline{x}$. (Here $\ell(x_i)$ is the length of the m-adic notation for x_i).

Theorem 2 *(a) If $n \geq 3$, then $f(\underline{x}) \in \mathcal{E}^n$ iff there is a Turing Machine Z which computes f and a function $g \in \mathcal{E}^n$ such that $\tau_Z(\underline{x}) \leq g(\underline{x})$ for all $\underline{x}$.*

(b) Same as (2), with $\sigma_Z(\underline{x})$ replacing $\tau_Z(\underline{x})$.

Remark 2 Of course Theorem 2 is equally valid if the functions g are chosen from some class of functions cofinal with $\mathcal{E}^n$ instead of $\mathcal{E}^n$ itself. For example, the class

$$\{\xi_{n_1}(\max \underline{x}, c_1) + c_2 \mid c_1, c_2 \text{ positive integers}\}$$

is cofinal with $\mathcal{E}^n$, $n \geq 0$.

Remark 3 **Shepherdson-Sturgis machines**. Theorems 1 and 2 remain valid when other computer models are used besides Turing machines, provided τ_Z and σ_Z are defined properly; and in particular the Turing machines may have several taps and several read/write heads per tape. The unlimited register machines of Shepherdson and Sturgis [SS63] will do as the computer model, provided $\tau_Z(\underline{x})$ is taken to be the length of the m-adic notation of the maximum number occurring in any register during the course of the computation with input $\underline{x}$. Then $(*)$ in Theorem 1 is equivalent to requiring that the numbers in each register of the machine be bounded by polynomials[1] in $\underline{x}$, which in turn is equivalent (since $\mathcal{E}^2$ is cofinal with the polynomials) to requiring that the members of each register be bounded by some member of $\mathcal{E}^2$. In fact, in general for $n \geq 2$, $\mathcal{E}^n$ consists exactly of those functions computable by some Shepherdson-Sturgis machine in which *the numbers in all* registers are bounded by some member of $\mathcal{E}^n$.

Remark 4 There is no known characterization of $\mathcal{E}^2$ in terms of $\sigma_Z(\underline{x})$ analogous to the characterizations of $\mathcal{E}^n$ for $n > 2$ state in theorem 2. This is one reason for introducing the class $\mathcal{L}$ defined in the next section.

3 Computation Time and Limited Recursion on Notation

Cobham [Cob65] introduced the class $\mathcal{L}$ of functions which is defined in terms of computation time. A function $f(\underline{x})$ is in $\mathcal{L}$ iff there is a Turing machine Z which computes f in m-adic notation and a polynomial $P(\underline{t})$ such that

$$\sigma_Z(x_1, \ldots, x_n) \leq P(\ell(x_1), \ldots, \ell(x_n))$$

for all $\underline{x}$. Cobham stated the following characterization of $\mathcal{L}$.

1. Robert Elschlager points out that the numbers can always be bounded by $\max \underline{x}$ if the function computed is a characteristic function

Theorem 3 *$\mathcal{L}$ is the least class of functions containing $S_i(x)$, $1 = 1, \ldots, m$, $x^{\ell(y)}$ and closed under substitution and limited recursion on notation. Here $S_i(x)$ is $x * i$ (the ith m-adic successor function.)*

The proof is similar to the proof of theorem 1. The class $\mathcal{L}$ is independent of the choice of m since a Turing machine can convert from m-adic to n-adic notation sufficiently rapidly.

The class $\mathcal{L}$, characterizable in terms of computation time requirements, is a natural analog of $\mathcal{E}^2$, characterizable in terms of storage requirements. Note the parallel between theorems 1 and 3, contrasting limited recursion with limited recursion on notation.

$\mathcal{E}^2$ does not contain $\mathcal{L}$ because the function $x^{\ell(y)}$ in $\mathcal{L}$ grows too fast to be in $\mathcal{E}^2$, but it is not known whether $\mathcal{L} \supseteq \mathcal{E}^2$. Cobham points out that the function $f(n) =$ the nth prime is known to be in $\mathcal{E}^2$, but suggests that it is too time consuming to compute to be in $\mathcal{L}$, that $\mathcal{L}$ and $\mathcal{E}^2$ are probably incomparable. Similarly, it is a good guess that the $\mathcal{L}$-relations and $\mathcal{E}^2$-relations are incomparable, or at least the former should not include the latter.

Extended Positive Rudimentary Functions, $\mathcal{L}^+$. These were introduced by Bennett [Ben62], p. 67, and can be defined as the class of functions associated with the extended positive rudimentary relations (see section 5), where the bounding functions are taken to be those of the form $x^{(\ell(y))^n + c}$ for arbitrary constants n and c. Bennett states that this class of functions, which we might call $\mathcal{L}^+$, is closed under substitution and limited recursion on notation, and since $\mathcal{L}^+$ certainly contains $x + 1$ and $x^{\ell(y)}$, we can conclude

$$\mathcal{L} \subseteq \mathcal{L}^+$$

Whereas $\mathcal{L}$ can be characterized as consisting of those functions whose Turing machine computation time is bounded by a polynomial in the lengths of the arguments, $\mathcal{L}^+$ has the same characterization except that we must allow the Turing machines to be non-deterministic. It seems likely that this non-determinism increases the computing power of the machine, but this may be difficult to prove.

4 The Ritchie Hierarchy

Ritchie [Rit63] introduced a sequence $\langle F_i \rangle$ of classes of functions satisfying

$$\mathcal{E}^2 \subseteq F_1 \subseteq F_2 \subseteq \cdots \subseteq \mathcal{E}^3$$

F_i consists of just those functions computable by a Turing machine Z whose tape function τ_Z satisfies

$$\tau_Z(\underline{x}) \leq f_{i-1}(K(\max \underline{x}, 1)) \qquad (**)$$

for some constant K, where $f_0(x) = x, f_1(x) = 2^x$, and in general $f_{i+1}(x) = 2^{f_i(x)}$. For $i \geq 2$, $(**)$ is equivalent to requiring $\tau_Z(\underline{x})$ be bounded by some member of F_{i-1}, since for each i the class of functions $\{f_i(K(\max \underline{x}, 1)) \mid K \text{a positive integer}\}$ is cofinal with F_i. Because of the latter characterization, Ritchie called the functions in F_i the "predicatively computable functions"; i.e., one always knows that the tape function for a member of F_i is bounded by some member of the class F_{i-1}, which has already been "obtained".

Since the class $\{f_i(K(\max \underline{x}, 1)) \mid i, K \text{positive integers}\}$ is cofinal with $\mathcal{E}^3$, it follows from theorem 2 that

$$\bigcup_{i=1}^{\infty} F_i = \mathcal{E}^3 \quad \text{(elementary functions)}$$

Also, by theorem 1, each F_i contains $\mathcal{E}^2$ properly. Ritchie used a diagonal argument to show that the F_i-relations are properly contained in the F_{i+1}-relations for all i.

Each class F_i is closed under explicit transformation, but none is closed under composition or limited recursion. Bennett [Ben62], p. 74 points out the following characterization of F_1 follows from Ritchie's work: $f \in F_i$ iff $f(\underline{x}) = g(\underline{x}, f_1(K \max \underline{x})$ for some $g \in \mathcal{E}^2$ and integer K

5 Other Classes

All of the following classes except the context-sensitive relations (and languages) are discussed by Bennett [Ben62]. Smullyan [Smu61] introduced the m-rudimentary relations and the constructive arithmetic relations.

Notation $x \overset{m}{*} y$ is the number whose m-adic notation is the concatenation of the m-adic notations for x and y.

The Strictly m-rudimentary relations are the least class of containing the three place relation $x \overset{m}{*} y = z$ and closed under explicit transformation, the Boolean operations, and subpart quantification. Bennett states that these are distinct for each m.

The positive *m*-rudimentary relations are the least class containing $x \overset{m}{*} y = z$ and closed under explicit transformation, conjunction, disjunction, subpart quantification, and $\exists_{\leq}$. Bennett shows these form the same class for each m.

The Strongly *m*-rudimentary relations are those relations R such that both R and $\neg R$ are positive m-rudimentary.

This class is independent of m and is closed under explicit transformation, the Boolean operations, and subpart quantification (Bennett).

The *m*-rudimentary relations are the least class of relations containing $x \overset{m}{*} y = z$ and closed under explicit transformation, the Boolean operations and bounded quantification.

The constructive arithmetic relations are the least class containing the two three-place relations $x + y = z$ and $x - y = z$ and closed under explicit transformation, the Boolean operations, and bounded quantification.

Bennett's main result in chapter I of [Ben62] is that this class is the same as the class of m-rudimentary relations for each m.

The extended positive *m*-rudimentary relations are those of the form

$$(\exists y \leq g_k(\underline{x}))R(\underline{x}, y)$$

where $R(\underline{x}, y)$ is positive m-rudimentary, k is an integer, and $g_k(\underline{x}) = m^{(\ell(\max \underline{x}))^k}$, $\ell(z)$ is the length of the m-adic notation for z.

Using Theorem 3 in section 3 and Bennett's discussion pp. 62-67 it is easily shown that this class is (independent of m) precisely the closure of Cobham's class of $\mathcal{L}$-relations under the operation $\exists_{\leq}$. The class is also closed under disjunction, conjunction, explicit transformation, and subpart quantification.

A context-sensitive language is the set of strings over some alphabet generated by a semi-Thue system of whose productions $u \to v$ satisfy $\ell(u) \leq \ell(v)$. Kuroda [Kur64] characterized these languages as those recognizable by some non-deterministic Turing machine whose tape function is bounded by some linear function of the length of the input string. This characterization suggests that a context sensitive *relation* be defined as one recognizable in the same way. Using m-adic notation we can consider these context sensitive relations to be relations on integers, and it is easy to see the resulting class of relations will not depend on m. Then by Theorem 1, we find that the $\mathcal{E}^2$ relations are a subclass of the context sensitive relations.

Spectra The spectrum of a formula of the first order predicate calculus with equality is the set of all cardinalities of its finite models. Bennett generalized the notion of spectrum of formula from such a theory to a many-sorted theory of types by

defining the spectrum of a formula from such a theory as the relation which is true on those types of integers which are the cardinalities of the basic domains of individuals for some finite model of the sentence. He denotes by $\mathcal{S}^n$ the class of spectra of n^{th} order formulas.

Bennett's main result is the following (p. 116, 125).

Theorem 4

(a) For each $n \geq 1$ and $m \geq 2$, $\mathcal{S}^{2n-1}$ is the class of relations of the form $(\exists y \leq g(\underline{x}))R(\underline{x}, y)$ where $g(\underline{x}) = f_n((\max(\underline{x}))^j)$ for some $j \geq 1$ (see Sec. 4 for f_n) and R is strictly m-rudimentary.

(b) The same as (4) except R may be chosen to be any extended positive rudimentary relation.

(c) For each $n \geq 1$, $\mathcal{S}^{2n}$ is the class of relations of the forms $(\exists y \leq g(\underline{x}))R(\underline{x}, y)$ where R is constructive arithmetic and $g(\underline{x}) = f_n((\max(\underline{x}))^j)$ for some $j \geq 1$.

(d) For each $n \geq 1$, $\mathcal{S}^n$ is a subclass of $\mathcal{S}^{n+1}$ and a proper subclass of $\mathcal{S}^{n+2}$.

(e) $\bigcup_{n=1}^{\infty} \mathcal{S}^n$ is the class of $\mathcal{E}^3$ (elementary)-relations.

(f) The F_1-relations are a subclass of $\mathcal{S}^3$, and for each $n \geq 2$, $\mathcal{S}^{2n-2}$ is a subclass of the F_n-relations, which in turn are a subclass of $\mathcal{S}^{2n+1}$. Moreover, for no n, $p \geq 1$ is $\mathcal{S}^p$ identical with the class of F_n-relations.

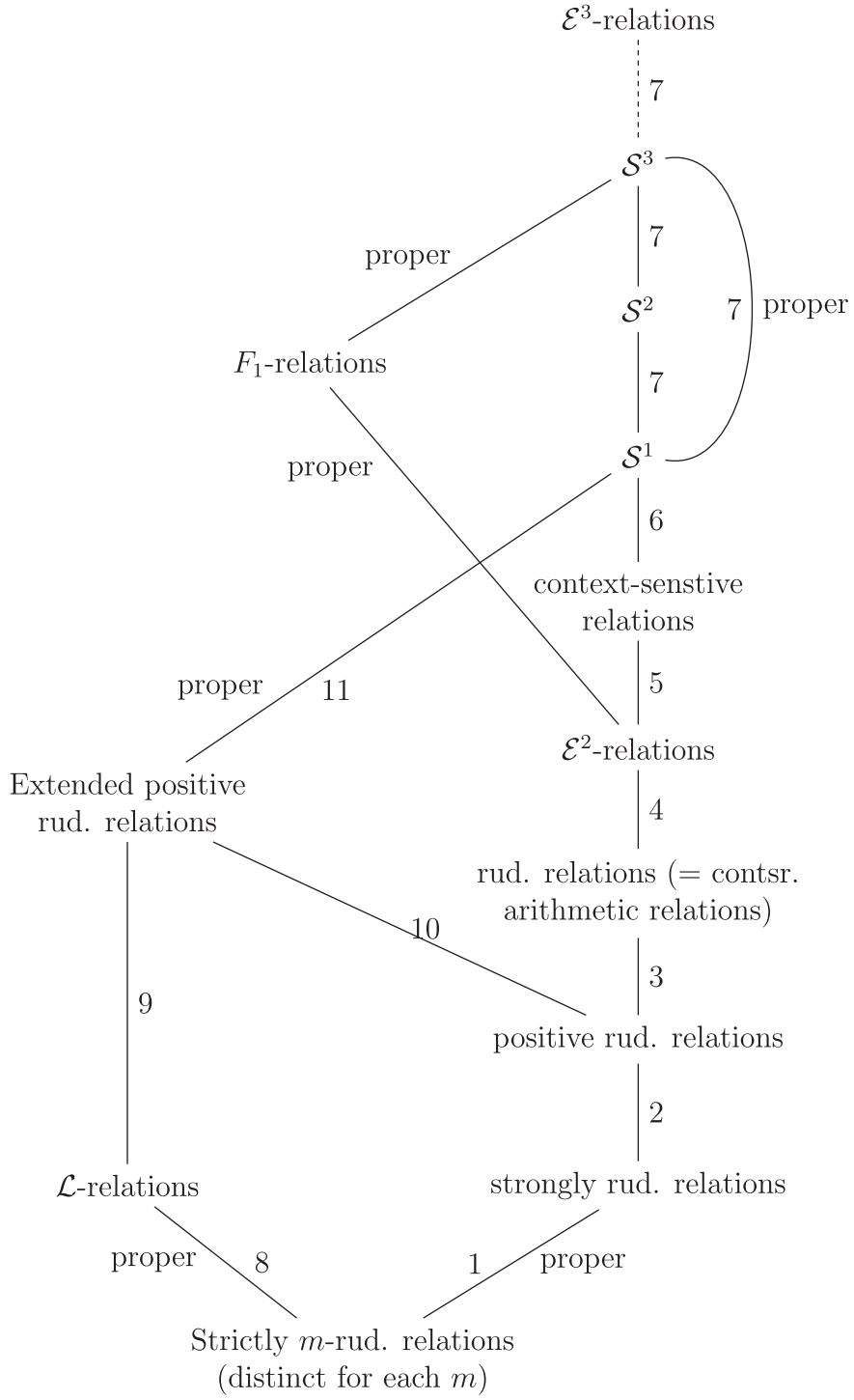
(g) The constructive arithmetic relations form a proper subclass of $\mathcal{S}^2$, and the extended positive rudimentary relations form a proper subclass of $\mathcal{S}^1$.

6 Summary of Facts and Open Questions

The chart on the next page indicates the inclusion relationships among most of the classes of relations discussed earlier. A line from one class to one above it indicates the higher class contains the lower. If the inclusion is known to be proper, the line is so labelled. The numbers on the lines refer to the following list of sources for the proofs of inclusion.

Sources

1, 8.	Stated by Bennett, p. 13
2, 3.	Bennett, p. 13 Immediate from definitions
4	Bennett, p. 75 Follows immediately form the definitions and the fact that $\mathcal{E}^2$-relations are closed under explicit transformation, the Boolean operations, and limited quantification
5	Kuroda [Kur64] and Theorem 1.
6	Theorem 4, part (4), Kuroda's characterization of context-sensitive languages, and an easy argument.
7,11	Theorem 4 (Bennett).
9,10	See under definition of extended positive rudimentary relations.


$\mathcal{E}^3$-relations
7
$\mathcal{S}^3$
7
proper
$\mathcal{S}^2$
7 proper
F_1-relations
7
$\mathcal{S}^1$
proper
6
context-senstive relations
proper
11
5
$\mathcal{E}^2$-relations
Extended positive rud. relations
4
rud. relations (= contsr. arithmetic relations)
10
3
9
positive rud. relations
2
$\mathcal{L}$-relations
strongly rud. relations
proper
8
1
proper
Strictly m-rud. relations (distinct for each m)

Closure under operations of relation classes All the classes of relations discussed previously are closed under explicit transformation, subpart quantification, disjunction, and conjunction. This follows from the definitions either directly or by easy arguments. The following table indicates which classes are known to be closed under negation, $\exists\ \leq$ and $\forall\ \leq$. It is tempting to argue that where a "?" appears the answer is probably most often "no", partly from intuition, and partly because plenty of techniques for proving positive results are known, but very few for proving negative results are known.

One of the more interesting questions is whether the positive m-rudimentary relations are closed under negation. If the answer is yes, then this class is the same as the class of constructive arithmetic relations, so $\mathcal{S}^{2n-1} = \mathcal{S}^{2n}$ for all $n \geq 1$, so $\mathcal{S}^n$ would be closed under negation for all n.

	negation	$\exists \leq$	$\forall \leq$	Source
$\mathcal{E}^n$-relations, $n \geq 0$	yes	yes	yes	Grzegorczck [Grz53]
F_n-relations, $n \geq 1$	yes	yes	yes	See section 4
$\mathcal{L}$-relations	yes	?	?	See section 3
Strictly m-rudimentary relations	yes	no	no	
Positive m-rudimentary relations	?	yes	?	Bennett, p. 13
Constructive arithmetic relations	yes	yes	yes	
Extended positive rudimentary relations	?	yes	?	Bennett, p. 62
Context sensitive relations	?	yes	yes	See definition, Sec. 5
$\mathcal{S}^n$, n odd	?	yes	yes	
$\mathcal{S}^n$, n even	yes	yes	yes	Bennett, p. 124

Functions Each of the classes $\mathcal{E}^n$, $n \geq 0$, F_n, $n \geq 1$, $\mathcal{L}$, $\mathcal{L}^+$ is closed under explicit transformation, composition, and limited recursion on notation. In addition, all except F_n and possibly $\mathcal{L}$ and $\mathcal{L}^+$ are closed under limited recursion.

References

[Ben62] J. H. Bennett. 1962. *On Spectra*. Ph.D. thesis. Princeton University.

[Cho59] N. Chomsky. 1959. On certain formal properties of grammars. *Inform. Control* 2, 137–167. DOI: https://doi.org/10.1016/S0019-9958(59)90362-6.

