# LOWER BOUNDS FOR UNIFORM CONSTANT DEPTH CIRCUITS

BY VIVEK KASHINATH GORE

**A dissertation submitted to the**

**Graduate School—New Brunswick**

**Rutgers, The State University of New Jersey**

**in partial fulfillment of the requirements**

**for the degree of**

**Doctor of Philosophy**

**Graduate Program in Computer Science**

**Written under the direction of**

**Professor Eric Allender**

**and approved by**

_____

_____

_____

_____

**New Brunswick, New Jersey**

**May, 1993**

## ABSTRACT OF THE DISSERTATION

# Lower Bounds for Uniform Constant Depth Circuits

by Vivek Kashinath Gore, Ph.D.

Dissertation Director: Professor Eric Allender

Boolean circuits were introduced in complexity theory to provide a model for parallel computation. A big advantage of studying Boolean circuits is that they can be viewed as simple combinatorial objects and thus allow us to use many algebraic and combinatorial techniques to derive upper and lower bounds on their computational power. The relationship between Boolean circuits and the traditional Turing machine model helps us to translate the bounds obtained for circuits into bounds for the Turing machine model which is otherwise hard to analyze. In this thesis we study the power of certain kinds of uniform constant depth circuits and provide upper and lower bounds.

When we talk about Turing machines, the most important resources are *time* and *space*. The most important resources in case of Boolean circuits are *size* and *depth*. Circuits that have constant depth and polynomial size have been studied intensely in the last decade or so. The class $AC^0$, consisting of languages accepted by constant depth, polynomial size circuits that have NOT gates, and unbounded fan-in AND and OR gates is very well understood and good lower bounds are known for it. On the other hand, the class ACC, that consists of languages accepted by constant depth, polynomial size circuits that have NOT gates, and unbounded fan-in AND, OR and $MOD_m$ gates, for various moduli $m$, is not well understood at all. It is consistent with our knowledge that an ACC circuit family can compute the Satisfiability problem.

In this thesis, we prove that we need exponential size to compute the the Permanent function using uniform circuit families that have constant depth and consist of NOT gates, and unbounded fan-in AND, OR and MOD gates. We also show that there are languages in PP and $C_=P$ that cannot be recognized by uniform ACC type circuits of subsubexponential size.

We also study the question of finding a strong separation between NP and $AC^0$. We show that non-relativizing proof techniques result from any answer to the question "Does NP contain sets that are immune to $AC^0$?." It is also shown that $P^{PP}$ contains sets that are immune to ACC and hence to $AC^0$.

Neil Jones introduced logspace reductions as a tool for studying the relative complexity of problems in P. He also introduced a restricted version of logspace reducibility, called logspace bounded rudimentary reductions to study small complexity classes like Dspace($\log n$). We show that these logspace bounded rudimentary reductions characterize uniform $AC^0$.

# Acknowledgements

I would like to thank everybody who helped me during these five years of my studies at Rutgers.

I am very grateful to my advisor, Eric Allender, for his guidance. He has been a "dream advisor" of sorts; very patient, always willing to answer questions, always ready to listen to my ideas that were often nonsensical, and most importantly, very accessible.

I wish to thank Endre Szemerédi, Ann Yasuhara and Richard Beigel for serving on my committee.

Special thanks are due to Richard Beigel for some key ideas that helped in proving the results in Chapter 3. I am greatly indebted to him for his help in proving the results in Theorems 3.5 and 3.8.

I wish to thank all my teachers for their help. Ann Yasuhara taught me the foundations of logic and computability, and Eric Allender taught me the fundamentals of complexity theory. The other teachers whose courses were extremely rewarding include Ravi Boppana, Vašek Chvátal, Mike Fredman, Jeff Kahn, Mike Saks, Diane Souvaine and Endre Szemerédi, among others.

It is very hard for me to imagine how things would have been had it not been for my cousin Jayashree Kale, her husband Dr. Prabhakar Kale, and their children Himanshu and Shantanu, who provided a home away from home. Sincere thanks are also due to Barbara Ryder, her husband Jonathan, and their children Beth and Andrew for being the greatest of friends. Barbara helped me learn a lot by letting me be her teaching assistant for various courses. The conversations that I had with her have always been enlightening. I am greatly indebted to Dawn Cohen for all her help, especially during the initial adjustment period after my advent to the US. I have benefitted greatly from her unique perspective of life. Thanks are due to Mukesh Dalal for all his help

during that period as well. I would also like to thank Claire Todd for all her support throughout my graduate studies. Very special thanks are due to Valentine Rolfe for being the motherly figure that she has been.