[Cob65] A. Cobham. 1965. The intrinsic computational difficulty of functions. In *Proceedings of the 1964 International Congress for Logic, Methodology, and Philosophy of Science*. North-Holland, Amsterdam, 24–30.

[Grz53] A. Grzegorczyk. 1953. Some classes of recursive functions. *Rozprawy Mat.* 4, 46. ISSN 0860-2581.

[Kur64] S. Kuroda. 1964. Classes of languages and linear-bounded automata. *Inf. Control.* 7, 2, 207–223. DOI: https://doi.org/10.1016/S0019-9958(64)90120-2.

[Rit63] R. W. Ritchie. 1963. Classes of predictably computable functions. *Trans. Am. Math. Soc.* 106, 139–173. DOI: https://doi.org/10.2307/1993719.

[SS63] J. C. Shepherdson and H. E. Sturgis. April. 1963. Computability of recursive functions. *J. ACM* 10, 2, 217–255. DOI: https://doi.org/10.2307/2271277.

[Smu61] R. M. Smullyan. 1961. *Theory of Formal Systems, Revised Edition*. Annals of Mathematical Studies. Vol. 47, Princeton.

18 Further Reading

There are a number of excellent sources for readers who would like to learn more about computational complexity theory. These include Arora and Barak's *Computational Complexity: A Modern Approach* [AB09], Du and Ko's *Theory of Computational Complexity* [DK14], Goldreich's *Computational Complexity: A Conceptual Perspective* [Gol08], Papadimitriou's *Computational Complexity* [Pap94b], and Wegener's *Complexity Theory* [Weg05]. The Complexity Zoo, found at complexityzoo.net, contains entries for the complexity classes referred to in this volume, with brief definitions and pointers to the literature. Information on bounded arithmetic and proof complexity can be found in the books by Krajíček, *Bounded Arithmetic, Propositional Logic, and Complexity Theory* [Kra95] and *Proof Complexity* [Kra19], and Cook's own *Logical Foundations of Proof Complexity* [CN10], co-authored with Nguyen.

More information about the P vs NP problem and the theory of NP-completeness can be found in several sources. Fortnow's book *The Golden Ticket* [For13] provides a good introduction to the area, as do the surveys "P=NP" [Aar16] by Aaronson and "P, NP and mathematics—A computational complexity perspective" [Wig06] and "Knowledge, creativity and P versus NP" [Wig09] by Wigderson. Sipser's "The history and status of the P versus NP question" [Sip92] presents the state of the problem as it stood in 1992. Finally, Cook's "The P versus NP problem" [Coo06a], written for the Clay Mathematics Institute's Millennium Problems site, gives his own overview of the area.

Previously published biographical information about Steve Cook may be found in the book *Out of Their Minds* [LS98] by Lazere and Shasha, as well as in an interview with Cook, done on behalf of the Charles Babbage Institute by Philip Frana, that appeared in *Communications of the ACM* in 2021 [Fra12].

Bibliography

Bibliography of the Works of Stephen A. Cook

1965

[Coo65] S. A. Cook. 1965. Algebraic techniques and the mechanization of number theory. Rand Memorandum RM-4319-Pr, Rand Corporation, Santa Monica, California. https://www.rand.org/pubs/research_memoranda/RM4319.html.

1966

[Coo66a] S. A. Cook. 1966a. *On the Minimum Computation Time of Functions*. Ph.D. Dissertation. Harvard University.

[Coo66b] S. A. Cook. 1966b. The solvability of the derivability problem for one-normal systems. *J. ACM* 13, 2, 223–225. DOI: https://doi.org/10.1145/321328.321333.

[CW66] S. A. Cook and H. Wang. 1966. Characterizations of ordinal numbers in set theory. *Math. Ann.* 164, 1–25. DOI: https://doi.org/10.1007/BF01351806.

1968

[Coo68] S. A. Cook. 1968. Off line Turing machine computation. In *Proceedings of the Hawaii International Conference on System Sciences*. University of Hawaii Press, 14–16.

[CS68] S. A. Cook and W. Savitch. 1968. *Mazes and Turing Machines.* Computer Center Technical Report 29. University of California, Berkeley.

1969

[Coo69] S. A. Cook. May 5–7. 1969. Variations on pushdown machines (detailed abstract). In *Proceedings of the 1st Annual ACM Symposium on Theory of Computing*. Marina del Rey, CA. ACM, 229–231. DOI: https://doi.org/10.1145/800169.805437.

[CA69] S. A. Cook and S. O. Aanderaa. 1969. On the minimum computation time of functions. *Trans. Am. Math. Soc.* 142, 291–314. ISSN 0002-9947. DOI: https://doi.org/10.2307/1995359.

1970

[Coo70] S. A. Cook. 1970. Path systems and language recognition. In *Proceedings of the 2nd Annual ACM Symposium on Theory of Computing*, May 4–6, 1970, Northampton, Massachusetts. ACM, 70–72. DOI: https://doi.org/10.1145/800161.805151.

1971

[Coo71a] S. A. Cook. 1971a. Characterizations of pushdown machines in terms of time-bounded computers. *J. ACM* 18, 1, 4–18. DOI: https://doi.org/10.1145/321623.321625.

[Coo71b] S. A. Cook. May 3–5. 1971b. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*. Shaker Heights, Ohio. ACM, 151–158. DOI: https://doi.org/10.1145/800157.805047.

1972

[Coo72a] S. A. Cook. 1972a. Linear time simulation of deterministic two-way pushdown automata. In *Information Processing 71 (Proc. IFIP Congress, Ljubljana, 1971).* Vol. 1: Foundations and Systems. North-Holland, Amsterdam, 75–80.

[Coo72b] S. A. Cook. 1972b. A hierarchy for nondeterministic time complexity. In *Proceedings of the 4th Annual ACM Symposium on Theory of Computing*, May 1–3, 1972, Denver, Colorado. ACM, 187–192. DOI: https://doi.org/10.1145/800152.804913.

[CR72] S. A. Cook and R. A. Reckhow. 1972. Time-bounded random access machines. In *Proceedings of the 4th Annual ACM Symposium on Theory of Computing*, May 1–3, 1972, Denver, Colorado. ACM, 73–80. DOI: http://doi.acm.org/10.1145/800152.804898.

1973

[Coo73a] S. A. Cook. 1973a. A hierarchy for nondeterministic time complexity. *J. Comput. Syst. Sci.* 7, 4, 343–353. DOI: https://doi.org/10.1016/S0022-0000(73)80028-5.

[Coo73b] S. A. Cook. April 30–May 2. 1973b. An observation on time–storage trade off. In *Proceedings of the 5th Annual ACM Symposium on Theory of*

Computing. Austin, Texas. ACM, 29–33. DOI: https://doi.org/10.1145/800125.804032.

[CR73] S. A. Cook and R. A. Reckhow. 1973. Time bounded random access machines. *J. Comput. Syst. Sci.* 7, 4, 354–375. DOI: https://doi.org/10.1016/S0022-0000(73)80029-7.

1974

[BC74] A. Borodin and S. A. Cook. 1974. On the number of additions to compute specific polynomials (preliminary version). In *Proceedings of the 6th Annual ACM Symposium on Theory of Computing*, April 30–May 2, 1974, Seattle, Washington. ACM, 342–347. DOI: https://doi.org/10.1145/800119.803913.

[Coo74] S. A. Cook. 1974. An observation on time–storage trade off. *J. Comput. Syst. Sci.* 9, 3, 308–316. DOI: https://doi.org/10.1016/S0022-0000(74)80046-2.

[CR74] S. A. Cook and R. A. Reckhow. April 30–May 2. 1974. On the lengths of proofs in the propositional calculus (preliminary version). In *Proceedings of the 6th Annual ACM Symposium on Theory of Computing*. Seattle, Washington. ACM, 135–148. DOI: https://doi.org/10.1145/800119.803893.

[CS74] S. A. Cook and R. Sethi. 1974. Storage requirements for deterministic polynomial time recognizable languages. In *Proceedings of the 6th Annual ACM Symposium on Theory of Computing*, April 30–May 2, 1974, Seattle, Washington. ACM, 33–39. DOI: https://doi.org/10.1145/800119.803882.

1975

[Coo75a] S. A. Cook. February. 1975a. *Axiomatic and Interpretive Semantics for an ALGOL Fragment*. Technical Report 79. Department of Computer Science, University of Toronto.

[Coo75b] S. A. Cook. 1975b. Feasibly constructive proofs and the propositional calculus (preliminary version). In *Proceedings of the 7th Annual ACM Symposium on Theory of Computing*, May 5–7, 1975, Albuquerque, New Mexico. ACM, 83–97. DOI: https://doi.org/10.1145/800116.803756.

[CO75] S. A. Cook and D. C. Oppen. 1975. An assertion language for data structures. In *Conference Record of the Second ACM Symposium on Principles of Programming Languages*, January 1975, Palo Alto, California. ACM, 160–166. DOI: https://doi.org/10.1145/512976.512993.

[OC75] D. C. Oppen and S. A. Cook. 1975. Proving assertions about programs that manipulate data structures. In *Proceedings of the 7th Annual ACM Symposium on Theory of Computing*, May 5–7, 1975, Albuquerque, New Mexico. ACM, 107–116. DOI: https://doi.org/10.1145/800116.803758.

1976

[BC76] A. Borodin and S. A. Cook. 1976. On the number of additions to compute specific polynomials. *SIAM J. Comput.* 5, 1, 146–157. DOI: https://doi.org/10.1137/0205013.

[Coo76a] S. A. Cook. 1976a. A short proof of the pigeon-hole principle using extended resolution. *SIGACT News* 8, 4, 28–32. DOI: https://doi.org/10.1145/1008335.1008338.

[Coo76b] S. A. Cook. May. 1976b. Feasibly constructive proofs and the propositional calculus. In *Proceedings of the Saventh Annual ACM Conference on the Theory of Computing*, 83–97.

[CS76] S. A. Cook and R. Sethi. 1976. Storage requirements for deterministic polynomial time recognizable languages. *J. Comput. Syst. Sci.* 13, 1, 25–37. DOI: https://doi.org/10.1016/S0022-0000(76)80048-7.

[CR+76] S. A. Cook, C. W. Rackoff, et al. 1976. Lecture notes for CSC2429F, "Topics in the theory of computation", a course presented by the Department of Computer Science, University of Toronto during the fall.

1978

[Coo78a] S. A. Cook. September. 1978a. *Deterministic CFL's Are Accepted Simultaneously in Polynomial Time and Log Squared Space*. Technical Report TR-124/78. Department of Computer Science, University of Toronto.

[Coo78b] S. A. Cook. 1978b. Soundness and completeness of an axiom system for program verification. *SIAM J. Comput.* 7, 1, 70–90. DOI: https://doi.org/10.1137/0207005.

1979

[Coo79] S. A. Cook. April 30–May 2. 1979. Deterministic CFL's are accepted simultaneously in polynomial time and log squared space. In *Proceedings of the 11th Annual ACM Symposium on Theory of Computing*. Atlanta, Georgia. ACM, 338–345. DOI: https://doi.org/10.1145/800135.804426.

[CR79] S. A. Cook and R. A. Reckhow. 1979. The relative efficiency of propositional proof systems. *J. Symb. Log.* 44, 1, 36–50. DOI: https://doi.org/10.2307/2273702.

1980

[BC80] A. Borodin and S. A. Cook. April 28–30. 1980. A time–space tradeoff for sorting on a general sequential model of computation. In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing*. Los Angeles, California. ACM, 294–301. DOI: https://doi.org/10.1145/800141.804677.

[Coo80] S. A. Cook. February. 1980. *Towards a complexity theory of synchronous parallel computation.* Presented at Internationales Symposium über Logik und Algorithmik zu Ehren von Professor Ernst Specker, Zurich, L'Enseignement Mathematique, to appear.

[CR80] S. A. Cook and C. Rackoff. 1980. Space lower bounds for maze threadability on restricted machines. *SIAM J. Comput.* 9, 3, 636–652. DOI: https://doi.org/10.1137/0209048.

[DC80] P. W. Dymond and S. A. Cook. October 13–15. 1980. Hardware complexity and parallel computation (preliminary version). In *21st Annual Symposium on Foundations of Computer Science*. Syracuse, New York. IEEE, 360–372. DOI: https://doi.org/10.1109/SFCS.1980.22.

1981

[Coo81a] S. A. Cook. 1981a. Towards a complexity theory of synchronous parallel computation. *Enseign. Math.* 2, 27, 1–2, 99–124. ISSN 0013-8584.

[Coo81b] S. A. Cook. 1981b. Corrigendum: Soundness and completeness of an axiom system for program verification. *SIAM J. Comput.* 10, 3, 612. DOI: https://doi.org/10.1137/0210045.

1982

[BC82] A. Borodin and S. A. Cook. 1982. A time–space tradeoff for sorting on a general sequential model of computation. *SIAM J. Comput.* 11, 2, 287–297. DOI: https://doi.org/10.1137/0211022.

[CD82] S. A. Cook and C. Dwork. 1982. Bounds on the time for parallel RAM's to compute simple functions. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing* May 5–7, 1982, San Francisco, California. ACM, 231–233. DOI: https://doi.org/10.1145/800070.802196.

1983

[Coo83a] S. A. Cook. 1983a. An overview of computational complexity. *Commun. ACM* 26, 6, 400–408. DOI: https://doi.org/10.1145/358141.358144.

[Coo83b] S. A. Cook. 1983b. The classification of problems which have fast parallel algorithms. In *Foundations of Computation Theory, Proceedings of the 1983 International FCT-Conference*, August 21–27, 1983, Borgholm, Sweden. Lecture Notes in Computer Science, Vol. 158. Springer, 78–93. DOI: https://doi.org/10.1007/3-540-12689-9_95.

[BCP83] A. Borodin, S. A. Cook, and N. Pippenger. 1983. Parallel computation for well-endowed rings and space-bounded probabilistic machines. *Inf. Control* 58, 1–3, 113–136. DOI: https://doi.org/10.1016/S0019-9958(83)80060-6.

[MC83] P. McKenzie and S. A. Cook. 1983. The parallel complexity of the abelian permutation group membership problem. In *24th Annual Symposium on Foundations of Computer Science*, November 7–9, 1983, Tucson, Arizona. IEEE, 154–161. DOI: https://doi.org/10.1109/SFCS.1983.74.

[vBCMV83] B. von Braunmühl, S. A. Cook, K. Mehlhorn, and R. Verbeek. 1983. The recognition of deterministic CFLs in small time and space. *Inf. Control* 56, 1–2, 34–51. DOI: https://doi.org/10.1016/S0019-9958(83)80049-7.

1984

[BCH84] P. Beame, S. A. Cook, and H. J. Hoover. 1984. Log depth circuits for division and related problems. In *25th Annual Symposium on Foundations of Computer Science*, October 24–26, 1984. West Palm Beach, Florida. IEEE, 1–6. DOI: https://doi.org/10.1109/SFCS.1984.715894.

[Coo84] S. A. Cook. 1984. Can computers routinely discover mathematical proofs? *Proc. Am. Philos. Soc*. 128, 1, 40–43. ISSN 0003049X.

1985

[Coo85] S. A. Cook. 1985. A taxonomy of problems with fast parallel algorithms. *Inf. Control* 64, 1–3, 2–21. DOI: https://doi.org/10.1016/S0019-9958(85)80041-3.

[CH85] S. A. Cook and H. J. Hoover. 1985. A depth-universal circuit. *SIAM J. Comput.* 14, 4, 833–839. DOI: https://doi.org/10.1137/0214058.

1986

[BCH86] P. Beame, S. A. Cook, and H. J. Hoover. 1986. Log depth circuits for division and related problems. *SIAM J. Comput.* 15(4), 4, 994–1003. DOI: https://doi.org/10.1137/0215070.

[CDR86] S. A. Cook, C. Dwork, and R. Reischuk. 1986. Upper and lower time bounds for parallel random access machines without simultaneous writes. *SIAM J. Comput.* 15, 1, 87–97. DOI: https://doi.org/10.1137/0215006.

1987

[CM87] S. A. Cook and P. McKenzie. 1987. Problems complete for deterministic logarithmic space. *J. Algorithms* 8, 3, 385–394. DOI: https://doi.org/10.1016/0196-6774(87)90018-6.

[MC87] P. McKenzie and S. A. Cook. 1987. The parallel complexity of abelian permutation group problems. *SIAM J. Comput.* 16, 5, 880–909. DOI: https://doi.org/10.1137/0216058.

1988

[BCD+88] A. Borodin, S. A. Cook, P. W. Dymond, W. L. Ruzzo, and M. Tompa. 1988. Two applications of complementation via inductive counting. In *Proceedings of Third Annual Structure in Complexity Theory Conference*, June 14–17, 1988, Georgetown University, Washington, DC. IEEE, 116–125. DOI: https://doi.org/10.1109/SCT.1988.5271.

[Coo88] S. A. Cook. 1988. Short propositional formulas represent nondeterministic computations. *Inf. Process. Lett.* 26, 5, 269–270. DOI: https://doi.org/10.1016/0020-0190(88)90152-4.

[CL88] S. A. Cook and M. Luby. 1988. A simple parallel algorithm for finding a satisfying truth assignment to a 2-CNF formula. *Inf. Process. Lett.* 27, 3, 141–145. DOI: https://doi.org/10.1016/0020-0190(88)90069-5.

1989

[BCD+89a] A. Borodin, S. A. Cook, P. W. Dymond, W. L. Ruzzo, and M. Tompa. 1989a. Two applications of inductive counting for complementation problems. *SIAM J. Comput.* 18, 3, 559–578. DOI: https://doi.org/10.1137/0218038.

[BCD+89b] A. Borodin, S. A. Cook, P. W. Dymond, W. L. Ruzzo, and M. Tompa. 1989b. Erratum: Two applications of inductive counting for complementation problems. *SIAM J. Comput.* 18, 6, 1283. DOI: https://doi.org/10.1137/0218084.

[CK89] S. A. Cook and B. M. Kapron. 1989. Characterizations of the basic feasible functionals of finite type (extended abstract). In *30th Annual Symposium on Foundations of Computer Science*, 30 October–1 November 1989, Research Triangle Park, North Carolina. IEEE, 154–159. DOI: https://doi.org/10.1109/SFCS.1989.63471.

[CU89] S. A. Cook and A. Urquhart. 1989. Functional interpretations of feasibly constructive arithmetic (extended abstract). In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, May 14–17, 1989, Seattle, Washington. ACM, 107–112. DOI: https://doi.org/10.1145/73007.73017.

[DC89] P. W. Dymond and S. A. Cook. 1989. Complexity theory of parallel time and hardware. *Inf. Comput.* 80, 3, 205–226. DOI: https://doi.org/10.1016/0890-5401(89)90009-6.

1990

[CK90] S. A. Cook and B. M. Kapron. 1990. Characterizations of the basic feasible functionals of finite type. In *Feasible Mathematics* (Ithaca, NY, 1989). Progress in Computer Science and Applied Logic, Vol. 9. Birkhäuser Boston, Boston, MA, 71–96. DOI: https://doi.org/10.1007/978-1-4612-3466-1_5.

[CP90] S. A. Cook and T. Pitassi. 1990. A feasibly constructive lower bound for resolution proofs. *Inf. Process. Lett.* 34, 2, 81–85. DOI: https://doi.org/10.1016/0020-0190(90)90141-J.

1991

[Coo91] S. A. Cook. 1991. Computational complexity of higher type functions. In *Proceedings of the International Congress of Mathematicians,* Vol. I and II (Kyoto, 1990), Mathematical Society of Japan, Tokyo. Springer, 55–69.

[KC91] B. M. Kapron and S. A. Cook. 1991. A new characterization of Mehlhorn's polynomial time functionals (extended abstract). In *32nd Annual Symposium on Foundations of Computer Science*, 1–4 October 1991, San Juan, Puerto Rico. IEEE, 342–347. DOI: https://doi.org/10.1109/SFCS.1991.185389.

1992

[BC92a] S. Bellantoni and S. A. Cook. 1992a. A new recursion-theoretic characterization of the polytime functions. *Comput. Complex*. 2, 97–110. DOI: https://doi.org/10.1007/BF01201998.

[BC92b] S. Bellantoni and S. A. Cook. 1992b. A new recursion-theoretic characterization of the polytime functions (extended abstract). In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, May 4–6, 1992, Victoria, British Columbia, Canada. ACM, 283–293. DOI: https://doi.org/10.1145/129712.129740.

[BCGR92] S. R. Buss, S. A. Cook, A. Gupta, and V. Ramachandran. 1992. An optimal parallel algorithm for formula evaluation. *SIAM J. Comput.* 21, 4, 755–780. DOI: https://doi.org/10.1137/0221046.

[Coo92] S. A. Cook. 1992. Computability and complexity of higher type functions. In *Logic from Computer Science* (Berkeley, CA, 1989). Mathematical Sciences Research Institute Publications, Vol. 21. Springer, New York, 51–72. DOI: https://doi.org/10.1007/978-1-4612-2822-6_3.

[PCP92] I. S. Pressman, S. A. Cook, and J. K. Pachl. 1992. The optimal placement of replicas in a network using a READ ANY–WRITE ALL policy. In *Proceedings of the 1992 Conference of the Centre for Advanced Studies on Collaborative Research*, November 9–12, 1992, Toronto, Ontario, Canada, Vol. 2. IBM Press, 189–201. http://doi.acm.org/10.1145/962274.

1993

[CD93] S. A. Cook and P. W. Dymond. 1993. Parallel pointer machines. *Comput. Complex.* 3, 19–30. DOI: https://doi.org/10.1007/BF01200405.

[CU93] S. A. Cook and A. Urquhart. 1993. Functional interpretations of feasibly constructive arithmetic. *Ann. Pure Appl. Log*. 63, 2, 103–200. DOI: https://doi.org/10.1016/0168-0072(93)90044-E.

1995

[BCE+95] P. Beame, S. A. Cook, J. Edmonds, R. Impagliazzo, and T. Pitassi. 1995. The relative complexity of NP search problems. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*, 29 May–1 June 1995, Las Vegas, Nevada. ACM, 303–314. DOI: https://doi.org/10.1145/225058.225147.

1996

[CM96] S. A. Cook and D. G. Mitchell. 1996. Finding hard instances of the satisfiability problem: A survey. In *Satisfiability Problem: Theory and Applications, Proceedings of a DIMACS Workshop*, March 11–13, 1996, Piscataway, New Jersey. American Mathematical Society, 1–18.

[KC96] B. M. Kapron and S. A. Cook. 1996. A new characterization of type-2 feasibility. *SIAM J. Comput.* 25, 1, 117–132. DOI: https://doi.org/10.1137/S0097539794263452.

1997

[CIY97] S. A. Cook, R. Impagliazzo, and T. Yamakami. 1997. A tight relationship between generic oracles and type-2 complexity theory. *Inf. Comput.* 137, 2, 159–170. DOI: https://doi.org/10.1006/inco.1997.2646.

1998

[BCE+98] P. Beame, S. A. Cook, J. Edmonds, R. Impagliazzo, and T. Pitassi. 1998. The relative complexity of NP search problems. *J. Comput. Syst. Sci.* 57, 1, 3–19. DOI: https://doi.org/10.1006/jcss.1998.1575.

1999

[CS99] S. Cook and M. Soltys. 1999. Boolean programs and quantified propositional proof systems. *Bull. Sect. Log. Univ. Łódź* 28, 3, 119–129. ISSN 0138-0680.

[HC99] A. Haken and S. A. Cook. 1999. An exponential lower bound for the size of monotone real circuits. *J. Comput. Syst. Sci.* 58, 2, 326–335. DOI: https://doi.org/10.1006/jcss.1998.1617.

2000

[KRC20] V. Kabanets, C. Rackoff, and S. A. Cook. 2000. Efficiently approximable real-valued functions. *Electron. Colloq. Comput. Complex. (ECCC)* 7, 34. http://eccc.hpi-web.de/eccc-reports/2000/TR00-034/index.html.

2001

[CK01a] S. A. Cook and A. Kolokolova. 2001a. A second-order system for polynomial-time reasoning based on Graedel's theorem. *Electron. Colloq. Comput. Complex. (ECCC)* 8, 24. http://eccc.hpi-web.de/eccc-reports/2001/TR01-024/index.html.

[CK01b] S. A. Cook and A. Kolokolova. 2001b. A second-order system for polytime reasoning using Graedel's theorem. In *Proceedings of 16th Annual IEEE Symposium on Logic in Computer Science*, June 16–19, 2001, Boston, Massachusetts. IEEE, 177–186. DOI: https://doi.org/10.1109/LICS.2001.932495.

2002

[Coo02] S. A. Cook. 2002. Complexity classes, propositional proof systems, and formal theories. In *Proceedings of 17th IEEE Symposium on Logic in Computer Science (LICS 2002)*, 22–25 July 2002, Copenhagen, Denmark. IEEE, 311. DOI: https://doi.org/10.1109/LICS.2002.10000.

[CL02] S. A. Cook and Y. Liu. 2002. A complete axiomatization for blocks world. In *International Symposium on Artificial Intelligence and Mathematics, AI&M 2002*, January 2–4, 2002, Fort Lauderdale, Florida.

[CPP02] S. A. Cook, J. K. Pachl, and I. S. Pressman. 2002. The optimal location of replicas in a network using a READ-ONE–WRITE-ALL policy. *Distrib. Comput.* 15, 1, 57–66. DOI: https://doi.org/10.1007/s446-002-8031-5.

[SC02] M. Soltys and S. A. Cook. 2002. The proof complexity of linear algebra. In *Proceedings of 17th IEEE Symposium on Logic in Computer Science (LICS 2002)*, 22–25 July 2002, Copenhagen, Denmark. IEEE, 335–344. DOI: https://doi.org/10.1109/LICS.2002.1029841.

2003

[Coo03] S. A. Cook. 2003. The importance of the P versus NP question. *J. ACM* 50, 1, 27–29. DOI: https://doi.org/10.1145/602382.602398.

[CK03] S. A. Cook and A. Kolokolova. 2003. A second-order system for polytime reasoning based on Grädel's theorem. *Ann. Pure Appl. Log.* 124, 1–3, 193–231. DOI: https://doi.org/10.1016/S0168-0072(03)00056-3.

[CL03] S. A. Cook and Y. Liu. 2003. A complete axiomatization for blocks world. *J. Log. Comput.* 13, 4, 581–594. DOI: https://doi.org/10.1093/logcom/13.4.581.

2004

[Coo04] S. A. Cook. 2004. Theories for complexity classes and their propositional translations. In *Complexity of Computations and Proofs. Quaderni di Matematica, Vol. 13*. Departimento di Matematica, Seconda Universita degli Studi di Napoli, Caserta, 175–227.

[CT04a] S. A. Cook and N. Thapen. 2004a. The strength of replacement in weak arithmetic. *CoRR*, cs.LO/0409015. DOI: https://doi.org/10.48550/arXiv.cs/0409015.

[CT04b] S. A. Cook and N. Thapen. 2004b. The strength of replacement in weak arithmetic. In *Proceedings of the 19th IEEE Symposium on Logic in Computer Science (LICS 2004)*, July 14–17, 2004, Turku, Finland. IEEE, 256–264. DOI: https://doi.org/10.1109/LICS.2004.1319620.

[CK04] S. A. Cook and A. Kolokolova. 2004. A second-order theory for NL. In *Proceedings of the 19th IEEE Symposium on Logic in Computer Science (LICS 2004)*, July 14–17, 2004, Turku, Finland. IEEE, 398–407. DOI: https://doi.org/10.1109/LICS.2004.1319634.

[NC04] P. Nguyen and S. A. Cook. 2004. VTC^0: A second-order theory for TC^0. In *Proceedings of the 19th IEEE Symposium on Logic in Computer Science (LICS 2004)*, July 14–17, 2004, Turku, Finland. IEEE, 378–387. DOI: https://doi.org/10.1109/LICS.2004.1319632.

[SC04] M. Soltys and S. A. Cook. 2004. The proof complexity of linear algebra. *Ann. Pure Appl. Log.* 130, 1–3, 277–323. DOI: https://doi.org/10.1016/j.apal.2003.10.018.

2005

[BC05] M. Braverman and S. A. Cook. 2005. Computing over the reals: Foundations for scientific computing. *CoRR*, abs/cs/0509042. DOI: https://doi.org/10.48550/arXiv.cs/0509042.

[CM05] S. A. Cook and T. Morioka. 2005. Quantified propositional calculus and a second-order theory for NC^1. *Arch. Math. Log.* 44, 6, 711–749. DOI: https://doi.org/10.1007/s00153-005-0282-2.

[NC05] P. Nguyen and S. A. Cook. 2005. Theories for TC^0 and other small complexity classes. *CoRR*, abs/cs/0505013. DOI: https://doi.org/10.48550/arXiv.cs/0505013.

2006

[BC06] M. Braverman and S. A. Cook. 2006. Computing over the reals: Foundations for scientific computing. *Not. Am. Math. Soc.* 53, 3, 318–329. ISSN 0002-9920.

[Coo06a] S. Cook. 2006a. The P versus NP problem. In *The Millennium Prize Problems*. Clay Mathematics Institute, Cambridge, MA, 87–104.

[Coo06b] S. A. Cook. 2006b. Comments on Beckmann's uniform reducts. *CoRR*, abs/cs/0601086. DOI: https://doi.org/10.48550/arXiv.cs/0601086.

[CT06] S. A. Cook and N. Thapen. 2006. The strength of replacement in weak arithmetic. *ACM Trans. Comput. Log.* 7, 4, 749–764. DOI: https://doi.org/10.1145/1183278.1183283.

[NC06] P. Nguyen and S. A. Cook. 2006. Theories for TC^0 and other small complexity classes. *Log. Methods Comput. Sci*. 2, 1. DOI: https://doi.org/10.2168/LMCS-2(1:3)2006.

2007

[ACN07] K. Aehlig, S. A. Cook, and P. Nguyen. September 11–15. 2007. Relativizing small complexity classes and their theories. In *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Proceedings*. Lausanne, Switzerland. Lecture Notes in Computer Science, Vol. 4646. Springer, 374–388. DOI: https://doi.org/10.1007/978-3-540-74915-8_29.

[CK07] S. A. Cook and J. Krajíček. 2007. Consequences of the provability of $NP \subseteq P/poly$. *J. Symb. Log.* 72, 4, 1353–1371. DOI: https://doi.org/10.2178/jsl/1203350791.

[NC07] P. Nguyen and S. A. Cook. July 10–12. 2007. The complexity of proving the discrete Jordan curve theorem. In *22nd IEEE Symposium on Logic in Computer Science (LICS 2007), Proceedings.* Wroclaw, Poland. IEEE, 245–256. DOI: https://doi.org/10.1109/LICS.2007.48.

2009

[BCM+09a] M. Braverman, S. A. Cook, P. McKenzie, R. Santhanam, and D. Wehr. December 15–17. 2009a. Fractional pebbling and thrifty branching programs. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2009*. IIT Kanpur, India. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 109–120. DOI: https://doi.org/10.4230/LIPIcs.FSTTCS.2009.2311.

[BCM+09b] M. Braverman, S. A. Cook, P. McKenzie, R. Santhanam, and D. Wehr. August 24–28. 2009b. Branching programs for tree evaluation. In *Mathematical Foundations of Computer Science 2009, 34th International Symposium, MFCS 2009, Proceedings*. Novy Smokovec, High Tatras, Slovakia. Lecture Notes in Computer Science, Vol. 5734. Springer, 175–186. DOI: https://doi.org/10.1007/978-3-642-03816-7_16.

2010

[CF10] S. A. Cook and L. Fontes. August 23–27. 2010. Formal theories for linear algebra. In *Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Proceedings*. Brno, Czech Republic. Lecture Notes in Computer Science, Vol. 6247. Springer, 245–259. DOI: https://doi.org/10.1007/978-3-642-15205-4_21.

[CN10] S. Cook and P. Nguyen. 2010. *Logical Foundations of Proof Complexity*. Perspectives in Logic. Cambridge University Press, Cambridge; Association for Symbolic Logic, La Jolla, CA. ISBN 978-0-521-51729-4. DOI: https://doi.org/10.1017/CBO9780511676277.

[CMW+10] S. A. Cook, P. McKenzie, D. Wehr, M. Braverman, and R. Santhanam. 2010. Pebbles and branching programs for tree evaluation. *CoRR*, abs/1005.2642. DOI: https://doi.org/10.48550/arXiv.1005.2642.

[KC10] A. Kawamura and S. A. Cook. June 5–8. 2010. Complexity theory for operators in analysis. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010*. Cambridge, Massachusetts. ACM, 495–502. DOI: https://doi.org/10.1145/1806689.1806758.

[NC10] P. Nguyen and S. A. Cook. 2010. The complexity of proving the discrete Jordan curve theorem. *CoRR*, abs/1002.2954. DOI: https://doi.org/10.48550/arXiv.1002.2954.

2011

[LC11a] D. T. M. Le and S. A. Cook. 2011a. Formalizing randomized matching algorithms. *Log. Methods Comput. Sci.* 8, 3. DOI: https://doi.org/10.2168/LMCS-8(3:5)2012.

[LC11b] D. T. M. Le and S. A. Cook. June 21–24. 2011b. Formalizing randomized matching algorithms. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011*. Toronto, Ontario, Canada. IEEE, 185–194. DOI: https://doi.org/10.1109/LICS.2011.12.

[LCY11a] D. T. M. Le, S. A. Cook, and Y. Ye. 2011a. Complexity classes and theories for the comparator circuit value problem. *CoRR*, abs/1106.4142. http://arxiv.org/abs/1106.4142.

[LCY11b] D. T. M. Le, S. A. Cook, and Y. Ye. September 12–15, 2011b. A formal theory for the complexity class associated with the stable marriage problem. In *Computer Science Logic, 25th International Workshop/20th Annual Conference of the EACSL, CSL 2011, Proceedings*. Bergen, Norway. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 381–395. DOI: https://doi.org/10.4230/LIPIcs.CSL.2011.381.

2012

[ACC+12] A. Ada, A. Chattopadhyay, S. A. Cook, L. Fontes, M. Koucký, and T. Pitassi. June 26–29. 2012. The hardness of being private. In *Proceedings of the 27th*

Conference on Computational Complexity, CCC 2012. Porto, Portugal. IEEE, 192–202. DOI: https://doi.org/10.1109/CCC.2012.24.

[ACN12] K. Aehlig, S. A. Cook, and P. Nguyen. 2012. Relativizing small complexity classes and their theories. *CoRR*, abs/1204.5508. http://arxiv.org/abs/1204.5508.

[Coo12a] S. A. Cook. 2012a. Relativized propositional calculus. *CoRR*, abs/1203.2168. http://arxiv.org/abs/1203.2168

[Coo12b] S. A. Cook. September 3–6. 2012b. Connecting complexity classes, weak formal theories, and propositional proof systems (invited talk). In *Computer Science Logic (CSL'12)—26th International Workshop/21st Annual Conference of the EACSL, CSL 2012*. Fontainebleau, France. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 9–11. DOI: https://doi.org/10.4230/LIPIcs.CSL.2012.9.

[CF12] S. A. Cook and L. Fontes. 2012. Formal theories for linear algebra. *Log. Methods Comput. Sci.* 8, 1. DOI: https://doi.org/10.2168/LMCS-8(1:25)2012.

[CFL12] S. A. Cook, Y. Filmus, and D. T. M. Le. 2012. The complexity of the comparator circuit value problem. *CoRR*, abs/1208.2721. http://arxiv.org/abs/1208.2721.

[CMW+12] S. A. Cook, P. McKenzie, D. Wehr, M. Braverman, and R. Santhanam. 2012. Pebbles and branching programs for tree evaluation. *ACM Trans. Comput. Theory* 3, 2, Article 4, 1–43. DOI: https://doi.org/10.1145/2077336.2077337.

[KC12] A. Kawamura and S. A. Cook. 2012. Complexity theory for operators in analysis. *ACM Trans. Comput. Theory* 4, 2, Article 5, 1–24. DOI: https://doi.org/10.1145/2189778.2189780.

[NC12] P. Nguyen and S. A. Cook. 2012. The complexity of proving the discrete Jordan curve theorem. *ACM Trans. Comput. Log.* 13, 1, Article 9, 1–24. DOI: https://doi.org/10.1145/2071368.2071377.

2013

[FPRC13a] Y. Filmus, T. Pitassi, R. Robere, and S. A. Cook. 2013a. Average case lower bounds for monotone switching networks. *Electron. Colloq. Comput. Complex. (ECCC)* 20, 54. http://eccc.hpi-web.de/report/2013/054.

[FPRC13b] Y. Filmus, T. Pitassi, R. Robere, and S. A. Cook. October 26–29. 2013b. Average case lower bounds for monotone switching networks. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013*. Berkeley, CA. IEEE, 598–607. DOI: https://doi.org/10.1109/FOCS.2013.70.

[GC13] K. Ghasemloo and S. A. Cook. September 2–5. 2013. Theories for subexponential-size bounded-depth Frege proofs. In *Computer Science Logic 2013 (CSL 2013), CSL 2013*. Torino, Italy. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 296–315. DOI: https://doi.org/10.4230/LIPIcs.CSL.2013.296.

[KC13] A. Kawamura and S. A. Cook. 2013. Complexity theory for operators in analysis. *CoRR*, abs/1305.0453. DOI: https://doi.org/10.48550/arXiv.1305.0453.

2014

[ACC+14] A. Ada, A. Chattopadhyay, S. A. Cook, L. Fontes, M. Koucký, and T. Pitassi. 2014. The hardness of being private. *ACM Trans. Comput. Theory* 6, 1, Article 1, 1–24. DOI: https://doi.org/10.1145/2567671.

[CFL14] S. A. Cook, Y. Filmus, and D. T. M. Le. 2014. The complexity of the comparator circuit value problem. *ACM Trans. Comput. Theory* 6, 4, Article 15, 1–44. DOI: https://doi.org/10.1145/2635822.

2016

[ACN16] K. Aehlig, S. A. Cook, and P. Nguyen. 2016. Relativizing small complexity classes and their theories. *Comput. Complex*. 25, 1, 177–215. DOI: https://doi.org/10.1007/s00037-015-0113-8.