I would like to thank my fellow graduate students for providing me a very conducive environment to pursue my studies. I have shared offices with Jaikumar Radhakrishnan, Jim Llewellyn, Rakesh Sinha, Babu Ramakrishnan and Doug Katzman. They have always been willing to share their ideas with me. I thank Shiva Chaudhuri, Haripriyan Hampapuram and Ileana Streinu for explaining their work to me and teaching me many new things. I have greatly enjoyed my conversations with Mark Kayll, Vasilis Capoyleas, Kumar Vadaparty and Sanjeev Saluja.

I always enjoyed my interaction with the undergraduate students, both as an instructor and a teaching assistant. I would like to thank them all for their patience and understanding, as well as for all their goodwill and praise.

Life at Rutgers would not have been the same without the company of my friends. I would like to thank Jaikumar Radhakrishnan, Dawn Cohen, Barbara DiEugenio, Kumar Vadaparty, Balakrishna Prasad, Sriram, Jana, Jayram, Manoj, Vipul, Sriram Kannan, Rao, Kakka and Shiyu Zhou, to name a few. Special thanks are due to Krishnan Kumaran, S. Vishwanathan and Pritesh Shah for providing great companionship. Arup Acharya helped me get my reality breaks by being a regular squash partner. I am very grateful to Sreekanth Bollam for being a great friend, and for introducing me to the joys of ballroom dancing and weightlifting.

I would like to thank the Computer Science Department, the Graduate School and DIMACS for providing financial support during my studies. I was also supported in part by NSF grant CCR-9204874.

Finally, I would like to express my deepest gratitude to the most important people in my life: my mother, my father and my brother. Without their constant support and inspiration, this would have been but just a dream. This document is dedicated to them.

# Table of Contents

# Chapter 1

# Introduction

Complexity theory aims to classify problems based on the amount of resources needed to solve them. Given a particular problem, there are many ways to talk about its complexity. Firstly, one could show that given a certain amount of resources, the problem can be solved using no more than that amount. This provides what is called an upper bound on its complexity. One could also show that a certain amount of resources is not enough to solve the problem; that provides a lower bound on its complexity. Finally, if there are two or more resources involved, one could talk about the various trade-offs between them. Giving an upper bound requires finding efficient solutions and improving the efficiency of known solutions. Providing a lower bound requires a study of factors inherent to the problem that contribute to the difficulty of solving it.

In complexity theory, problems are usually modeled as *language recognition problems*. A language $L$ is simply a subset of $\Sigma^*$, the set of all finite strings that can be generated from the underlying alphabet $\Sigma$. Since any alphabet can be coded using strings of 0's and 1's, it suffices to consider languages that are subsets of $\{0,1\}^*$. For a language $L$, the associated problem is to determine whether a given string $x$ is in $L$ or not. This may be thought of as computing the characteristic function of the language, $\chi_L : \{0,1\}^* \rightarrow \{0,1\}$, such that for every $x \in \{0,1\}^*$, $\chi_L(x) = 1$ iff $x \in L$. Languages are classified based on the complexity of computing their characteristic function.

In order to talk about the complexity of languages, we must first make precise the notion of a computing device. Historically, Turing machines have served the role of a model computing device. The two most important resources to be considered in this model are *time* and *space*. When computing a function $\chi_L$ with a Turing machine, we consider the time and space needed in order to do the computation. One problem here

is that the computation might take different amounts of time and space on different inputs. We circumvent this problem by expressing the time and space requirements as functions of the length of the input. The class of all functions (languages) that can be computed by machines whose running times are bounded by a polynomial in the input length is denoted by P. Similarly, the class of all languages that can be computed by machines whose space complexities are bounded by the logarithm of a polynomial in the input length is denoted by DLOG.

## 1.1 The Circuit Model

Inspite of its simplicity, the Turing machine model does not admit easy combinatorial and algebraic analysis. Therefore, other models, such as the Boolean Circuit model have been considered. A Boolean circuit on $n$ variables computes a function from $\{0,1\}^n$ to $\{0,1\}$. Since a circuit has a fixed number of inputs, the task of computing the characteristic function $\chi$ of a language using circuits must be thought of as computing an infinite sequence of functions $\{\chi_n\}$ where $\chi_n : \{0,1\}^n \rightarrow \{0,1\}$ is the restriction of $\chi$ to inputs of length $n$. If $C_n$ computes $\chi_n$ for every $n$, then $\{C_n\}$ is a family of circuits that computes $\chi$. The most important resources considered in this model are *size* and *depth*. In order to evaluate the efficiency of a circuit family $\{C_n\}$, we consider the size and depth of $C_n$ as functions of $n$.

Though different in many ways, the Turing machine and the circuit model are related. For example, a language is in P if and only if it has *uniform*[1] polynomial size circuits. Likewise, a language is in P/poly[2] if and only if it has nonuniform polynomial size circuits.