[CEMP16a] S. A. Cook, J. Edmonds, V. Medabalimi, and T. Pitassi. July 11–15. 2016a. Lower bounds for nondeterministic semantic read-once branching programs. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016*. Rome, Italy. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 36:1–36:13. DOI: https://doi.org/10.4230/LIPIcs.ICALP.2016.36.

[CPRR16b] S. A. Cook, T. Pitassi, R. Robere, and B. Rossman. 2016b. Exponential lower bounds for monotone span programs. *Electron. Colloq. Comput. Complex. (ECCC)* 23, 64. http://eccc.hpi-web.de/report/2016/064.

[RPRC16] R. Robere, T. Pitassi, B. Rossman, and S. A. Cook. October 9–11. 2016. Exponential lower bounds for monotone span programs. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016*. Hyatt Regency, New Brunswick, New Jersey. IEEE, 406–415. DOI: https://doi.org/10.1109/FOCS.2016.51.

2017

[CK17] S. A. Cook and B. M. Kapron. 2017. A survey of classes of primitive recursive functions. *Electron. Colloq. Comput. Complex. (ECCC)* 24, 1. https://eccc.weizmann.ac.il/report/2017/001.

[TC17] I. Tzameret and S. A. Cook. June 20–23. 2017. Uniform, integral and efficient proofs for the determinant identities. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017*. Reykjavik, Iceland, 1–12. DOI: https://doi.org/10.1109/LICS.2017.8005099.

References

[Aar16] S. Aaronson. 2016. P$\stackrel{?}{=}$NP. In J. F. Nash and M. T. Rassias (Eds.), *Open Problems in Mathematics*. Springer, 1–122. DOI: https://doi.org/10.1007/978-3-319-32162-2_1.

[ACM16] ACM. February 25. 2016. Stephen Cook, 1982 ACM Turing Award recipient. Video transcript. Interviewed by Bruce M. Kapron. https://amturing.acm.org/pdf/CookTuringTranscript.pdf.

[Adl78] L. Adleman. 1978. Two theorems on random polynomial time. In *Proceedings of 19th IEEE Symposium on Foundations of Computer Science*. IEEE Computer Society, Los Angeles, 75–83. DOI: https://doi.org/10.1109/SFCS.1978.37.

[APR83] L. Adleman, C. Pomerance, and R. S. Rumley. January. 1983. On distinguishing prime numbers from composite numbers. *Ann. Math.* 117, 173–206. DOI: https://doi.org/10.2307/2006975.

[AHU68] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. September. 1968. Time and tape complexity of pushdown automaton languages. *Inform. Contr.* 13, 3, 186–206. DOI: https://doi.org/10.1016/S0019-9958(68)91087-5.

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

[Ajt83] M. Ajtai. 1983. $\sum_1^1$-formulae on finite structures. *Ann. Pure Appl. Log.* 24, 1, 1–48. DOI: https://doi.org/10.1016/0168-0072(83)90038-6.

[Ajt88] M. Ajtai. October 24–26. 1988. The complexity of the pigeonhole principle. In *29th Annual Symposium on Foundations of Computer Science*. IEEE, White Plains, New York, 346–355. DOI: https://doi.org/10.1109/SFCS.1988.21951.

[Ale82] R. Aleliunas. August 18–20. 1982. Randomized parallel communication (preliminary version). In R. L. Probert, M. J. Fischer, and N. Santoro (Eds.), *ACM SIGACT–SIGOPS Symposium on Principles of Distributed Computing*. ACM, Ottawa, Canada, 60–72. DOI: https://doi.org/10.1145/800220.806683.

[AKL$^+$79] R. Aleliunas, R. M. Karp, R. J. Lipton, L. Lovász, and C. Rackoff. October. 1979. Random walks, universal traversal sequences, and complexity of maze problems. In *Proceedings of the 20th Annual Symposium of Foundations of Computer Science*, 218–223. DOI: https://doi.org/10.1109/SFCS.1979.34.

[ABRW04] M. Alekhnovich, E. Ben-Sasson, A. A. Razborov, and A. Wigderson. 2004. Pseudorandom generators in propositional proof complexity. *SIAM J. Comput.* 34, 1, 67–88. DOI: https://doi.org/10.1137/S0097539701389944.

[Ara00] T. Arai. 2000. A bounded arithmetic AID for Frege systems. *Ann. Pure Appl. Log.* 103, 1–3, 155–199. DOI: https://doi.org/10.1016/S0168-0072(99)00041-X.

[ADKF70] V. L. Arlazarov, Y. A. Dinitz, M. Kronrod, and I. Faradzhev. 1970. On economical construction of the transitive closure of an oriented graph. *Dokl. Akademii Nauk SSSR* 194, 3, 487–488.

[AB09] S. Arora and B. Barak. 2009. *Computational Complexity—A Modern Approach*. Cambridge University Press. ISBN 978-0-521-42426-4. DOI: https://doi.org/10.1017/CBO9780511804090.

[adHei79] F. M. auf der Heide. 1979. *A Comparison between Two Variations of a Pebble Game on Graphs*. Master's thesis. Fakultät für Mathematik, Universität Bielefeld.

[Aut79] J. Autebert. 1979. Une note sur le cylindre des langages déterministes. *Theor. Comput. Sci.* 8, 3, 395–399. DOI: https://doi.org/10.1016/0304-3975(79)90020-3.

[BLS87] L. Babai, E. M. Luks, and Á. Seress. 1987. Permutation groups in NC. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*. ACM, New York, 409–420. DOI: https://doi.org/10.1145/28395.28439.

[BBK+68] G. H. Barnes, R. M. Brown, M. Kato, D. J. Kuck, D. L. Slotnick, and R. A. Stokes. 1968. The ILLIAC IV computer. *IEEE Trans. Comput.* 17, 8, 746–757. DOI: https://doi.org/10.1109/TC.1968.229158.

[Bar86] D. A. M. Barrington. May 28–30. 1986. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*. ACM, Berkeley, California, 1–5. DOI: https://doi.org/10.1145/12130.12131.

[Bar89] D. A. M. Barrington. 1989. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *J. Comput. Syst. Sci.* 38, 1, 150–164. DOI: https://doi.org/10.1016/0022-0000(89)90037-8.

[Bat68] K. E. Batcher. April 30–May 2. 1968. Sorting networks and their applications. In *American Federation of Information Processing Societies: AFIPS Conference Proceedings: 1968 Spring Joint Computer Conference*, Atlantic City, NJ, Vol. 32, AFIPS Conference Proceedings. Thomson Book Company, Washington, DC, 307–314. DOI: https://doi.org/10.1145/1468075.1468121.

[BS82] W. Baur and V. Strassen. 1982. The complexity of partial derivatives. Preprint.

[Bea89] P. Beame. May 14–17. 1989. A general sequential time–space tradeoff for finding unique elements. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*. ACM, Seattle, Washington, 197–203. DOI: https://doi.org/10.1145/73007.73026.

[Bea91] P. Beame. 1991. A general sequential time–space tradeoff for finding unique elements. *SIAM J. Comput.* 20, 2, 270–277. DOI: https://doi.org/10.1137/0220017.

[BM12] P. Beame and P. McKenzie. 2012. A note on Nečiporuk's method for nondeterministic branching programs. http://www.cs.washington.edu/homes/beame/papers/neci.pdf.

[BSSV03] P. Beame, M. Saks, X. Sun, and E. Vee. 2003. Time-space tradeoff lower bounds for randomized computation of decision problems. *J. ACM* 50, 154–195. DOI: https://doi.org/10.1145/636865.636867.

[BB14] A. Beckmann and S. R. Buss. 2014. Improved witnessing and local improvement principles for second-order bounded arithmetic. *ACM Trans. Comput. Log.* 15, 1, Article 2:1–35. DOI: https://doi.org/10.1145/2559950.

[Ben65] V. E. Benes. 1965. *Mathematical Theory of Connecting Networks and Telephone Traffic*. Academic Press, New York, NY. DOI: https://doi.org/10.1016/S0076-5392(08)62459-5.

[Ben62] J. H. Bennett. 1962. *On Spectra*. Ph.D. thesis. Princeton University.

[Ber70] E. R. Berlekamp. 1970. Factoring polynomials over large finite fields. *Math. Comp.* 24, 713–735. DOI: https://doi.org/10.2307/2004849.

[Blu67] M. Blum. April. 1967. A machine-independent theory of the complexity of recursive functions. *J. ACM* 14, 2, 322–336. DOI: https://doi.org/10.1145/321386.321395.

[BM82] M. Blum and S. Micali. 1982. How to generate cryptographically strong sequences of pseudo random bits. In *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science*. IEEE Computer Society, Los Angeles, 112–117. DOI: https://doi.org/10.1109/SFCS.1982.72.

[Boo76] R. V. Book. 1976. Translational lemmas, polynomial time, and $(\log n)^j$-space. *Theor. Comput. Sci.* 1, 3, 215–226. DOI: https://doi.org/10.1016/0304-3975(76)90057-8.

[BLS84] R. V. Book, T. J. Long, and A. L. Selman. 1984. Quantitative relativizations of complexity classes. *SIAM J. Comput.* 13, 3, 461–487. DOI: https://doi.org/10.1137/0213030.

[Bor77] A. Borodin. 1977. On relating time and space to size and depth. *SIAM J. Comput.* 6, 4, 733–744. DOI: https://doi.org/10.1137/0206054.

[Bor80] A. Borodin. February. 1980. *Structured vs General Models in Computational Complexity*. Presented at Internationales Symposium über Logik und Algorithmik zu Ehren von Professor Ernst Specker. Zurich, L'Enseignement Mathematique, to appear.

[Bor82] A. Borodin. 1982. Structured vs. general models in computational complexity. In *Logic and Algorithmic*, Monographie no. 30 de L'Enseignement Mathematique Université de Genève.

[BM75] A. Borodin and I. Munro. 1975. *The Computational Complexity of Algebraic and Numeric Problems*. Elsevier, New York.

[BFK+79] A. Borodin, M. J. Fischer, D. G. Kirkpatrick, N. A. Lynch, and M. Tompa. October 29–31. 1979. A time–space tradeoff for sorting on non-oblivious

machines. In *20th Annual Symposium on Foundations of Computer Science*. IEEE, San Juan, Puerto Rico, 319–327. DOI: https://doi.org/10.1109/SFCS.1979.4.

[BFK+81] A. Borodin, M. J. Fischer, D. G. Kirkpatrick, N. A. Lynch, and M. Tompa. 1981. A time–space tradeoff for sorting on non-oblivious machines. *J. Comput. Syst. Sci.* 22, 3, 351–364. DOI: https://doi.org/10.1016/0022-0000(81)90037-4.

[BvGH82] A. Borodin, J. von zur Gathen, and J. Hopcroft. 1982. Fast parallel matrix and GCD computations. In *23rd IEEE Symposium on Foundations of Computer Science.* IEEE Computer Society, Los Angeles, 65–71. DOI: https://doi.org/10.1109/SFCS.1982.17.

[BFM+86] A. Borodin, F. E. Fich, F. Meyer auf der Heide, E. Upfal, and A. Wigderson. January 16–18. 1986. A time–space tradeoff for element distinctness. In *Proceedings of STACS 86, 3rd Annual Symposium on Theoretical Aspects of Computer Science*. Lecture Notes in Computer Science, Vol. 210. Springer, Orsay, France, 353–358. DOI: https://doi.org/10.1007/3-540-16078-7_89.

[BFM+87] A. Borodin, F. E. Fich, F. Meyer auf der Heide, E. Upfal, and A. Wigderson. 1987. A time–space tradeoff for element distinctness. *SIAM J. Comput.* 16, 1, 97–99. DOI: https://doi.org/10.1137/0216007.

[BRS93] A. Borodin, A. Razborov, and R. Smolensky. 1993. On lower bounds for read-k-times branching programs. *Comput. Complex.* 3, 1–18. DOI: https://doi.org/10.1007/BF01200404.

[Bra79] G. Brassard. 1979. A note on the complexity of cryptography (corresp.). *IEEE Trans. Inf. Theory* 25, 2, 232–233. DOI: https://doi.org/10.1109/TIT.1979.1056010.

[Bre74] R. P. Brent. 1974. The parallel evaluation of general arithmetic expressions. *J. ACM* 21, 2, 201–206. DOI: https://doi.org/10.1145/321812.321815.

[BD78] R. W. Brockett and D. Dobkin. 1978. On the optimal evaluation of a set of bilinear forms. *Linear Algebra Appl.* 19, 207–235. DOI: https://doi.org/10.1016/0024-3795(78)90012-5.

[Bus86a] S. R. Buss. 1986a. *Bounded Arithmetic*. Studies in Proof Theory (Lecture Notes). Revision of 1985 Princeton University Ph.D. thesis. Bibliopolis, Naples, Italy.

[Bus86b] S. R. Buss. June 2–5. 1986b. The polynomial hierarchy and intuitionistic bounded arithmetic. In A. L. Selman (Ed.), *Structure in Complexity Theory, Proceedings of the Conference Held at the University of California*. Lecture Notes in Computer Science, Vol. 223. Springer, Berkeley, California, 77–103. DOI: https://doi.org/10.1007/3-540-16486-3_91.

[Bus87] S. R. Buss. 1987. Polynomial size proofs of the propositional pigeonhole principle. *J. Symb. Log.* 52, 4, 916–927. DOI: https://doi.org/10.2307/2273826.

[Bus91] S. R. Buss. 1991. Propositional consistency proofs. *Ann. Pure Appl. Log.* 52, 1–2, 3–29. DOI: https://doi.org/10.1016/0168-0072(91)90036-L.

[Bus15] S. Buss. 2015. Quasipolynomial size proofs of the propositional pigeonhole principle. *Theor. Comput. Sci.* 576, 77–84. DOI: https://doi.org/10.1016/j.tcs.2015.02.005.

[BK94] S. R. Buss and J. Krajíček. July. 1994. An application of Boolean complexity to separation problems in bounded arithmetic. *Proc. Lond. Math. Soc.* s3–69, 1, 1–21. DOI: https://doi.org/10.1112/plms/s3-69.1.1.

[CD86] B. E. Carpenter and R. W. Doran. (Eds). 1986. *A.M. Turing's ACE Report of 1946 and Other Papers.* MIT Press, Cambridge, Massachusetts. DOI: https://dl.acm.org/doi/10.5555/5946.

[CW67] A. B. Carroll and R. T. Wetherald. 1967. Application of parallel processing to numerical weather prediction. *J. ACM* 14, 3, 591–614. DOI: https://doi.org/10.1145/321406.321419.

[Cha69] G. J. Chaitin. January. 1969. On the length of programs for computing finite binary sequences. *J. ACM* 13, 4, 547–569 (October 1966); *J. ACM* 16, 1, 145–159. DOI: https://doi.org/10.1145/321356.321363.

[Cha75] G. J. Chaitin. July. 1975. A theory of program size formally identical to informational theory. *J. ACM* 22, 3, 329–340. DOI: https://doi.org/10.1145/321892.321894.

[CS76] A. K. Chandra and L. J. Stockmeyer. October 25–27. 1976. Alternation. In *17th Annual Symposium on Foundations of Computer Science*. IEEE, Houston, Texas, 98–108. DOI: https://doi.org/10.1109/SFCS.1976.4.

[CKS78] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. 1978. Alternation. IBM Research Report RC 7489.

[CKS81] A. K. Chandra, D. Kozen, and L. J. Stockmeyer. 1981. Alternation. *J. ACM* 28, 1, 114–133. DOI: https://doi.org/10.1145/322234.322243.

[CSV82] A. K. Chandra, L. J. Stockmeyer, and U. Vishkin. November 3–5. 1982. A complexity theory for unbounded fan-in parallelism. In *23rd Annual Symposium on Foundations of Computer Science*. IEEE, Chicago, Illinois, 1–13. DOI: https://doi.org/10.1109/SFCS.1982.3.

[CL73] C. L. Chang and C. T. Lee. 1973. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press.

[CBI02] Charles Babbage Institute. October 18. 2002. An interview with Stephen Cook, OH # 350. Interviewed by Philip Frana. https://cacm.acm.org/magazines/2012/1/144816-an-interview-with-stephen-a-cook/fulltext.

[CDL01] A. Chiu, G. I. Davida, and B. E. Litow. 2001. Division in logspace-uniform NC^1. *RAIRO Theor. Inform. Appl.* 35, 3, 259–275. DOI: https://doi.org/10.1051/ita:2001119.

[Cho56] N. Chomsky. 1956. Three models for the description of language. *IRE Trans. Inf. Theory* 2, 3, 113–124. DOI: https://doi.org/10.1109/TIT.1956.1056813.

[Cho62] N. Chomsky. 1962. *Context-Free Grammars and Push-Down Storage*. Research Laboratory of Electronics, Quarterly Progress Report 65. Massachusetts Institute of Technology.

[Chu56] A. Church. 1956. *Introduction to Mathematical Logic*, vol. I. Princeton University Press, Princeton.

[Clo53] C. Clos. March. 1953. A study of non-blocking switching networks. *Bell Syst. Tech. J.* 32, 2, 406–424. DOI: https://doi.org/10.1002/j.1538-7305.1953.tb01433.x.

[CR67] J. P. Cleave and H. E. Rose. 1967. E^n-arithmetic. In Crossley (Ed.), *Sets, Models, and Recursion Theory*. North-Holland, Amsterdam.

[CT92] P. Clote and G. Takeuti. 1992. Bounded arithmetic for NC, ALogTIME, L and NL. *Ann. Pure Appl. Log.* 56, 1–3, 73–117. DOI: https://doi.org/10.1016/0168-0072(92)90068-B.

[Cob65] A. Cobham. 1965. The intrinsic computational difficulty of functions. In *Proceedings of the 1964 International Congress for Logic, Methodology, and Philosophy of Science*. North-Holland, Amsterdam, 24–30.

[Cob66] A. Cobham. 1966. *The Recognition Problem for the Set of Perfect Squares*. Conference Record, IEEE 7th Annual Symposium on Switching and Automata Theory, 78–87.

[CS70] J. Cocke and J. T. Schwartz. 1970. *Programming Languages and their Compilers*. Technical Report. Courant Institute of Mathematical Sciences.

[CS58] J. Cocke and D. L. Slotnick. July. 1958. *The Use of Parallelism in Numerical Calculations*. Technical Report IBM Research Memorandum RC-55. IBM.

[CL82] H. Cohen and H. W. Lenstra Jr. 1982. Primarily testing and Jacobi sums. Report 82–18, University of Amsterdam, Dept. of Math.

[Col66] S. N. Cole. October. 1966. Real-time computation by *n*-dimensional iterative arrays of finite-state machines. In *IEEE Conf. Record of 1966 Seventh Annual Symposium on Switching and Automata Theory*, 53–77.

[Con63] M. Conway. 1963. A multiprocessor system design. In *Proceedings of the AFIPS Fall Joint Computer Conference*, Vol. 24. ACM, 139–146. DOI: https://doi.org/10.1145/1463822.1463838.

[CT65] J. M. Cooley and J. W. Tukey. 1965. An algorithm for the machine calculation of complex Fourier series. *Math. Comput.* 19, 297–301. DOI: https://doi.org/10.1090/S0025-5718-1965-0178586-1.

[CW82] D. Coppersmith and S. Winograd. 1982. On the asymptomatic complexity of matrix multiplication. *SIAM J. Comp*. 11, 472–492. DOI: https://doi.org/10.1109/SFCS.1981.27.

[CG70] D. G. Corneil and C. C. Gotlieb. January. 1970. An efficient algorithm for graph isomorphism. *J. Assoc. Comput. Machinery* 17, 1, 51–64. DOI: https://doi.org/10.1145/321556.321562.

[DP60] M. Davis and H. Putnam. 1960. A computing procedure for quantification theory. *J. Assoc. Comput. Machinery* 7, 201–215. DOI: https://doi.org/10.1145/321033.321034.

[DH76] W. Diffie and M. E. Hellman. 1976. New directions in cryptography. *IEEE Trans. Inform. Theory IT* 22, 6, 644–654. DOI: https://doi.org/10.1109/TIT.1976.1055638.

[DLR79] D. Dobkin, R. J. Lipton, and S. Reiss. 1979. Linear programming is log-space hard for P. *Inf. Process. Lett.* 8, 96–97. DOI: https://doi.org/10.1016/0020-0190(79)90152-2.

[Dow78] M. Dowd. May 1–3. 1978. Propositional representation of arithmetic proofs (preliminary version). In R. J. Lipton, W. A. Burkhard, W. J. Savitch, E. P. Friedman, and A. V. Aho (Eds.), *Proceedings of the 10th Annual ACM Symposium on Theory of Computing*. ACM, San Diego, California, 246–252. DOI: https://doi.org/10.1145/800133.804354.

[Dow79] M. Dowd. 1979. *Propositional Representation of Arithmetic Proofs*. Ph.D. Dissertation. Dept. of Computer Science, University of Toronto.

[DK14] D.-Z. Du and K.-I. Ko. 2014. *Theory of Computational Complexity* (2nd. ed.). Wiley. ISBN 9781118306086. DOI: https://doi.org/10.1002/9781118595091.

[Dym80] P. Dymond. 1980. *Simultaneous Resource Bounds and Parallel Computation.* Ph.D. Dissertation. University of Toronto, Dept. of Computer Science.

[Ear70] J. Earley. February. 1970. An efficient context-free parsing algorithm. *Comm. ACM* 13, 2, 94–102.

[ECTS59] J. P. Eckert, J. C. Chu, A. B. Tonik, and W. F. Schmitt. 1959. Design of Univac-LARC System: I. In *Papers Presented at the December 1–3, 1959, Eastern Joint IRE–AIEE–ACM Computer Conference IRE–AIEE–ACM '59 (Eastern)*. ACM, New York, NY, 59–65. ISBN 9781450378680. DOI: https://doi.org/10.1145/1460299.1460305.

[Edm65a] J. Edmonds. 1965a. Paths, trees, and flowers. *Can. J. Math.* 17, 449–467. DOI: https://doi.org/10.4153/CJM-1965-045-4.

[Edm65b] J. Edmonds. 1965b. Minimum partition of a matroid into independent subsets. *J. Res. Natl. Bur. Stand. Sect. B* 69, 67–72. DOI: https://doi.org/10.6028/JRES.069B.004.

[Edm67] J. Edmonds. 1967. Optimum branchings. *J. Res. Nat. Bur. Stand. B* 71, 4, 233–240.

[EIRS01] J. Edmonds, R. Impagliazzo, S. Rudich, and J. Sgall. 2001. Communication complexity towards lower bounds on circuit depth. *Comput. Complex.* 10, 3, 210–246.

[EM71] A. Ehrenfeucht and J. Mycielski. 1971. Abbreviating proofs by adding new axioms. *Bull. Am. Math. Soc.* 77, 366–367. DOI: https://doi.org/10.1090/S0002-9904-1971-12696-4.

[Ers58] A. P. Ershov. 1958. On programming of arithmetic operations. *Commun. ACM* 1, 8, 3–6. DOI: https://doi.org/10.1145/368892.368907.

[Eve63] R. J. Evey. 1963. The theory and application of pushdown store machines. In *Mathematical Linguistics and Automatic Translation*, Report No. NSF-10. Computation Laboratory, Harvard University, 217–255.

[Fef60] S. Feferman. 1960. Arithmetization of metamathematics in a general setting. *Fundam. Math*. 49, 1, 35–92.

[Fel68] W. Feller. 1968. *An Introduction to Probability Theory and its Applications*. I, John Wiley, New York.

[FR79] J. Ferrante and C. W. Rackoff. 1979. The Computational Complexity of Logical Theories. *Lecture Notes in Mathematics*. #718, Springer Verlag, New York.

[FM71] M. Fischer and A. Meyer. 1971. *Boolean Matrix Multiplication and Transitive Closure*. Conference Record, IEEE 12th Annual Symposium on Switching and Automata Theory, 129–131.

[FP74] M. J. Fischer and N. Pippenger. 1974. M. J. Fischer lectures on network complexity. Universität Frankfurt. Preprint. Journal version: N. Pippenger and M. J. Fischer. 1979. Relations among complexity measures. *J. ACM* 26, 2, 361–381. DOI: https://doi.org/10.1145/322123.322138.

[FR74] M. J. Fischer and M. O. Rabin. 1974. Super-exponential complexity of Presburger arithmetic. In R. Karp (Ed.), *Complexity of Computation. SIAM–AMS Proc.* 7, 27–42.

[Fly66] M. J. Flynn. 1966. Very high-speed computing systems. *Proc. IEEE* 54, 12, 1901–1909. DOI: https://doi.org/10.1109/PROC.1966.5273.

[For13] L. Fortnow. 2013. *The Golden Ticket - P, NP, and the Search for the Impossible*. Princeton University Press. ISBN 9780691156491. http://press.princeton.edu/titles/9937.html.

[FW78] S. Fortune and J. Wyllie. May 1–3. 1978. Parallelism in random access machines. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing*. ACM, San Diego, California, 114–118. DOI: https://doi.org/10.1145/800133.804339.

[Fra12] P. L. Frana. January. 2012. An interview with Stephen A. Cook. *Commun. ACM* 55, 1, 41–46. DOI: https://doi.org/10.1145/2063176.2063193.

[Fre67] G. Frege. 1967. *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. Halle, 1879. English translation in *From Frege to Godel, a Source Book in Mathematical Logic* (J. van Heijenoort, Editor), Harvard University Press, Cambridge, 1–82.

[Fre79] G. Frege. 1879. *Begriffsschrift: eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. Halle a/S., Louis Nebert.

[Fre80] G. N. Frederickson. January. 1980. *Upper bounds for time-space trade-offs in sorting and selection*, Tech. Rep. CS-80-3, Dept. Computer Science, Pennsylvania State University.

[FSS81] M. L. Furst, J. B. Saxe, and M. Sipser. October 28–30. 1981. Parity, circuits, and the polynomial-time hierarchy. In *22nd Annual Symposium on Foundations of Computer Science*. IEEE, Nashville, Tennessee, 260–270. DOI: https://doi.org/10.1109/SFCS.1981.35.

[GKM08] A. Gál, M. Koucký, and P. McKenzie. 2008. Incremental branching programs. *Theory Comput. Syst.* 43, 2, 159–184.

[GJ79] M. R. Garey and D. S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman. ISBN 0-7167-1044-7.

[Gen36] G. Gentzen. 1936. Die Widerspruchsfreiheit der reinen Zahlentheorie. *Math. Ann.* 112, 493–565. DOI: https://doi.org/10.1007/BF01565428.

[GLT79] J. R. Gilbert, T. Lengauer, and R. E. Tarjan. April 30–May 2. 1979. The pebbling problem is complete in polynomial space. In *Proceedings of the 11th Annual ACM Symposium on Theory of Computing*. ACM, Atlanta, Georgia, 237–248. DOI: https://doi.org/10.1145/800135.804418.

[GLT80] J. R. Gilbert, T. Lengauer, and R. E. Tarjan. 1980. The pebbling problem is complete in polynomial space. *SIAM J. Comput.* 9, 3, 513–524. DOI: https://doi.org/10.1137/0209038.

[Gil58] S. Gill. 1958. Parallel programming. *Comput. J.* 1, 1, 2–10. DOI: https://doi.org/10.1093/comjnl/1.1.2.

[Gil77] J. Gill. 1977. Computational complexity of probabilistic Turing machines. *SIAM J. Comput.* 6, 675–695. DOI: https://doi.org/10.1137/0206049.

[GG66] S. Ginsburg and S. A. Greibach. 1966. Deterministic context free languages. *Inf. Control* 9, 6, 620–648. DOI: https://doi.org/10.1016/S0019-9958(66)80019-0.

[GGH67] S. Ginsburg, S. A. Greibach, and M. A. Harrison. January. 1967. Stack automata and compiling. *J. ACM* 14, 1, 172–201. DOI: https://doi.org/10.1145/321371.321385.

[Göd36] K. Gödel. 1936. Über die länge von Beweisen. In K. Menger (Ed.), *Ergebnisse Eines Mathematischen Kolloquiums*, Vol. 7. B.G. Teubner, 23–24.

[GP18] P. W. Goldberg and C. H. Papadimitriou. 2018. Towards a unified complexity theory of total functions. *J. Comput. Syst. Sci.* 94, 167–192. DOI: https://doi.org/10.1016/j.jcss.2017.12.003.

[Gol08] O. Goldreich. 2008. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press. ISBN 9780521884730. DOI: https://doi.org/10.1017/CBO9780511804106.

[Gol77] L. M. Goldschlager. 1977. *Synchronous Parallel Computation*. Ph.D. Dissertation. University of Toronto. Also, Technical Report 114, Department of Computer Science, University of Toronto.

[Gol78] L. M. Goldschlager. May 1–3. 1978. A unified approach to models of synchronous parallel machines. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing*. ACM, San Diego, California, 89–94. DOI: https://doi.org/10.1145/800133.804336.

[GSS82] L. M. Goldschlager, R. A. Shaw, and J. Staples. 1982. The maximum flow problem is log space complete for P. *Theor. Comput. Sci.* 21, 1, 105–111. DOI: https://doi.org/10.1016/0304-3975(82)90092-5.

[Goo54] R. L. Goodstein. 1954. Logic-free formalisations of recursive arithmetic. *Math. Scand.* 2, 2, 247–261. http://www.jstor.org/stable/24489038.

[Goo57] R. L. Goodstein. 1957. *Recursive Number Theory*. North-Holland, Amsterdam.

[GHI67] J. Gray, M. A. Harrison, and O. H. Ibarra. July, August. 1967. Two-way pushdown automata. *Inform. Contr.* 11, 1, 2, 30–70. DOI: https://doi.org/10.1016/S0019-9958(67)90369-5.

[Gre73] S. A. Greibach. 1973. The hardest context-free language. *SIAM J. Comput.* 2, 4, 304–310. DOI: https://doi.org/10.1137/0202025.

[Gre74] S. A. Greibach. 1974. Jump PDA's and hierarchies of deterministic context-free languages. *SIAM J. Comput.* 3, 2, 111–127. DOI: https://doi.org/10.1137/0203009.

[Grz53] A. Grzegorczyk. 1953. Some classes of recursive functions. *Rozprawy Mat.* 4, 46. ISSN 0860-2581.

[HP93] P. Hájek and P. Pudlák. 1993. *Metamathematics of First-Order Arithmetic*. Perspectives in Mathematical Logic. Springer. ISBN 978-3-540-63648-9. http://www.springer.com/mathematics/book/978-3-540-63648-9.

[Hak85] A. Haken. 1985. The intractability of resolution. *Theor. Comput. Sci.* 39, 297–308. DOI: https://doi.org/10.1016/0304-3975(85)90144-6.

[HLMW86] J. Y. Halpern, M. C. Loui, A. R. Meyer, and D. Weise. 1986. On time versus space III. *Math. Syst. Theory* 19, 1, 13–28. DOI: https://doi.org/10.1007/BF01704903.

[HI68] M. A. Harrison and O. H. Ibarra. November. 1968. Multitape and multihead pushdown automata. *Inform. Contr.* 13, 5, 433–470. DOI: https://doi.org/10.1016/S0019-9958(68)90901-7.

[Har67] J. Hartmanis. 1967. On memory requirements for context-free language recognition. *J. ACM* 14, 4, 663–665. DOI: https://doi.org/10.1145/321420.321424.

[Har81] J. Hartmanis. January. 1981. Observations about the development of theoretical computer science. *Annals Hist. Comput.* 3, 1, 42–51. DOI: https://doi.org/10.1109/MAHC.1981.10005.

[Har93] J. Hartmanis. 1993. Gödel, von Neumann and the P =? NP problem. In G. Rozenberg and A. Salomaa (Eds.), *Current Trends in Theoretical Computer Science—Essays and Tutorials*, Vol. 40: World Scientific Series in Computer Science. World Scientific, 445–450. DOI: https://doi.org/10.1142/9789812794499_0033.

[HH71] J. Hartmanis and J. E. Hopcroft. 1971. An overview of the theory of computational complexity. *J. ACM* 18, 2, 444–475. DOI: https://doi.org/10.1145/321650.321661.

[HS74] J. Hartmanis and J. Simon. October 14–16. 1974. On the power of multiplication in random access machines. In *15th Annual Symposium on Switching and Automata Theory*. IEEE, New Orleans, Louisiana, 13–23. DOI: https://doi.org/10.1109/SWAT.1974.20.

[HS64] J. Hartmanis and R. E. Stearns. November 11–13. 1964. Computational complexity of recursive sequences. In *5th Annual Symposium on Switching Circuit Theory and Logical Design*. IEEE, Princeton, New Jersey, 82–90. DOI: https://doi.org/10.1109/SWCT.1964.6.

[HS65] J. Hartmanis and R. E. Stearns. 1965. On the computational complexity of algorithms. *Trans. Am. Math. Soc.* 117, 285–306. DOI: https://doi.org/10.2307/1994208.

[HIM78] J. Hartmanis, N. Immerman, and S. Mahaney. October. 1978. One-way log tape reductions. In *19th Annual Symposium on Foundations of Computer Science*, 65–71.

[HW93] J. Håstad and A. Wigderson. 1993. Composition of the universal relation. In *Proceedings of the Advances in Computational Complexity Theory, AMS-DIMACS 13*, 119–134.

[HLF18] Heidelberg Laureate Forum. May 8. 2018. 5th HLF—Laureate interview: Stephen A. Cook and William Morton Kahan. Video, 41 minutes. Interviewed by Tom Geller, September 24–29, 2017. Published May 8, 2018. https://www.youtube.com/watch?v=8v7uPDBekyo.

[HHN+93] L. A. Hemaspaandra, A. Hoene, A. V. Naik, M. Ogiwara, A. L. Selman, T. Thierauf, and J. Wang. July. 1993. *Selectivity: Reductions, Nondeterminism and Function Classes*. Technical Report TR 93-21. State University of New York at Buffalo, Buffalo, NY.

[Hen65] F. Hennie. 1965. *Crossing Sequences and Off-Line Turing Machine Computations*. Conference Record IEEE Symposium on Switching Circuit Theory and Logical Design, 179–190.

[HS66] F. C. Hennie and R. E. Stearns. October. 1966. Two-tape simulation of multitape Turing machines. *J. ACM* 13, 4, 533–546. DOI: https://doi.org/10.1145/321356.321362.

[HP07] P. Hertel and T. Pitassi. 2007. Black–white pebbling is PSPACE-complete. *Electron. Colloq. Comput. Complex. (ECCC)*, 14, 044. http://eccc.hpi-web.de/eccc-reports/2007/TR07-044/index.html.

[HP10] P. Hertel and T. Pitassi. 2010. The PSPACE-completeness of black–white pebbling. *SIAM J. Comput.* 39, 6, 2622–2682. DOI: https://doi.org/10.1137/080713513.

[HAB02] W. Hesse, E. Allender, and D. A. M. Barrington. 2002. Uniform constant-depth threshold circuits for division and iterated multiplication. *J. Comput. Syst. Sci.* 65, 4, 695–716. DOI: https://doi.org/10.1016/S0022-0000(02)00025-9.

[HAB14] W. Hesse, E. Allender, and D. A. M. Barrington. 2014. Corrigendum to "Uniform constant-depth threshold circuits for division and iterated multiplication" [J. Comput. Syst. Sci. 65, 4 (2002), 695–716]. *J. Comput. Syst. Sci.* 80, 2, 496–497. DOI: https://doi.org/10.1016/j.jcss.2013.09.002.

[HA50] D. Hilbert and W. Ackermann. 1950. *Principles of Mathematical Logic*. AMS Chelsea.

[HB34] D. Hilbert and P. I. Bernays. 1934. *Grundlagen der Mathematik. I*. Die Grundlehren der mathematischen Wissenschaften. Springer-Verlag.

[HB39] D. Hilbert and P. I. Bernays. 1939. *Grundlagen der Mathematik. I*. Die Grundlehren der mathematischen Wissenschaften. Springer-Verlag.

[Hoa69] C. A. R. Hoare. 1969. An axiomatic basis for computer programming. *Commun. ACM* 12, 10, 576–580. DOI: https://doi.org/10.1145/363235.363259.

[HS68] L. Hodes and E. Specker. 1968. Lengths of formulas and elimination of quantifiers I. In K. Schutte (Ed.), *Contributions to Mathematical Logic*, North Holland Publ. Co. 175–188. DOI: https://doi.org/10.1016/S0049-237X(08)70524-X.

[Hon80] J. W. Hong. April. 1980. On some space complexity problems about the set of assignments satisfying a boolean formula. In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing*. 310–317.

[Hoo79] H. J. Hoover. December. 1979. *Some Topics in Circuit Complexity*. M.Sc. thesis and Technical Report TR-138/79. Department of Computer Science, University of Toronto.

[HU67] J. E. Hopcroft and J. D. Ullman. August. 1967. Nonerasing stack automata. *J. Comput. Syst. Sci.* 1, 2, 166–186. DOI: https://doi.org/10.1016/S0022-0000(67)80013-8.

[HU69] J. E. Hopcroft and J. D. Ullman. 1969. *Formal Languages and their Relation to Automata*. Addison-Wesley, 262.

[HU79] J. E. Hopcroft and J. D. Ullman. 1979. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley. ISBN 0-201-02988-X.

[HPV75] J. E. Hopcroft, W. J. Paul, and L. G. Valiant. October 13–15. 1975. On time versus space and related problems. In *16th Annual Symposium on Foundations of Computer Science*. IEEE, Berkeley, California, 57–64. DOI: https://doi.org/10.1109/SFCS.1975.23.

[HPV77] J. E. Hopcroft, W. J. Paul, and L. G. Valiant. 1977. On time versus space. *J. ACM* 24, 2, 332–337. DOI: https://doi.org/10.1145/322003.322015.

[Imm88] N. Immerman. 1988. Nondeterministic space is closed under complementation. *SIAM J. Comput.* 17, 5, 935–938. DOI: https://doi.org/10.1137/0217058.

[JáJ92] J. F. JáJá. 1992. *An Introduction to Parallel Algorithms*. Addison-Wesley. ISBN 0-201-54856-9.

[Jeř04] E. Jeřábek. 2004. Dual weak pigeonhole principle, Boolean complexity, and derandomization. *Ann. Pure Appl. Log.* 129, 1–3, 1–37. DOI: https://doi.org/10.1016/j.apal.2003.12.003.