In their most general form, circuits are very difficult to deal with and at present our knowledge of their functioning is fairly limited. In order to understand the way circuits function, researchers have often considered restricted versions of the model. It

[1] Informally, this means that the circuit families are easy to construct. The notion of uniform circuits will be explained in detail in Section 1.3.

[2] A language $L$ is in P/poly if there is a polynomially bounded function $h : \mathrm{N} \rightarrow \{0,1\}^*$ and a language $A \in \mathrm{P}$ such that for every $x \in \{0,1\}^*$, $x \in L \iff \langle x, h(|x|)\rangle \in A$.

is hoped that the insights thus gained will eventually enable us to deal with the model in full generality. Typically, the restrictions considered are constant depth circuits, monotone circuits (circuits without NOT gates) and formulas. In this thesis, we shall be concerned mainly with constant depth circuits.

## 1.2 Constant Depth Circuits

The classes of languages that are accepted by constant depth circuit families have been studied intensely in the last decade or so. One such class is $AC^0$, the class of languages accepted by constant depth circuits of polynomial size that consist of NOT gates and unbounded fan-in AND and OR gates. It is easy to see that addition can be done using a depth three $AC^0$ circuit family. $AC^0$ circuit families are also known for many regular and context free languages. However, the lower bounds for $AC^0$ suggest that it is a very tiny complexity class. Ajtai [Ajt83] and Furst, Saxe and Sipser [FSS84], in their quest to separate the levels of the polynomial hierarchy by oracles, proved the first lower bounds for $AC^0$. They showed that constant depth circuits that use NOT gates, and AND and OR gates of unbounded fan-in need superpolynomial size for computing the PARITY ($MOD_2$) function. Their techniques were later refined by Yao [Yao85], Håstad [Hås87] and Aspnes, Beigel, Furst and Rudich [ABFR91] who improved the lower bound to exponential. Since PARITY is reducible to other functions such as MAJORITY, Multiplication and Transitive Closure [FSS84], these functions are outside $AC^0$ as well. These lower bounds prompted people to look at a slightly more powerful model, namely constant depth, polynomial size circuits consisting of NOT gates, and unbounded fan-in AND and OR gates with the addition of unbounded fan-in PARITY gates. However, Razborov [Raz87] showed that these circuits could not compute the MAJORITY function. Smolensky [Smo87] extended Razborov's method to show that constant depth, polynomial size circuits with NOT gates, and unbounded fan-in AND, OR and $MOD_p$ gates cannot compute the $MOD_q$ function if $p$ and $q$ are distinct primes. This also implies that no constant depth, polynomial size circuit family containing AND, OR, NOT and MOD gates for a single prime modulus can compute the MAJORITY function. This led to further enhancement of the model, the circuit

families could now have different MOD gates, for an arbitrary but fixed set of moduli. The class of languages computed by these circuit families is called ACC. ACC is an acronym for "$AC^0$ with counters", where the various MOD gates are the counters. This class was first mentioned implicitly by Barrington [Bar86] (see also [Bar89]), who was studying the classes of languages recognized by group programs over different group structures. Barrington and Thérien [BT88] showed that ACC is equal to the class of languages recognized by polynomial length programs over solvable monoids.

Proving lower bounds for ACC has not been easy at all; it is consistent with our knowledge that an ACC circuit family can compute the SATISFIABILITY problem. However, Barrington has conjectured that ACC $\subsetneq$ $NC^1$, where $NC^1$ is the class of languages accepted by polynomial size, $O(\log n)$ depth circuits with NOT gates and *bounded* fan-in AND and OR gates.

Another class of constant depth circuits that has been studied is $TC^0$, the class of languages accepted by constant depth, polynomial size circuits consisting of *threshold* gates. A threshold gate computes a threshold function; in general, the $k^{\text{th}}$ threshold function $T_k^n$ is a boolean function on $n$ variables that takes value 1 precisely when there are at least $k$ 1's in the input. Note that the MAJORITY function is the same as $T^n_{\lceil n/2 \rceil}$ and the generalized OR function is $T_1^n$. Threshold circuits are probably quite powerful; it is not known if there is a language in NP that cannot be recognized by depth three, polynomial size threshold circuits. However, Hajnal, Maass et al [HMP+87] have shown that for threshold circuits of polynomial size, depth three is more powerful than depth two. Håstad and Goldmann [HG90] have proved that computing the "generalized inner product function" (considered in [BNS89]) with depth three threshold circuits requires exponential size if the bottom level fan-in is restricted to be $< \frac{1}{2}\log n$. As far as the relationship between $TC^0$ and $NC^1$ is concerned, we have competing conjectures. Immerman and Landau [IL89] have conjectured that $TC^0 = NC^1$, while Yao [Yao89] talks about the possibility of $TC^0 \neq NC^1$.