[Jeř07] E. Jeřábek. 2007. Approximate counting in bounded arithmetic. *J. Symb. Log.* 72, 3, 959–993. DOI: https://doi.org/10.2178/jsl/1191333850.

[Jeř09] E. Jeřábek. 2009. Approximate counting by hashing in bounded arithmetic. *J. Symb. Log.* 74, 3, 829–860. DOI: https://doi.org/10.2178/jsl/1245158087.

[JPY88] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. 1988. How easy is local search? *J. Comput. Syst. Sci.* 37, 1, 79–100. DOI: https://doi.org/10.1016/0022-0000(88)90046-3.

[Jon75] N. D. Jones. 1975. Space-bounded reducibility among combinatorial problems. *J. Comput. Syst. Sci.* 11, 68–85. DOI: https://doi.org/10.1016/S0022-0000(75)80050-X.

[JL76] N. D. Jones and W. T. Laaser. 1976. Complete problems for deterministic polynomial time. *Theor. Comput. Sci.* 3, 1, 105–117. DOI: https://doi.org/10.1016/0304-3975(76)90068-2.

[JL77] N. D. Jones and W. T. Laaser. 1977. Complete problems for deterministic polynomial time. *Theor. Comput. Sci.* 3, 105–117.

[JLL76] N. D. Jones, Y. E. Lien, and W. T. Laaser. 1976. New problems complete for nondeterministic log space. *Math. Syst. Theory* 10, 1–17. DOI: https://doi.org/10.1007/BF01683259.

[Kal82a] E. Kaltofen. May 5–7. 1982a. A polynomial reduction from multivariate to bivariate integer polynomial factorization. In *Proceedings of the 14th ACM Symposium in Theory Computing*. San Francisco, CA, 261–266.

[Kal82b] E. Kaltofen. 1982b. A polynomial-time reduction from bivariate to univariate integral polynomial factorization. In *Proceedings of the 23rd*

IEEE Symposium on Foundations of Computer Science. IEEE Computer Society, Los Angeles, 57–64. DOI: https://doi.org/10.1109/SFCS.1982.56.

[KS88] B. Kalyanasundaram and G. Schnitger. May 2–4. 1988. On the power of white pebbles (extended abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*. ACM, Chicago, Illinois, 258–266. DOI: https://doi.org/10.1145/62212.62237.

[KS91] B. Kalyanasundaram and G. Schnitger. 1991. On the power of white pebbles. *Combinatorica* 11, 2, 157–171. DOI: https://doi.org/10.1007/BF01206359.

[KO62] A. Karatsuba and Y. Ofman. 1962. Multiplication of multidigit numbers on automata. *Doklady Akad. Nauk* 145, 2, 293–294. Translated in *Soviet Phys. Doklady.* 7, 7 (1963), 595–596.

[KRW95] M. Karchmer, R. Raz, and A. Wigderson. 1995. Super-Logarithmic depth lower bounds via direct sum in communication complexity. *Comput. Complex.* 5, 191–204.

[Kar72] R. M. Karp. 1972. Reducibility among combinatorial problems. In *Proceedings of a Symposium on the Complexity of Computer Computations*, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York. Springer, 85–103. DOI: https://doi.org/10.1007/978-1-4684-2001-2_9.

[Kar19] R. Karp. 2019. A personal view of computer science at Berkeley. University of California, Berkeley. Retrieved June 17, 2019 from https://www2.eecs.berkeley.edu/bears/CS_Anniversary/karp-talk.html.

[KL80] R. M. Karp and R. J. Lipton. April. 1980. Some connections between nonuniform and uniform complexity classes. In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing.* 302–309. Also presented at the *Specker Symposium on Algorithms and Complexity*. Zurich, February 1980.

[KM69] R. M. Karp and R. E. Miller. 1969. Parallel program schemata. *J. Comput. Syst. Sci.* 3, 2, 147–195. DOI: https://doi.org/10.1016/S0022-0000(69)80011-5.

[KMW67] R. M. Karp, R. E. Miller, and S. Winograd. 1967. The organization of computations for uniform recurrence equations. *J. ACM* 14, 3, 563–590. DOI: https://doi.org/10.1145/321406.321418.

[Kas66] T. Kasami. March. 1966. *An Efficient Recognition and Syntax-Analysis Algorithm for Context-Free Language*. Technical Report R-257. Coordinated Science Laboratory, University of Illinois.

[Kha79] L. G. Khachian. 1979. A polynomial time algorithm for linear programming. *Doklady Akad. Nauk SSSR*. 244, 5, 1093–1096. Translated in *Soviet Math. Doklady*. 20, 191–194.

[Kla83] M. M. Klawe. November 7–9. 1983. A tight bound for black and white pebbles on the pyramid. In *24th Annual Symposium on Foundations of*

Computer Science. IEEE, Tucson, Arizona, 410–419. DOI: https://doi.org/10.1109/SFCS.1983.3.

[Kla85] M. M. Klawe. 1985. A tight bound for black and white pebbles on the pyramid. *J. ACM* 32, 1, 218–228. DOI: https://doi.org/10.1145/2455.214115.

[Kle52] S. C. Kleene. 1952. *Introduction to Metamathematics*. D. Van Nostrand, New York, NY.

[Knu65] D. E. Knuth. 1965. On the translation of languages from left to right. *Inf. Control* 8, 6, 607–639. DOI: https://doi.org/10.1016/S0019-9958(65)90426-2.

[Knu72] D. E. Knuth. 1972. Mathematical analysis of algorithms. In C. V. Freeman (Ed.), *Proceedings of the IFIP Congress 71*, *Vol. 1*. North-Holland, Amsterdam, 19–27.

[Knu73] D. E. Knuth. 1973. *The Art of Computer Programming,* Vol. 3 Sorting and Searching. Addison-Wesley, Reading, MA.

[Knu74] D. E. Knuth. 1974. A terminological proposal. *SIGACT News* 6, 1, 12–18. DOI: https://doi.org/10.1145/1811129.1811130.

[KB67] D. E. Knuth and R. H. Bigelow. October. 1967. Programming languages for automata. *J. ACM* 14, 4, 615–635. DOI: https://doi.org/10.1145/321420.321421.

[Ko91] K. Ko. 1991. *Complexity Theory of Real Functions*. Birkhäuser/Springer. ISBN 978-1-4684-6802-1. DOI: https://doi.org/10.1007/978-1-4684-6802-1.

[KF82] K. Ko and H. Friedman. 1982. Computational complexity of real functions. *Theor. Comput. Sci.* 20, 3, 323–352. DOI: https://doi.org/10.1016/S0304-3975(82)80003-0.

[Kol65] A. N. Kolmogorov. 1965. Three approaches to the concept of the amount of information. *Probl. Pered. Inf. (Probl. of Inf. Transm.)* 1.

[KU58] A. N. Kolmogorov and V. A. Uspenski. 1958. On the definition of an algorithm, *Uspehi Mat. Nauk.* 13, 4(82), 3–28. ISSN 0042-1316. AMS Transl. 2nd ser. 29 (1963), 217–245.

[KNT11] L. A. Kolodziejczyk, P. Nguyen, and N. Thapen. 2011. The provably total NP search problems of weak second order bounded arithmetic. *Ann. Pure Appl. Log.* 162, 6, 419–446. DOI: https://doi.org/10.1016/j.apal.2010.12.002.

[Koz76] D. Kozen. October 25–27. 1976. On parallelism in Turing machines. In *17th Annual Symposium on Foundations of Computer Science*. IEEE, Houston, Texas, 89–97. DOI: https://doi.org/10.1109/SFCS.1976.20.

[Kra95] J. Krajíček. 1995. *Bounded Arithmetic, Propositional Logic, and Complexity Theory*, Vol. 60: Encyclopedia of Mathematics and Its Applications. Cambridge University Press. ISBN 978-0-521-45205-2. DOI: https://doi.org/10.1017/CBO9780511529948.

[Kra01] J. Krajíček. 2001. On the weak pigeonhole principle. *Fund. Math.* 170, 1, 123–140. DOI: https://doi.org/10.4064/fm170-1-8.

[Kra10] J. Krajíček. 2010. *Forcing with Random Variables and Proof Complexity*. London Mathematical Society Lecture Note Series. Cambridge University Press. DOI: https://doi.org/10.1017/CBO9781139107211.

[Kra19] J. Krajíček. 2019. *Proof Complexity*, Vol. 170: Encyclopedia of Mathematics and Its Applications. Cambridge University Press. DOI: https://doi.org/10.1017/9781108242066.

[KP89] J. Krajíček and P. Pudlák. 1989. Propositional proof systems, the consistency of first order theories and the complexity of computations. *J. Symb. Log.* 54, 3, 1063–1079. DOI: https://doi.org/10.2307/2274765.

[KP90] J. Krajíček and P. Pudlák. 1990. Quantified propositional calculi and fragments of bounded arithmetic. *Math. Log. Q.* 36, 1, 29–46. DOI: https://doi.org/10.1002/malq.19900360106.

[KT92] J. Krajíček and G. Takeuti. 1992. On induction-free provability. *Ann. Math. Artif. Intell.* 6, 1–3, 107–125. DOI: https://doi.org/10.1007/BF01531024.

[KPW95] J. Krajíček, P. Pudlák, and A. R. Woods. 1995. An exponential lower bound to the size of bounded depth Frege proofs of the pigeonhole principle. *Random Struct. Algorithms* 7, 1, 15–40. DOI: https://doi.org/10.1002/rsa.3240070103.

[KST07] J. Krajíček, A. Skelley, and N. Thapen. 2007. NP search problems in low fragments of bounded arithmetic. *J. Symb. Log.* 72, 2, 649–672. DOI: https://doi.org/10.2178/jsl/1185803628.

[KR64] D. L. Kreider and R. W. Ritchie. 1964. Predictably computable functionals and definitions by recursion. *Zeitschrift für math. Logik und Grundlagen der Math*. 10, 65–80.

[Kur64] S. Kuroda. 1964. Classes of languages and linear-bounded automata. *Inf. Control.* 7, 2, 207–223. DOI: https://doi.org/10.1016/S0019-9958(64)90120-2.

[Lad75a] R. E. Ladner. 1975a. On the structure of polynomial time reducibility. *J. ACM* 22, 1, 155–171. DOI: https://doi.org/10.1145/321864.321877.

[Lad75b] R. E. Ladner. 1975b. The circuit value problem is log space complete for P. *SIGACT News* 7, 1, 18–20. DOI: https://doi.org/10.1145/990518.990519.

[LF80] R. E. Ladner and M. J. Fischer. 1980. Parallel prefix computation. *J. ACM* 27, 4, 831–838. DOI: https://doi.org/10.1145/322217.322232.

[Las67] D. Lascar. March. 1967. Cobham's characterization of *L*. In *Topics in the Theory of Computation.* notes of a seminar conducted by R. M. Baer and S. A. Cook, Dept. of mathematics, University of California at Berkeley. (Unpublished).

[LS98] C. A. Lazere and D. E. Shasha. 1998. *Out of Their Minds: The Lives and Discoveries of 15 Great Computer Scientists*. Copernicus, an imprint of Springer-Verlag. ISBN 0-387-98269-8.

[Lee59] C. Y. Lee. 1959. Representation of switching circuits by binary-decision programs. *Bell Syst. Tech. J.* 38, 4, 985–999. DOI: https://doi.org/10.1002/j.1538-7305.1959.tb01585.x.

[Lei93] D. Leivant. 1993. Stratified functional programs and computational complexity. In *Proceedings of the 20th ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages, POPL '93*. ACM, 325–333. ISBN 0897915607. DOI: https://doi.org/10.1145/158511.158659.

[LT79] T. Lengauer and R. E. Tarjan. April 30–May 2. 1979. Upper and lower bounds on time–space tradeoffs. In *Proceedings of the 11th Annual ACM Symposium on Theory of Computing*. ACM, Atlanta, Georgia, 262–277. DOI: https://doi.org/10.1145/800135.804420.

[LT80] T. Lengauer and R. E. Tarjan. 1980. The space complexity of pebble games on trees. *Inf. Process. Lett.* 10, 4–5, 184–188. DOI: https://doi.org/10.1016/0020-0190(80)90136-2.

[LT82] T. Lengauer and R. E. Tarjan. 1982. Asymptotically tight bounds on time–space trade-offs in a pebble game. *J. ACM* 29, 4, 1087–1130. DOI: https://doi.org/10.1145/322344.322354.

[LLL82] A. K. Lenstra, H. W. Lenstra, and L. Lovasz. 1982. *Factoring Polynomials with Rational Coefficients*. Report 82–05. University of Amsterdam, Dept. of Math.

[Lev73] L. A. Levin. 1973. Universal sequential search problems. *Probl. Inf. Transm.* 9, 3, 115–116. [Universal'nye perebornye zadachi. *Problemy Peredachi Informatsii* 9, 3, 115–116, 1973].

[LK72] K. N. Levitt and W. H. Kautz. 1972. Cellular arrays for the solution of graph problems. *Commun. ACM* 15, 9, 789–801. DOI: https://doi.org/10.1145/361573.361576.

[LM64] D. R. Lewis and G. E. Mellen. 1964. Stretching LARC's capability by 100—a new multiprocessor system. In *1964 Symposium on Microelectronics and Large Systems*. Spartan Books, Washington, DC.

[LSH65] P. M. Lewis II, R. E. Stearns, and J. Hartmanis. October 6–8. 1965. Memory bounds for recognition of context-free and context-sensitive languages. In *6th Annual Symposium on Switching Circuit Theory and Logical Design*. IEEE, Ann Arbor, Michigan, 191–202. DOI: https://doi.org/10.1109/FOCS.1965.14.

[Lou79] M. Loui. 1979. *The Space Complexity of Two Pebble Games on Trees*. Technical Report LCS 133. MIT, Cambridge, Massachussetts.

[Lou80] M. C. Loui. 1980. A note on the pebble game. *Inf. Process. Lett.* 11, 1, 24–26. DOI: https://doi.org/10.1016/0020-0190(80)90027-7.

[Luk80] E. M. Luks. 1980. Isomorphism of graphs of bounded valence can be tested in polynomial time. In *Proceedings of the 21st IEEE Symposium on*

Foundations of Computer Science. IEEE Computer Society, Los Angeles, 42–49. DOI: https://doi.org/10.1109/SFCS.1980.24.

[Mag69] G. Mager. 1969. Writing pushdown acceptors. *J. Comput. Syst. Sci.* 3, 3, 276–319. DOI: https://doi.org/10.1016/S0022-0000(69)80017-6.

[Mah07] M. Mahajan. 2007. Polynomial size log depth circuits: between $\mathbf{NC}^1$ and $\mathbf{AC}^1$. *Bull. EATCS* 91, 30–42.

[Mey75] A. R. Meyer. 1975. *Weak Monadic Second-Order Theory of Successor is not Elementary-Recursive*. Lecture Notes in Mathematics, Vol. 453. Springer Verlag, New York, 132–154. DOI: https://doi.org/10.1007/BFb0064872.

[MS72] A. R. Meyer and L. J. Stockmeyer. 1972. The equivalence problem for regular expressions with squaring requires exponential space. In *Proceedings of the 13th IEEE Symposium on Switching and Automata Theory.* 125–129. DOI: https://doi.org/10.1109/SWAT.1972.29.

[Meh74] K. Mehlhorn. April 30–May 2. 1974. Polynomial and abstract subrecursive classes. In R. L. Constable, R. W. Ritchie, J. W. Carlyle, and M. A. Harrison (Eds.), *Proceedings of the 6th Annual ACM Symposium on Theory of Computing*. ACM, Seattle, Washington, 96–109. DOI: https://doi.org/10.1145/800119.803890.

[Meh76] K. Mehlhorn. 1976. Polynomial and abstract subrecursive classes. *J. Comput. Syst. Sci.* 12, 2, 147–178. DOI: https://doi.org/10.1016/S0022-0000(76)80035-9.

[Meh80] K. Mehlhorn. July 14–18. 1980. Pebbling mountain ranges and its application of DCFL-recognition. In *Automata, Languages and Programming, 7th Colloquium*. Lecture Notes in Computer Science, Vol. 85. Springer, Noordweijkerhout, The Netherlands, 422–435. DOI: https://doi.org/10.1007/3-540-10003-2_89.

[Mey79] F. Meyer auf der Heide. July 16–20. 1979. A comparison between two variations of a pebble game on graphs. In *Automata, Languages and Programming, 6th Colloquium*. Lecture Notes in Computer Science, Vol. 71. Springer, Graz, Austria, 411–421. DOI: https://doi.org/10.1007/3-540-09510-1_32.

[Mey81] F. Meyer auf der Heide. 1981. A comparison of two variations of a pebble game on graphs. *Theor. Comput. Sci.* 13, 3, 315–322. DOI: https://doi.org/10.1016/S0304-3975(81)80004-7.

[Mil76] G. L. Miller. 1976. Riemann's Hypothesis and tests for primality. *J. Comput. System Sci.* 13, 300–317. DOI: https://doi.org/10.1016/S0022-0000(76)80043-8.

[Mos52] A. Mostowski. 1952. *Sentences Undecidable in Formalized Arithmetic.* Studies in Logic and the Foundations of Mathematics, Vol. 10. North-Holland. DOI: https://doi.org/10.1016/S0049-237X(08)71704-X.

[MP75] D. E. Muller and F. P. Preperata. April. 1975. Bounds to complexities of networks for sorting and for switching. *J. ACM* 22, 2, 195–201. DOI: https://doi.org/10.1145/321879.321882.

[Neč66] È. Nečiporuk. 1966. On a boolean function. *Doklady of the Academy of the USSR* 169, 4, 765–766. (English translation in *Soviet Mathematics Doklady 7*, 4, 999–1000.)

[NG21] D. T. Nguyen and K. Gaj. 2021. Fast NEON-based multiplication for lattice-based NIST post-quantum cryptography finalists. In *Post-Quantum Cryptography: 12th International Workshop, PQCrypto 2021*. Lecture Notes in Computer Science, Vol. 12841. Springer, Berlin, 234–254. ISBN 978-3-030-81292-8. DOI: https://doi.org/10.1007/978-3-030-81293-5_13.

[Nie64] J. Nievergelt. 1964. Parallel methods for integrating ordinary differential equations. *Commun. ACM* 7, 12, 731–733. DOI: https://doi.org/10.1145/355588.365137.

[Nor09] J. Nordström. 2009. New wine into old wineskins: A survey of some pebbling classics with supplemental results. http://people.csail.mit.edu/jakobn/research/.

[Opp78] D. C. Oppen. 1978. A $2^{2^{2^{pn}}}$ upper bound on the complexity of Presburger arithmetic. *J. Comput. Syst. Sci.* 16, 323–332. DOI: https://doi.org/10.1016/0022-0000(78)90021-1.

[Pap94a] C. H. Papadimitriou. 1994a. On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. Syst. Sci.* 48, 3, 498–532. DOI: https://doi.org/10.1016/S0022-0000(05)80063-7.

[Pap94b] C. H. Papadimitriou. 1994b. *Computational Complexity*. Addison-Wesley. ISBN 978-0-201-53082-7.

[PS82] C. H. Papadimitriou and K. Steiglitz. 1982. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Englewood Cliffs, NJ.