Some very interesting upper bound results are also known for the constant depth circuit classes. It is well known that the polynomial size, depth three circuits for addition can be used to obtain polynomial size, $O(\log n)$ depth circuits for computing $T_k^n$

for every $k$. This implies that $\mathrm{TC}^0 \subseteq \mathrm{NC}^1$. In fact, monotone circuits of polynomial size and $O(\log n)$ depth for computing $T_k^n$ are implied by the sorting network due to Ajtai, Komlós and Szemerédi [AKS83]. Using the intuition of Toda's result [Tod91] and some known connections between the polynomial hierarchy and constant depth circuits [FSS84], Allender [All89a] showed that every language in $\mathrm{AC}^0$ can be recognized by depth three threshold circuits of size $2^{(\log n)^{O(1)}}$. This was improved by Yao [Yao90] who showed that every language in ACC can be computed by probabilistic depth two circuits of size $2^{(\log n)^{O(1)}}$ with a symmetric[3] gate at the top and AND gates of $(\log n)^{O(1)}$ fan-in at the second level. Beigel and Tarui [BT91] improved Yao's result by showing deterministic (instead of probabilistic) circuits of the above kind for ACC. These results imply that all of ACC can be computed by depth three threshold circuits of size $2^{(\log n)^{O(1)}}$. Green, Köbler and Torán [GKT92] have improved upon Beigel and Tarui's [BT91] result by showing that the top gate in the depth two circuit can be a fixed symmetric gate, which they call a MidBit gate.

### 1.3 Uniform Circuits

So far we have talked about circuits in their most general form; this is usually referred to as the *nonuniform* version. Nonuniform circuits are quite powerful in the sense that the nonuniform circuit complexity classes contain problems that are not computable at all in the ordinary sense. For example, every unary language (that includes nonrecursive languages) is in nonuniform $\mathrm{AC}^0$. Therefore we need to restrict the model in some way so as to keep it realistic. To compute with a circuit family, we must be able to construct the circuit for each input size. We may informally define a *uniform* circuit family as one in which the behavior on all inputs, of any size, is specified by a single, finite string. This finite string may be the description of a Turing machine, which on input $1^n$ produces the circuit $C_n$. This gives us a weak uniformity condition that can be strengthened by placing resource bounds on the Turing machine.

The notion of uniform circuit complexity was suggested by Borodin [Bor77]. Ruzzo

[3] A symmetric function is a Boolean function whose output depends only on the number of 1's in the input. A symmetric gate computes a symmetric function.

[Ruz81] argued that uniform circuit complexity is a reasonable model of parallel complexity. He also investigated various definitions of uniform circuit complexity and showed that it is fairly insensitive to the choice of definition. A circuit family $\{C_n\}$ is logspace-uniform if the circuit $C_n$ can be constructed by a Turing machine on input $1^n$ using $O(\log n)$ space. It is P-uniform if $C_n$ can be constructed from input $1^n$ in time polynomial in $n$. For uniform circuits of polynomial size, these two notions are equivalent and the class of languages recognized by these circuit families is P.

Since the depth of a circuit family is a measure of the parallel time complexity, people have looked at circuit families with restricted depth. For $k \geq 1$, the class $\mathrm{NC}^k$ consistes of languages recognized by logspace-uniform circuit familes of polynomial size and depth $O((\log n)^k)$ where all the gates have bounded fan-in. If we allow unbounded fan-in we get the class $\mathrm{AC}^k$. For all $k \geq 0$, $\mathrm{AC}^k \subseteq \mathrm{NC}^{k+1} \subseteq \mathrm{AC}^{k+1}$. The class NC is the union of the classes $\mathrm{NC}^k$ over all $k \geq 1$. Many natural problems have been shown to be in NC; in particular, it is known that nondeterministic logspace (NLOG) is in $\mathrm{NC}^2$ [Sud75, Ruz80]. Pippenger [Pip79] and Ruzzo [Ruz81] have given alternate characterizations of NC. Another related class that has been considered is the P-uniform version of NC called PUNC. Allender [All89b] studied this class and provided alternate characterizations in terms of alternating Turing machines and other parallel machines. He provides evidence that NC does not adequately model the notion of "feasible parallelism" and argues that PUNC is a better candidate.

Logspace-uniformity is a fairly robust notion of uniformity when we are dealing with classes like NC. However, since $\mathrm{NC}^1 \subseteq \mathrm{DLOG} \subseteq \mathrm{NC}^2$, using logspace-uniformity for $\mathrm{NC}^1$ and its subclasses like $\mathrm{AC}^0$ and ACC is somewhat unsatisfying. Using logspace-uniformity for these classes allows the circuit constructor to have more power than the circuits themselves possess. This has the annoying consequence of making the complexity class fairly sensitive to the notion of uniformity that is used to define it. Therefore, a variety of uniformity conditions suitable for studying $\mathrm{AC}^0$, ACC, $\mathrm{TC}^0$ and $\mathrm{NC}^1$ have been considered.

Ruzzo [Ruz81] came up with the notion of $\mathrm{NC}^1$-uniformity (which he calls $U_{E^*}$-uniformity; see also Cook [Coo85]). He showed that $\mathrm{NC}^1$-uniform $\mathrm{NC}^1$ is equal to

alternating logarithmic time. This notion of uniformity is different from logspace-uniformity and P-uniformity in the sense that here the "constructor" is not powerful enough to actually output the circuit description, but it can answer certain kinds of questions about the circuit in alternating logarithmic time. This is a reasonable notion of uniformity for classes $NC^1$ and above, but to study subclasses of $NC^1$, still more restrictive notions are needed. It should be noted that a class like P-uniform $NC^1$ is still important because it represents problems for which a very fast circuit can be constructed sequentially in a reasonable amount of time. Division and some related problems are known to have P-uniform $NC^1$ circuits by the results of Beame, Cook and Hoover [BCH84]. It is still not known if Division has logspace-uniform $NC^1$ circuits.

Barrington, Immerman and Straubing [BIS90] investigated a variety of notions of uniformity for studying subclasses of $NC^1$. One of the important notions that they studied is based on deterministic logarithmic time and the alternating logarithmic time hierarchy of Sipser [Sip83]. This notion is also used by Buss [Bus87] in his proof that the Boolean formula value problem is in alternating logarithmic time. Ruzzo introduced a similar notion of uniformity in [Ruz81]. He defines the *direct connection language* of a circuit family and the notion of uniformity requires that this language be decidable by an alternating Turing machine running in logarithmic time. The condition in [BIS90] is more restrictive and it requires that the direct connection language be decidable by a *deterministic* machine running in logarithmic time. We shall hereafter refer to this notion as dlogtime-uniformity. They show that dlogtime-uniform $NC^1$ is in fact equal to $NC^1$-uniform $NC^1$. They also give persuasive arguments (see also [BCGR92]) to show that dlogtime-uniformity is the right notion to study $AC^0$ and ACC. In particular, dlogtime-uniform $AC^0$ is known to have many different characterizations demonstrating its robustness.

**Theorem 1.1** [BIS90] The following classes of sets are equal:

- The class of sets accepted by dlogtime-uniform circuits with NOT gates, AND and OR gates of unbounded fan-in, constant depth and polynomial size.

- The class of sets accepted in $O(1)$ time on a CRAM[4] with polynomially many processors.

- The class of sets accepted by alternating Turing machines in $O(\log n)$ time and $O(1)$ alternations (i.e., the logarithmic time hierarchy (also referred to as LH) defined by Sipser in [Sip83]).

- The class of sets definable in first-order logic with additional predicates $<$ (linear order) and $BIT$ (where $BIT(i, j)$ means that bit $i$ of the binary representation of $j$ is 1). This class is sometimes denoted FO $+ < + BIT$.

In Chapter 7, we provide another characterization of dlogtime-uniform $AC^0$ in terms of the log-bounded rudimentary reductions defined by Jones [Jon75]. Additional characterizations of $AC^0$ may be found in [Clo90].

The lower bound results for $AC^0$ mentioned in the Section 1.2 apply to nonuniform $AC^0$ and hence to uniform $AC^0$ as well. Allender and Hertrampf [AH90] showed that the result in [All89a] also holds in the uniform setting. In Chapter 3 we show that the simulation of Beigel and Tarui [BT91] holds in the uniform setting as well. We define a notion of uniformity to deal with circuit families of subexponential size. A similar notion has been defined by Venkateswaran in [Ven92] where he gives a uniform circuit characterization of NP and other classes. In [Bar92], Barrington shows that the notion of uniformity introduced for constant depth circuits of polynomial size in [BIS90] can be extended to quasipolynomial ($2^{(\log n)^{O(1)}}$) size as well. This extended notion of uniformity coincides with the one we use. In the same paper, Barrington provides a different proof that shows that the simulation of Beigel and Tarui [BT91] holds for uniform circuits. He shows that the result of Green, Köbler and Torán [GKT92] holds in the uniform setting as well.