[Par71] R. Parikh. 1971. Existence and feasibility in arithmetic. *J. Symb. Log.* 36, 3, 494–508. DOI: https://doi.org/10.2307/2269958.

[Par73] R. J. Parikh. 1973. Some results on the length of proofs. *Trans. Am. Math. Soc.* 177, 29–36. DOI: https://doi.org/10.2307/1996581.

[PW81] J. B. Paris and A. J. Wilkie. 1981. Δ_0 sets and induction. In W. Guzicki, W. Marek, A. Pelc, and C. Rauszer (Eds.), *Open Days in Model Theory and Set Theory*. Warsaw University, 237–248.

[PWW88] J. B. Paris, A. J. Wilkie, and A. R. Woods. 1988. Provability of the pigeonhole principle and the existence of infinitely many primes. *J. Symb. Log.* 53, 4, 1235–1244. DOI: https://doi.org/10.1017/S0022481200028061.

[Pat72] M. Paterson. 1972. Tape bounds for time-bounded Turing machines. *J. Comput. Syst. Sci.* 6, 2, 116–124. DOI: https://doi.org/10.1016/S0022-0000(72)80017-5.

[Pat76] M. S. Paterson. 1976. An introduction to boolean function complexity. *Astérisque*, *Société Mathématique de France* 38–39, 183–201.

[PH70] M. S. Paterson and C. E. Hewitt. 1970. Comparative schematology. In *Record of the Project MAC Conference on Concurrent Systems and Parallel Computation*. ACM, 119–127. DOI: https://doi.org/10.1145/1344551.1344563.

[PFM74] M. S. Paterson, M. J. Fischer, and A. R. Meyer. 1974. *An Improved Overlap Argument for On-line Multiplication*. *SIAM–AMS Proc. 7*, Amer. Math. Soc., Providence, 97–111.

[PR79] W. J. Paul and R. Reischuk. October 29–31. 1979. On time versus space II. In *20th Annual Symposium on Foundations of Computer Science*. IEEE, San Juan, Puerto Rico, 298–306. DOI: https://doi.org/10.1109/SFCS.1979.30.

[PR81] W. J. Paul and R. Reischuk. 1981. On time versus space II. (Turing machines). *J. Comput. Syst. Sci.* 22, 3, 312–327. DOI: https://doi.org/10.1016/0022-0000(81)90035-0.

[PTC76a] W. J. Paul, R. E. Tarjan, and J. R. Celoni. May 3–5. 1976a. Space bounds for a game of graphs. In *Proceedings of the 8th Annual ACM Symposium on Theory of Computing*. ACM, Hershey, Pennsylvania, 149–160. DOI: https://doi.org/10.1145/800113.803643.

[PTC76b] W. J. Paul, R. E. Tarjan, and J. R. Celoni. 1976b. Space bounds for a game on graphs. *Math. Syst. Theory* 10, 239–251. DOI: https://doi.org/10.1007/BF01683275.

[PTC77] W. J. Paul, R. E. Tarjan, and J. R. Celoni. 1977. Correction to "Space bounds for a game on graphs." *Math. Syst. Theory* 11, 85. DOI: https://doi.org/10.1007/BF01768470.

[Pea67] M. C. Pease. 1967. Matrix inversion using parallel processing. *J. ACM* 14, 4, 757–764. DOI: https://doi.org/10.1145/321420.321434.

[Pea68] M. C. Pease. 1968. An adaptation of the fast Fourier transform for parallel processing. *J. ACM* 15, 2, 252–264. DOI: https://doi.org/10.1145/321450.321457.

[Per05] S. Perron. August 22–25. 2005. A propositional proof system for log space. In C. L. Ong (Ed.), *Computer Science Logic, 19th International Workshop, CSL 2005, 14th Annual Conference of the EACSL*, *Proceedings*. Lecture Notes in Computer Science, Vol. 3634. Springer, Oxford, UK, 509–524. DOI: https://doi.org/10.1007/11538363_35.

[Per09] S. Perron. 2009. *Power of Non-Uniformity in Proof Complexity*. Ph.D. thesis. Department of Computer Science, University of Toronto.

[Pet62] C. A. Petri. 1962. *Kommunikation mit Automaten*. Ph.D. thesis. Rheinisch-Westfalisches Institut fur Instrumentelle Mathematik, Universität Bonn.

[Pip] N. Pippenger. Personal communication.

[Pip77] N. Pippenger. 1977. Fast simulation of combinational logic networks by machines without random access storage. In *15th Allerton Conference on Communication Control and Computing*, 25–33.

[Pip79] N. Pippenger. October 29–31. 1979. On simultaneous resource bounds (preliminary version). In *20th Annual Symposium on Foundations of Computer Science*. IEEE, San Juan, Puerto Rico, 307–311. DOI: https://doi.org/10.1109/SFCS.1979.29.

[PV76] N. Pippenger and L. G. Valiant 1976. Shifting graphs and their applications. *J. Assoc. Comput. Mach.* 23, 423–432. DOI: https://doi.org/10.1145/321958.321962.

[PF79] N. J. Pippenger and M. J. Fischer. April. 1979. Relations among complexity measures. *J. Assoc. Comput. Mach.* 26, 2, 361–381. DOI: https://doi.org/10.1145/322123.322138.

[PBI93] T. Pitassi, P. Beame, and R. Impagliazzo. 1993. Exponential lower bounds for the pigeonhole principle. *Comput. Complex.* 3, 97–140. DOI: https://doi.org/10.1007/BF01200117.

[PS76] V. R. Pratt and L. J. Stockmeyer. 1976. A characterization of the power of vector machines. *J. Comput. Syst. Sci.* 12, 2, 198–221. DOI: https://doi.org/10.1016/S0022-0000(76)80037-2.

[PS78] V. Pratt and L. Stockmeyer. 1978. A characterization of the power of vector machines. *J. Comput. Syst. Sci.* 12, 198–221.

[PRS74] V. R. Pratt, M. O. Rabin, and L. J. Stockmeyer. April 30–May 2. 1974. A characterization of the power of vector machines. In R. L. Constable, R. W. Ritchie, J. W. Carlyle, and M. A. Harrison (Eds.), *Proceedings of the 6th Annual ACM Symposium on Theory of Computing*. ACM, Seattle, Washington, 122–134. DOI: https://doi.org/10.1145/800119.803892.

[Pra65] D. Prawitz. 1965. *Natural Deduction, A Proof-Theoretical Study*. Vol. 3: Stockholm Studies in Philosophy. Almqvist & Wiskell.

[Pud86] P. Pudlák. 1986. On the length of proofs of finitistic consistency statements in first order theories. In J. Paris, A. Wilkie, and G. Wilmers (Eds.), *Logic Colloquium '84*, Vol. 120: Studies in Logic and the Foundations of Mathematics, North-Holland, 165–196. DOI: https://doi.org/10.1016/S0049-237X(08)70462-2.

[Pud87a] P. Pudlák. 1987a. The hierarchy of boolean circuits. *Comput. Artif. Intell.* 6, 5, 449–468.

[Pud87b] P. Pudlák. 1987b. Improved bounds to the length of proofs of finitistic consistency statements. In *Logic and Combinatorics* (Arcata, Calif., 1985), Vol. 65: Contemporary Mathematics. Am. Math. Soc., Providence, RI, 309–332. DOI: https://doi.org/10.1090/conm/065/891256.

[Rab59] M. O. Rabin. 1959. Speed of computation and classification of recursive sets. In *Third Convention Science Society*. Israel, 1–2.

[Rab60] M. O. Rabin. 1960. Degree of difficulty of computing a function and a partial ordering of recursive sets. Tech. Rep. No. 1, O.N.R., Jerusalem.

[Rab76] M. O. Rabin. 1976. Probabilistic algorithms. In J. F. Traub (Ed.), *Algorithms and Complexity, New Directions and Recent Trends*. Academic Press, New York, 21–39.

[Rab77] M. O. Rabin. September. 1977. Complexity of computations. *Comm. ACM* 20, 9, 625–633. DOI: https://doi.org/10.1145/359810.359816.

[RF93] B. R. Rau and J. A. Fisher. 1993. Instruction-level parallel processing: History, overview, and perspective. *J. Supercomput.* 7, 1–2, 9–50. DOI: https://doi.org/10.1007/BF01205181.

[Raz91] A. Razborov. 1991. Lower bounds for deterministic and nondeterministic branching programs. In *Proceedings of the 8th International Symposium on Fundamentals of Computation Theory*. 47–60. DOI: https://doi.org/10.1007/3-540-54458-5_49.

[Raz98] A. A. Razborov. 1998. Lower bounds for the polynomial calculus. *Comput. Complex.* 7, 4, 291–324. DOI: https://doi.org/10.1007/s000370050013.

[Rec76] R. A. Reckhow. 1976. *On the Lengths of Proofs in the Propositional Calculus*. Ph.D. thesis. Department of Computer Science, University of Toronto.

[Rei82] J. H. Reif. May 5–7. 1982. Symmetric complementation. *Proceedings of the 14th ACM Symposium on Theory of Computing*. San Francisco, CA, 201–214. DOI: https://doi.org/10.1145/800070.802193.

[Rei83] J. H. Reif. November 7–9. 1983. Logarithmic depth circuits for algebraic functions. In *24th Annual Symposium on Foundations of Computer Science*. IEEE, Tucson, Arizona, 138–145. DOI: https://doi.org/10.1109/SFCS.1983.29.

[Rei05] O. Reingold. May 22–24. 2005. Undirected ST-connectivity in log-space. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*. ACM, Baltimore, MD, 376–385. DOI: https://doi.org/10.1145/1060590.1060647.

[Rei08] O. Reingold. 2008. Undirected connectivity in log-space. *J. ACM* 55, 4, Article 17, 1–24. DOI: https://doi.org/10.1145/1391289.1391291.

[RS82] S. Reisch and G. Schnitger. 1982. Three applications of Kolmorgorov complexity. *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science.* IEEE Computer Society, Los Angeles. 45–52.

[Rit60] R. W. Ritchie. 1960. *Classes of Recursive Functions of Predictable Complexity*. Doctoral Dissertation, Princeton University.

[Rit63] R. W. Ritchie. 1963. Classes of predictably computable functions. *Trans. Am. Math. Soc.* 106, 139–173. DOI: https://doi.org/10.2307/1993719.

[Riv77] R. L. Rivest. June. 1977. The necessity of feedback in minimal monotone combinational circuits. *IEEE Trans. Comput*. 26, 6, 606–607. DOI: https://doi.org/10.1109/TC.1977.1674886.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. February. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Comm. ACM* 21, 2, 120–126. DOI: https://doi.org/10.1145/359340.359342.

[Ros61] H. E. Rose. 1961. On the consistency and undecidability of recursive arithmetic. *Zeitschr. f. math. Logik and Grundlogen d. Math. Bd.* 7, S. 124–135.

[Rus78] R. M. Russell. 1978. The CRAY-1 computer system. *Commun. ACM* 21, 1, 63–72. DOI: https://doi.org/10.1145/359327.359336.

[Ruz79a] W. L. Ruzzo. October 29–31. 1979a. On uniform circuit complexity (extended abstract). In *20th Annual Symposium on Foundations of Computer Science*. IEEE, San Juan, Puerto Rico, 312–318. DOI: https://doi.org/10.1109/SFCS.1979.31.

[Ruz79b] W. L. Ruzzo. April 30–May 2. 1979b. Tree-size bounded alternation. In *Proceedings of the 11th Annual ACM Symposium on Theory of Computing*. ACM, Atlanta, Georgia, 352–359. DOI: https://doi.org/10.1145/800135.804428.

[Ruz81] W. L. Ruzzo. 1981. On uniform circuit complexity. *J. Comput. System Sci.* 22, 3, 365–383. DOI: https://doi.org/10.1016/0022-0000(81)90038-6.

[Sav72] J. E. Savage. 1972. Computational work and time on finite machines. *J. ACM* 19, 4, 660–674. DOI: https://doi.org/10.1145/321724.321731.

[Sav76] J. E. Savage. 1976. *The Complexity of Computing.* Wiley, New York.

[Sav77] C. D. Savage. 1977. *Parallel Algorithms for Graph Theoretic Problems*. Ph.D. thesis. University of Illinois at Urbana-Champaign.

[Sav69] W. J. Savitch. May 5–7. 1969. Deterministic simulation of non-deterministic Turing machines (detailed abstract). In P. C. Fischer, S. Ginsburg, and M. A. Harrison (Eds.), *Proceedings of the 1st Annual ACM Symposium on Theory of Computing*. ACM, Marina del Rey, CA, 247–248. DOI: https://doi.org/10.1145/800169.805439.

[Sav70] W. J. Savitch. 1970. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.* 4, 2, 177–192. DOI: https://doi.org/10.1016/S0022-0000(70)80006-X.

[SS76] W. J. Savitch and M. J. Stimson. April. 1976. The complexity of time bounded recursive computations. In *Proceedings of the 1976 Conference on Information Sciences and Systems*. The Johns Hopkins University, Baltimore, MD. Journal version: Time bounded random access machines with parallel processing. *J. ACM* 26, 1 (1979), 103–118. DOI: https://doi.org/10.1145/322108.322119.

[SS79] W. Savitch and M. Stimson. January. 1979. Time bounded random access machines with parallel processing. *J. Assoc. Comput. Mach.* 26, 103–118.

[Sch76] C. P. Schnorr 1976. The network complexity and the Turing machine complexity of finite functions. *Acta Informatica* 7, 95–107. DOI: https://doi.org/10.1007/BF00265223.

[Sch70] A. Schönhage. 1970. Universelle Turing speicherung. In Dürr and Hotz (Eds.), *Automaten Theorie und Formale Sprachen*. Bibliographisches Institut, Mannheim, 369–383.

[Sch79] A. Schönhage. 1979. Storage modification machines. Technical Report, Mathematisches Institut, Universität Tübingen, Germany.

[Sch80a] A. Schönhage. 1980a. Storage modification machines. *SIAM J. Comput.* 9, 3, 490–508. DOI: https://doi.org/10.1137/0209036.

[SS71] A. Schönhage and V. Strassen. 1971. Schnelle Multiplication grosser Zahlen. *Computing* 7, 281–292. DOI: https://doi.org/10.1007/BF02242355.

[Sch80b] J. T. Schwartz. October. 1980b. Probabilistic algorithms for verification of polynomial identifies. *J. ACM* 27, 4, 701–717.

[Sch80c] J. T. Schwartz. October. 1980c. Ultracomputers. *ACM Trans. on Prog. Languages and Systems* 2, 4, 484–521. DOI: https://doi.org/10.1145/357114.357116.

[SMB83] A. L. Selman, X. Mei-Rui, and R. V. Book. 1983. Positive relativizations of complexity classes. *SIAM J. Comput.* 12, 3, 565–579. DOI: https://doi.org/10.1137/0212037.

[Sha81] A. Shamir. July. 1981. On the generation of cryptographically strong pseudo random sequences. In *Proceedings of the 8th Int. Colloquium on Automata, Languages, and Programming*. Lecture Notes in Computer Science. Vol. 115, Springer, Verlag, New York, 544–550. DOI: https://doi.org/10.1007/3-540-10843-2_43.

[Sha49] C. E. Shannon. 1949. The synthesis of two-terminal switching circuits. *BSTJ* 28, 59–98. DOI: https://doi.org/10.1002/j.1538-7305.1949.tb03624.x.

[SL95] D. E. Shasha and C. Lazare. 1995. *Out of Their Minds: The Lives and Discoveries of 15 Great Computer Scientists*. Copernicus, New York.

[She67] G. S. Shedler. 1967. Parallel numerical methods for the solution of equations. *Commun. ACM* 10, 5, 286–291. DOI: https://doi.org/10.1145/363282.363301.

[SS63] J. C. Shepherdson and H. E. Sturgis. April. 1963. Computability of recursive functions. *J. ACM* 10, 2, 217–255. DOI: https://doi.org/10.2307/2271277.

[Sim77] J. Simon. May. 1977. On feasible numbers. In *Proceedings of the 9th Annual ACM Symposium on Theory of Computing*. 195–207. DOI: https://doi.org/10.1145/800105.803409.

[SGH78] J. Simon, J. Gill, and J. Hunt. October 16–18. 1978. On tape bounded probabilistic turing machine transducers (Extended abstract). In *Proceedings of the 19th Annual Symposium on Foundations of Computer Science*. IEEE, Ann Arbor, Michigan, 107–112. DOI: https://doi.org/10.1109/SFCS.1978.27.

[Sip92] M. Sipser. May 4–6. 1992. The history and status of the P versus NP question. In S. R. Kosaraju, M. Fellows, A. Wigderson, and J. A. Ellis (Eds.), *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*. ACM, Victoria, British Columbia, Canada, 603–618. DOI: https://doi.org/10.1145/129712.129771.

[Ske04] A. Skelley. September 20–24. 2004. A third-order bounded arithmetic theory for PSPACE. In J. Marcinkowski and A. Tarlecki (Eds.), *Computer Science Logic, 18th International Workshop, CSL 2004, 13th Annual Conference of the EACSL, Proceedings*. Lecture Notes in Computer Science. Vol. 3210, Springer, Karpacz, Poland, 340–354. DOI: https://doi.org/10.1007/978-3-540-30124-0_27.

[ST11] A. Skelley and N. Thapen. 2011. The provably total search problems of bounded arithmetic. *Proc. Lond. Math. Soc.* 103, 1, 106–138. DOI: https://doi.org/10.1112/plms/pdq044.

[Sko23] T. Skolem. 1923. *Begründung der Elementaren Arithmetik*. Videnskapsselskapets Skrifter. I. Matematisk-naturvidenskabelig Klasse. No. 6. Utgit for Fridtjof Nansens Fond.

[Slo82] D. L. Slotnick. 1982. The conception and development of parallel processors: A personal memoir. *IEEE Ann. Hist. Comput.* 4, 1, 20–30. DOI: https://doi.org/10.1109/MAHC.1982.10003.

[SBM63] D. L. Slotnick, W. C. Borck, and R. C. McReynolds. 1963. The SOLOMON computer – a preliminary report. In A. A. Barnum and M. A. Knapp (Eds.), *Proceedings of the 1962 Workshop on Computer Organization*. Spartan, Washington, DC, 62–92.

[Sma82a] S. Smale. 1982a. On the average speed of the simplex method of linear programming. Preprint.

[Sma82b] S. Smale. 1982b. The problem of the average speed of the simplex method. Preprint.

[Smu61] R. M. Smullyan. 1961. *Theory of Formal Systems, Revised Edition*. Annals of Mathematical Studies. Vol. 47, Princeton.

[SS77] R. Solovay and V. Strassen. 1977. A fast monte-carlo test for primality. *SIAM J. Comput*. 6, 84–85. DOI: https://doi.org/10.1137/0206006.

[Sta77] R. Statman. 1977. Complexity of derivations from quantifier-free horn formulae, mechanical introduction of explicit definitions, and refinement of completeness theorems. *Proceedings of the Logic Colloquium* 76 (Gandy and Hyland, Editors), *Studies in Logic and Foundations of Mathematics*, Vol. 87, North-Holland, Amsterdam.