In this thesis we deal exclusively with uniform circuits. We use different notions of uniformity, but they are all derived from the dlogtime-uniformity discussed above. The

[4]A CRAM (Concurrent Random Access Machine) is essentially a concurrent read, concurrent write parallel random access machine (CRCW PRAM) that has an extra instruction called *Shift* in its instruction set. For more details, see [Imm89].

notions will be made precise in Chapter 2. We are interested in uniform circuits for two reasons. Firstly, some of the results that we prove seem to work only for uniform circuits. In Chapter 3 we show that uniform ACC type circuits of subexponential size cannot compute the permanent (PERM) function. Note that when we say that a circuit family computes a function like PERM we mean that the circuit family recognizes the language $\{\langle M, j, b\rangle$ : the $j^{\text{th}}$ bit of the permanent of $M$ is $b\}$. We also show that there are languages in PP and $C_=P$ that cannot be computed by uniform ACC type circuits of subsubexponential size. Our lower bound results use the techniques developed by Beigel and Tarui [BT91], Yao [Yao90] and Toda [Tod91]. As a corollary, we also get the important result that uniform ACC is properly contained in PP (as well as $C_=P$). This seems to be one of the very few instances[5] where lower bounds are known for the uniform circuit complexity of certain languages or functions, but where nothing is known about the nonuniform circuit complexity. The uniformity condition is very critical in our results; it is still unknown if PP = logspace-uniform ACC. Although logspace-uniform ACC is trivially seen to be properly contained in PSPACE, it is not known if P-uniform ACC = PSPACE. In fact, it is even unknown if there is any set in $\text{Ntime}(2^{n^{O(1)}})$ that is not accepted by a nonuniform ACC circuit family.

The other reason that we consider uniform circuits is that some of the properties of circuit complexity classes that we study make sense only with respect to uniform circuits. In Chapter 4 we talk about sets that are immune to $AC^0$ and ACC. An infinite set $L$ is *immune* to a complexity class $\mathcal{C}$ if neither $L$ nor any of its infinite subsets are in $\mathcal{C}$. Every infinite set trivially has a sparse infinite subset in nonuniform $AC^0$. Therefore, in order to discuss immunity with respect to a circuit complexity class, we must consider uniform circuits.

---

[5] In fact, the only other instance that we are aware of is that it is not known if $\text{Dtime}(2^{n^{O(1)}})$ contains sets that are not in P/poly (the class of languages accepted by nonuniform circuit families of polynomial size), whereas it does contain sets that are not in P (which is the class of languages accepted by uniform circuit families of polynomial size).

## 1.4 Complex Sets, Immunity and Strong Separation

What does it mean to show that a set is complex? As a first attempt, one might show that the set does not lie in some "easy" complexity class such as Dtime($n^3$). However, if a set $L$ is not in Dtime($n^3$), it means merely that for each machine $M$ accepting $L$, $M$ must run for more than $n^3$ steps on infinitely many inputs – although this set of inputs could be *extremely* sparse – and it might even be the case that $M$ runs in linear time on an overwhelming majority of inputs. In some settings, such a set $L$ would not be considered sufficiently "complex," and thus two stronger notions of complexity have often been considered: randomness (or Church-randomness) and almost-everywhere complexity.

A set $L$ is said to be *Church-random* with respect to a class $\mathcal{C}$ if for each set $A \in \mathcal{C}$ the probability that $A$ and $L$ differ on a random string of length $n$ approaches $\frac{1}{2}$. (That is, the sequence $|\{x \in \Sigma^n : x \in A \Delta L\}|/2^n$ approaches $\frac{1}{2}$.) Very tight Church-randomness hierarchies are known; for all functions $t$ and $T$ such that Dtime($t(n)$) is known to be properly contained in Dtime($T(n)$), it is known that there is a set in Dtime($T(n)$) that is Church-random with respect to Dtime($t(n)$) [Wil83]. Also, it is known that the PARITY language is Church-random with respect to $\mathrm{AC}^0$ [Cai89, Bab87, Hås87].

However, even languages that are Church-random may have infinitely many "easy" special-case inputs. For example, any string of the form $0^n$ is trivially not in the PARITY language, and any string of the form $0^n1$ is trivially in the language. A language is *almost-everywhere complex* if there are no infinite classes of "special-case" inputs of this form. More formally, a set $L$ is said to be almost-everywhere complex with respect to Dtime($T(n)$) if any machine recognizing $L$ must run for more than $T(|x|)$ steps on *all* large inputs $x$. As with Church-randomness, there are very tight hierarchies for almost-everywhere complexity [GHS91]. We are interested in the study of almost-everywhere complexity because, unlike the case of Church-randomness, little is known about sets that are almost-everywhere complex with respect to circuit complexity classes like $\mathrm{AC}^0$ and ACC. Indeed, since the size and depth of a circuit is the same on all inputs of a given length, it is not at all clear what it should mean for a set to be

almost-everywhere complex with respect to a circuit complexity class. For guidance, we turn to the better-understood notions of time and space complexity.

Almost-everywhere complexity has been studied in a variety of settings dealing with time and space complexity [BS85, GHS91, GK90, ABHH90], and in all instances it has been shown to be intimately connected with immunity. Recall (from Section 1.3) that an infinite set $L$ is immune to a class $\mathcal{C}$ if $L$ has no infinite subset in $\mathcal{C}$. For example, in [BS85] it is shown that for any time-constructible function $T$, a set $L$ is almost-everywhere complex with respect to Dtime($T(n)$) if and only if both $L$ and its complement are immune to Dtime($T(n)$). Thus, a study of sets that are immune to a complexity class is necessary in order to study the notion of almost-everywhere complex sets.

Because of these and other considerations, immunity has often been studied in complexity theory. (For example, see [Lis90] and the papers cited there.) In the literature, a class $\mathcal{D}$ is said to be *strongly separated from* a class $\mathcal{C}$ if there is a set in $\mathcal{D}$ that is immune to $\mathcal{C}$. In this thesis we study the immunity properties of circuit complexity classes such as $\mathrm{AC}^0$ and ACC. For the rest of this discussion, we use $\mathrm{AC}^0$ (ACC) to denote dlogtime-uniform $\mathrm{AC}^0$ (ACC). Recall that a circuit family is dlogtime-uniform if its direct connection language can be decided by a deterministic Turing machine that runs in logarithmic time. In Chapter 4 we show that $\mathrm{P}^{\mathrm{PP}}$ is strongly separated from ACC and hence from $\mathrm{AC}^0$. In Chapter 5 we investigate whether NP is strongly separated from $\mathrm{AC}^0$. Surprisingly, we find that any answer to the above question leads to new techniques for answering some interesting questions regarding the relationship between deterministic and nondeterministic time classes.

Let E = Dtime($2^{O(n)}$), NE = Ntime($2^{O(n)}$) and let RUD denote the class of Rudimentary sets. RUD is the linear time analog of the polynomial hierarchy. It was defined by Smullyan [Smu61] and has been studied extensively (see, e.g., [Wra78, Lip78, Boo78, PD80, Vol83]). One result of [Wra78] shows that the rudimentary sets can also be characterized in terms of linear-time nondeterministic oracle Turing machines, in analogy to the usual definition of the polynomial hierarchy. It is not known if there is a set in E that is immune to RUD. The following proposition shows that this question is related

to the question of strong separation between P and $AC^0$.

**Proposition 1.2** If there is a set in E that is immune to the class of Rudimentary sets then there is a set in P that is immune to $AC^0$.

The proof of the above proposition uses the relationship between RUD and $AC^0$, explained in the following proposition (proved in Chapter 5 as Proposition 5.3):

**Proposition 1.3** $L \in \text{RUD} \iff un(L) \in AC^0$, where $un(L) = \{0^n : n \in L\}$.

Proposition 1.2 implies that if NP is not strongly separated from $AC^0$, then P $\neq$ NP (for a proof, see Corollary 5.4).

Let E-solvable informally (formal definitions are given in Chapter 2) denote the notion "solvable in deterministic time $2^{O(n)}$." The question "Is every NE-predicate E-solvable?" is the natural exponential time analog of the so called *witness finding* question regarding NP and P. It was studied initially by [AW88]. Even though the P = NP question is usually formulated as a question about the complexity of recognizing languages, it is equivalent to the question of witness finding (e.g., finding a satisfying assignment, instead of merely reporting that a satisfying assignment exists). However, for exponential time, the corresponding questions are not known to be equivalent. In [IT89] it is shown that there is an oracle relative to which E = NE but not all NE-predicates are E-solvable (solvable in exponential time). Thus at least in some relativized worlds, assuming that all NE-predicates are E-solvable is strictly stronger than merely assuming E = NE.

The question of whether or not all NE-predicates are E-solvable is related to questions about $AC^0$. In particular, we show the following:

**Proposition 1.4** If E = RUD and all NE-predicates are E-solvable, then no set in NP is immune to $AC^0$.

The proof of the above proposition makes use of some connections between the complexity of NE-predicates and a version of time-bounded Kolmogorov complexity that are explained in Chapter 5. Proposition 1.4 implies that if NP is strongly separated

from $AC^0$, then either E $\neq$ RUD or there are NE-predicates that are not E-solvable. None of these conclusions is known to be true. Propositions 1.2 and 1.4 have been proved as Theorems 5.1 and 5.2 respectively.

## 1.5 Relativization

Oracle constructions have long been used to provide evidence that certain questions in complexity theory cannot be resolved using the usual techniques of simulation and diagonalization. One of the earliest and ground breaking results was proved by Baker, Gill and Solovay [BGS75] who showed the existence of oracles $A$ and $B$ such that $P^A = NP^A$ and $P^B \neq NP^B$. This shows that any technique that relativizes will not be able to resolve the P vs. NP question. Along similar lines, we are able to prove the existence of oracles relative to which the hypotheses of Proposition 1.2 and 1.4 hold. Thus the usual techniques will not be able to answer the question of whether or not NP is strongly separated from $AC^0$.