[SHL65] R. E. Stearns, J. Hartmanis, and P. M. Lewis. October 6–8. 1965. Hierarchies of memory limited computations. In *Proceedings of the 6th Annual Symposium on Switching Circuit Theory and Logical Design*. IEEE, Ann Arbor, Michigan, 179–190. DOI: https://doi.org/10.1109/FOCS.1965.11.

[Sto74] L. J. Stockmeyer. 1974. *The complexity of decision problems in automata theory and logic*. Doctoral Thesis. Dept. of Electrical Eng., Report TR-133, MIT Laboratory for Computer Science, MIT, Cambridge, MA.

[Sto75] L. J. Stockmeyer. 1975. *The Polynomial-Time Hierarchy*. Technical Report RC5379. IBM Thomas J. Watson Research Center, Yorktown Heights, NY.

[Sto79] L. J. Stockmeyer. 1979. Classifying the computational complexity of problems. Research Report RC 7606, Math. Sciences Dept., IBM T. J. Watson Research Center, Yorktown Heights, NY.

[Sto71] H. S. Stone. 1971. Parallel processing with the perfect shuffle. *IEEE Trans. Comput.* 20, 2, 153–161. DOI: https://doi.org/10.1109/T-C.1971.223205.

[Str69] V. Strassen. 1969. Gaussian elimination is not optimal. *Numer. Math.* 13, 4, 354–356. DOI: https://doi.org/10.1007/BF02165411.

[Str73] V. Strassen. 1973. Die Berechnungskomplexität von elementarsymmetrischen Funktionen und von Interpolationskoeffizienten. *Numer. Math.* 20, 238–251. DOI: https://doi.org/10.1007/BF01436566.

[Sud75] I. H. Sudborough. 1975. A note on tape-bounded complexity classes and linear context-free languages. *J. ACM* 22, 4, 499–500. DOI: https://doi.org/10.1145/321906.321913.

[Sud77] I. H. Sudborough. September 5–9. 1977. Time and tape bounded auxiliary pushdown automata. In *Proceedings of the Mathematical Foundations of Computer Science 1977, 6th Symposium*. Lecture Notes in Computer Science, Vol. 53, Springer, Tatranska Lomnica, Czechoslovakia, 493–503. DOI: https://doi.org/10.1007/3-540-08353-7_172.

[Sud78] I. H. Sudborough. 1978. On the tape complexity of deterministic context-free languages. *J. ACM* 25, 3, 405–414. DOI: https://doi.org/10.1145/322077.322083.

[Sze88] R. Szelepcsényi. 1988. The method of forced enumeration for nondeterministic automata. *Acta Inform.* 26, 3, 279–284. DOI: https://doi.org/10.1007/BF00299636.

[Tai05] M. Taitslin. 2005. An example of a problem from PTIME and not in NLogSpace. In *Proceedings of the Tver State University*. *Appl. Math.* 6, 12, 5–22.

[Tom78] M. Tompa. July. 1978. *Time-space tradeoffs for straight-line and branching programs.* Tech. Rep. 122/78, Dept. Computer Science, Univ. of Toronto.

[Too63] A. L. Toom. 1963. The complexity of a scheme of functional elements realizing the multiplication of integers. *Sov. Math. Dokl.* 3, 4, 714–716.

[Tow90] M. Townsend. 1990. Complexity for type-2 relations. *Notre Dame J. Form. Log.* 31, 2, 241–262. DOI: https://doi.org/10.1305/ndjfl/1093635419.

[Tra84] B. A. Trakhtenbrot. 1984. A survey of Russian approaches to perebor (brute-force searches) algorithms. *IEEE Ann. Hist. Comput.* 6, 4, 384–400. DOI: https://doi.org/10.1109/MAHC.1984.10036.

[Tse70] G. S. Tseitin. 1970. On the complexity of derivation in propositional calculus. *Semin. Math., V. A. Steklov Math. Inst., Leningrad* 8, 115–125; translation from 1968. *Zap. Nauchn. Semin. Leningr. Otd. Mat. Inst. Steklova* 8, 234–259.

[Tur36] A. M. Turing. 1936. On computable numbers, with an application to the Entscheidungsproblem. *Proc. Lond. Math. Soc. Second Series* 42, 230–265. DOI: https://doi.org/10.1112/plms/s2-42.1.230.

[Tur37] A. M. Turing. 1937. On computable numbers with an application to the Entscheidnungsproblem. *Proc. London Math. Soc. Ser.* 2, 42 (1936–7), 230–265. A correction. *ibid*. 43, 544–546.

[Val79a] L. G. Valiant. 1979a. The complexity of enumeration and reliability problems. *SIAM J. Comput.* 8, 410–421. DOI: https://doi.org/10.1137/0208032.

[Val79b] L. G. Valiant. 1979b. The complexity of computing the permanent. *Theor. Comput. Sci.* 8, 189–202. DOI: https://doi.org/10.1016/0304-3975(79)90044-6.

[Val82] L. G. Valiant. 1982. Parallel Computation. In *Proceedings of the 7th IBM Japan Symposium*. Academic 6 Scientific Programs, IBM Japan, Tokyo.

[VS81] L. G. Valiant, S. Skyum. 1981. Fast Parallel Computation Of Polynomials Using Few Processors. Lecture Notes in Computer Science. Vol. 118, Springer, Berlin, Heidelberg, 132–139. DOI: https://doi.org/10.1007/3-540-10856-4_79.

[Ven91] H. Venkateswaran. 1991. Properties that characterize LOGCFL. *J. Comput. Syst. Sci.* 43, 2, 380–404. DOI: https://doi.org/10.1016/0022-0000(91)90020-6.

[Vis83] U. Vishkin. April. 1983. *Synchronous Parallel Computation—A Survey*. Technical Report 71. Courant Institute, New York University.

[vBV80] B. von Braunmühl and R. Verbeek. October 13–15. 1980. A recognition algorithm for deterministic CFLS optimal in time and space. In *Proceedings of the 21st Annual Symposium on Foundations of Computer Science.* IEEE, Syracuse, New York, 411–420. DOI: https://doi.org/10.1109/SFCS.1980.8.

[vNeu27] J. von Neumann. 1927. Zur Hilbertschen Beweistheorie. *Mathematische Zeitschrift* 26, 1–46.

[vNeu53] J. von Neumann. 1953. A certain zero-sum two-person game equivalent to the optimal assignment problem. In H. W. Kuhn and A. W. Tucker (Eds.), *Contributions to the Theory of Games*, Vol. II, Princeton University Press, 5–12. DOI: https://doi.org/10.1515/9781400881970-002.

[Wan62] H. Wang. 1962. Dominoes and the AEA case of the decision problem. In *Proceedings of the Symposium on Mathematical Theory of Automata at Polytechnic Institute of Brooklyn*. Polytechnic Press, 23–55.

[Weg87] I. Wegener. 1987. *The Complexity of Boolean Functions*. Wiley-Teubner Series in Computer Science. B. G. Teubner & John Wiley, Stuttgart.

[Weg00] I. Wegener. 2000. *Branching Programs and Binary Decision Diagrams*. SIAM Monographs on Discrete Mathematics and Applications. SIAM, Philadelphia, PA.

[Weg05] I. Wegener. 2005. *Complexity Theory – Exploring the Limits of Efficient Algorithms*. Springer. ISBN 9783540210450. DOI: https://doi.org/10.1007/3-540-27477-4.

[Weh10] D. Wehr. 2010. Pebbling and branching programs solving the tree evaluation problem. MSc research paper, Department of Computer Science, University of Toronto. February. arXiv:1002.4676.

[Weh11] D. Wehr. 2011. Lower bound for deterministic semantic-incremental branching programs solving GEN. arXiv:1101.2705.

[Weh12] D. Wehr. 2012. Exact size of the smallest min-depth branching programs solving the tree evaluation problem. http://www.cs.toronto.edu/ wehr/research_docs/thrifty_and_read-once_exact_size.pdf.

[Wig06] A. Wigderson. 2006. $\mathcal{P}$, $\mathcal{NP}$ and mathematics—a computational complexity perspective. In *Proceedings of the ICM*. Vol. 6, 665–712.

[Wig09] A. Wigderson. 2009. Knowledge, creativity and P versus NP. http://www.math.ias.edu/avi/PUBLICATIONS/MYPAPERS/AW09/AW09.pdf.

[Wil85] R. E. Wilber. May 6–8. 1985. White pebbles help. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing.* ACM, Providence, Rhode Island, 103–112. DOI: https://doi.org/10.1145/22145.22157.

[Wil88] R. E. Wilber. 1988. White pebbles help. *J. Comput. Syst. Sci.* 36, 2, 108–124. DOI: https://doi.org/10.1016/0022-0000(88)90023-2.

[Wil51] M. V. Wilkes. July. 1951. The best way to design an automatic calculating machine. In *Proceedings of the Manchester University Computer Inaugural Conference*. Tillotsons (Bolton), Ltd, Manchester, England, 16–18.

[WB72] W. A. Wulf and G. Bell. December 5–7. 1972. C.mmp: A multi-mini-processor. In *American Federation of Information Processing*

Societies: Proceedings of the AFIPS '72 Fall Joint Computer Conference. Anaheim, California, USA—Part II, Vol. 41 of AFIPS Conference Proceedings. AFIPS/ACM/Thomson Book Company, Washington, DC, 765–777. DOI: https://doi.org/10.1145/1480083.1480098.

[Wyl79] J. C. Wyllie. 1979. *The complexity of parallel computations*. Ph.D. thesis and TR-79-387. Dept. of Computer Science, Cornell University.

[Yam62] H. Yamada. 1962. Real time computation and recursive functions not real-time computable. *IRE Trans. Electron Comput.* EC-11, 753–760. DOI: https://doi.org/10.1109/TEC.1962.5219459.

[Yan83] M. Yannakakis. November 7–9. 1983. A polynomial algorithm for the MIN CUT linear arrangement of trees (Extended abstract). In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science.* IEEE, Tucson, Arizona, 274–281. DOI: https://doi.org/10.1109/SFCS.1983.2.

[Yan85] M. Yannakakis. 1985. A polynomial algorithm for the min-cut linear arrangement of trees. *J. ACM* 32, 4, 950–988. DOI: https://doi.org/10.1145/4221.4228.

[Yao82] A. C. Yao. 1982. Theory and application of trapdoor functions (Extended abstract). In *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science.* IEEE Computer Society, Los Angeles, 80–91. DOI: https://doi.org/10.1109/SFCS.1982.45.

[You67] D. H. Younger. 1967. Recognition and parsing of context-free languages in time n^3. *Inform. Control* 10, 2, 189–208. DOI: https://doi.org/10.1016/S0019-9958(67)80007-X.

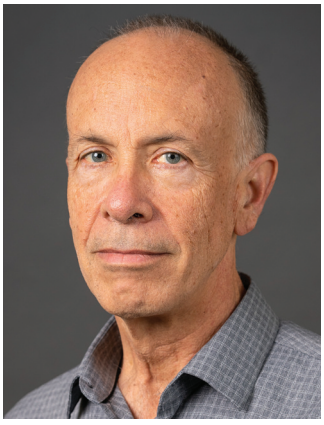
[Zam97] D. Zambella. 1997. End extensions of models of linearly bounded arithmetic. *Ann. Pure Appl. Log.* 88, 2–3, 263–277. DOI: https://doi.org/10.1016/S0168-0072(97)00026-2.

Contributors' Biographies

Paul Beame

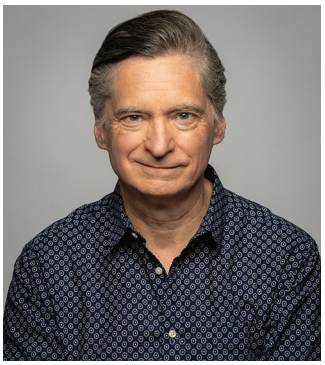
Paul Beame is a Professor and Associate Director in the Paul G. Allen School of Computer Science & Engineering at the University of Washington. He received his Ph.D. from the University of Toronto where his doctoral research on lower bounds and algorithms for parallel computation was supervised by Professor Cook. He is a Fellow of the ACM and former Chair of the ACM Special Interest Group on Algorithms and Computation Theory (SIGACT) and of the IEEE Computer Society Technical Committee on the Mathematical Foundations of Computing (TCMF). Professor Beame's research focuses on pure and applied computational complexity and proof complexity with particular emphases on complexity lower bounds and on applications in AI, databases, SAT-solving, and formal reasoning.

Sam Buss



Sam Buss is a Professor of Mathematics and Computer Science at the University of California, San Diego. He received his bachelor's from Emory University and his Ph.D. from Princeton University. After postdoctoral positions at the Mathematical Sciences Research Institute and the University of Berkeley, Buss has been a faculty member at UC San Diego since 1988. His research is in the areas of proof complexity, bounded arithmetic, proof theory, mathematical logic, and theory of computer science. Buss received the Bolzano Award in 2017 and was the ASL Godel Lecturer in 2019.

Bruce M. Kapron



Bruce M. Kapron is a Professor in the Computer Science Department at the University of Victoria. He received his Ph.D. from the University of Toronto, under the supervision of Steve Cook, in 1991. His research interests include complexity theory, computability in higher types, cryptography, and logic. He has been a Fellow of the Institute of Advanced Studies of *Alma Mater Studorium—Università di Bologna*, a Distinguished Professor of the *Fondation Sciences Mathématiques de Paris*, and has been a long-term visitor at institutions including Carnegie-Mellon University, Stanford University, the Institute for Advanced Study, and the Simons Institute for the Theory of Computing.

Jan Krajíček

Jan Krajíček is Professor of Mathematical Logic in the Faculty of Mathematics and Physics at Charles University, Prague. He contributed novel ideas to the fields of proof complexity and mathematical logic and has authored around 85 papers and three books. He held long-term visiting positions at the University of Illinois in Champaign-Urbana, the University of Toronto, Oxford University, the Institute for Advanced Study in Princeton, and the Isaac Newton Institute in Cambridge. He was also a researcher at the Mathematical Institute of the Academy of Sciences of the Czech Republic (1985–2012). He is a member of the Academia Europaea and of the Learned Society of the Czech Republic. He has been an invited speaker at the European Congress of Mathematicians 2004 and at the quadrennial International Congresses of Logic, Methodology and Philosophy of Science (1995, 2007, 2019).

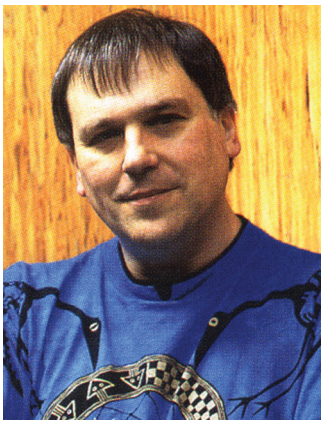
Pierre McKenzie

Pierre McKenzie obtained his Ph.D. from the University of Toronto in 1984 under the guidance of Steve Cook and Allan Borodin. He then joined the *Université de Montréal* where he spent his career, including as head of the *Département d'informatique et de recherche opérationnelle* in 2001–2005. He visited several institutions for extended periods, in particular the University of British Columbia, the University of Tübingen (as holder of a *Deutsche Forchungsgemainschaft* guest professorship), and *École Normale Supérieure de Cachan* (as 2013–2016 holder of a French *Fondation Digiteo* research chair). His main research interests are complexity theory, specifically the study of low-level complexity classes (of problems solvable in polynomial time) and some aspects of formal verification.

Christos Harilaos Papadimitriou

Christos Harilaos Papadimitriou is the Donovan Family Professor of Computer Science at Columbia University. Before joining Columbia in 2017, he was a professor at UC Berkeley for the previous 22 years, and before that he taught at Harvard, MIT, NTU Athens, Stanford, and UCSD. He has written five textbooks and many articles on algorithms and complexity, and their applications to optimization, databases, control, AI, robotics, economics and game theory, the Internet, evolution, and the brain. He holds a PhD from Princeton (1976), and eight honorary doctorates, including from ETH, University of Athens, EPFL, and *Université de Paris Dauphine*. He is a member of the National Academy of Sciences of the US, the American Academy of Arts and Sciences, and the National Academy of Engineering, and he has received the Knuth prize, the Gödel prize, the von Neumann medal, as well as the 2018 Harvey Prize from the Technion. In 2015 the president of the Hellenic Republic named him Commander of the Order of the Phoenix. He has also written three novels: "Turing", "Logicomix" and his latest, "Independence."

Nicholas Pippenger



Nicholas Pippenger was born in Abington, PA, in 1947. He received the B.S. degree in Natural Science from Shimer College in 1965. He received the B.S., M.S., and Ph.D. degrees, all in Electrical Engineering, from the Massachusetts Institute of Technology in 1967, 1969, and 1974, respectively. He worked for the MIT Instrumentation Laboratory from 1969 to 1973. He worked for IBM at the Thomas J. Watson Research Center from 1973 to 1980 and at the San Jose Research Laboratory from 1980 to 1989, where he was named an IBM Fellow in 1987. He was Professor of Computer Science at the University of British Columbia from 1988 to 2003, where he was appointed to a Canada Research Chair in 2001. He was Professor of Computer Science at Princeton University, Princeton, NJ, from 2003 to 2006. He was Professor of Mathematics at Harvey Mudd College, Claremont, CA, from 2006 to 2021, where he is currently Emeritus Professor of Mathematics. His research interests center in discrete mathematics and probability but also extend into communication theory and theoretical computer science. He is the author of *Theories of Computability*, published by Cambridge University Press in 1997. Dr. Pippenger is a Fellow of the Royal Society of Canada (Academy of Science), a Fellow of the Institute of Electrical and Electronics Engineers (IEEE), a Fellow of the Association for Computing Machinery (ACM), and a Fellow of the American Mathematical Society (AMS).

Michelle Waitzman

Michelle Waitzman is the author of four nonfiction books and a professional writer, ghostwriter, editor, and plain language trainer. Before her writing career, Michelle worked in TV production and corporate communications. She lives in Toronto, Canada, just a couple of blocks from the shores of Lake Ontario with her husband and two mixed-breed rescue dogs.

Index

Logic, Automata, and Computational Complexity
The Works of Stephen A. Cook

Bruce M. Kapron (Editor)

Professor Stephen A. Cook is a pioneer of the theory of computational complexity. His work on NP-completeness and the P vs. NP problem remains a central focus of this field. Cook won the 1982 Turing Award for "his advancement of our understanding of the complexity of computation in a significant and profound way." This volume includes a selection of seminal papers embodying the work that led to this award, exemplifying Cook's synthesis of ideas and techniques from logic and the theory of computation including NP-completeness, proof complexity, bounded arithmetic, and parallel and space-bounded computation. These papers are accompanied by contributed articles by leading researchers in these areas, which convey to a general reader the importance of Cook's ideas and their enduring impact on the research community. The book also contains biographical material, Cook's Turing Award lecture, and an interview. Together these provide a portrait of Cook as a recognized leader and innovator in mathematics and computer science, as well as a gentle mentor and colleague.

ABOUT ACM BOOKS

ACM Books is a series of high-quality books published by ACM for the computer science community. ACM Books publications are widely distributed in print and digital formats by major booksellers and are available to libraries and library consortia. Individual ACM members may access ACM Books publications via separate annual subscription.

BOOKS.ACM.ORG

ISBN 979-8-4007-0777-3