Though our relativization results deal with $AC^0$, we must emphasize that we do not deal with relativized notions of $AC^0$. Our results only use the usual notion of relativized deterministic and nondeterministic time. Indeed, it is not clear that it makes much sense to talk about relativized $AC^0$; for example, if $L$ is any set that is complete for P under $AC^0$ reductions, then $P^L = AC^{0L}$, although clearly P $\neq AC^0$.

Of course, it is now recognized that the mere existence of an oracle, relative to which some condition $X$ fails to hold, does not preclude the existence of an easy proof that $X$ does hold in the unrelativized case. We know that IP (the set of languages with efficient Interactive Proof Systems) contains $P^{PP}$ ([FKLN90]) and is in fact equal to PSPACE ([Sha90]). Contrast this with a previous result of Fortnow and Sipser [FS88] that showed the existence of an oracle relative to which co-NP $\not\subset$ IP. With that in mind, we conjecture that the hypothesis to Proposition 1.4 is false. If E = RUD, it follows that alternating linear time is equal to $\Sigma_k\text{Time}(n)$ for some $k$, which puts very low limits on the power of deterministic exponential time. Thus it may seem contradictory to assume simultaneously that deterministic time $2^{O(n)}$ is powerful enough to solve every NE-predicate. Note also that if E = RUD, then the polynomial hierarchy

collapses. However, the non-relativizing proof techniques developed in [FKLN90, Sha90, BFL91] have not been shown to be relevant for questions concerning deterministic and nondeterministic time classes; new non-relativizing proof techniques are still needed to answer these questions. We think that our oracle results are interesting because they point to certain limitations of relativizing proof techniques when it comes to proving the existence of sets that are immune to $AC^0$. Research that has focused on proving lower bounds for $AC^0$ has been very successful and we hope that our oracle results will point to ways of improving the lower bound techniques so as to say something about the relationship between NP and $AC^0$ with regard to strong separation. Any progress made here would result in new techniques for attacking the Dtime vs. Ntime question.

## 1.6 Rudimentary Reductions

Logspace reductions were defined and studied independently by Stockmeyer and Meyer [SM73] and Jones [Jon75] as a tool for studying the relative complexity of problems in P. Jones [Jon75] also introduced a restricted version of logspace-reducibility, called log-bounded rudimentary reducibility. His motivation for introducing this restriction came from (1) the desire to have a tool for talking about the structure of very small complexity classes such as Dspace($\log n$), and (2) an interest in generalizing the notion of "rudimentary relations," which at that time was the object of a considerable amount of attention [Smu61, Wra78, Sal73, PD80, Wil79, PHW85]. In [Jon75], Jones goes on to show that a number of problems are complete for various complexity classes under log-bounded rudimentary reducibility. For a function $S$, he defines $\mathrm{RUD}_S$, the class of $S(\cdot)$-bounded rudimentary predicates. Informally, consider a class of predicates that consists of primitive predicates that allow us to look at a particular bit of a string, express string concatenation, and express substitution of variables by constant strings. The closure of this class under logical connectives $\wedge, \vee, \neg$, and universal and existential quantification bounded by the function $S$ gives us the class $\mathrm{RUD}_S$. The function $S$ should be thought of as a space bound. The log-bounded rudimentary reductions are defined using the class $\mathrm{RUD}_{\log}$, which is the class of rudimentary predicates we get when the space bound $S$ is logarithmic. We observe that the class $\mathrm{RUD}_S$ of predicates can also be viewed as a

class of languages and in this formalism, we are able to show that $\mathrm{RUD}_{\log} = \mathrm{AC}^0$. The proof is presented in Chapter 7 and uses some of the alternate characterizations of $\mathrm{AC}^0$ given in [BIS90]. This provides further evidence that the notion of dlogtime-uniformity proposed by [BIS90] for studying small complexity classes such as $\mathrm{AC}^0$ is indeed the "right" notion.

## 1.7 Outline

The rest of the thesis is organized as follows. The notation and preliminary definitions are presented in Chapter 2. In Chapter 3 we present the results from [AG92], namely the lower bounds for PERM and PP. In Chapter 4 we show that $\mathrm{P}^{\mathrm{PP}}$ contains sets that are immune to uniform ACC (and hence immune to uniform $\mathrm{AC}^0$ as well). This follows from the results proved in Chapter 3. We also give an independent proof of the fact that $\mathrm{P}^{\mathrm{PP}}$ is strongly separated from uniform $\mathrm{AC}^0$. In Chapter 5 we show that the problem of finding a strong separation between NP and uniform $\mathrm{AC}^0$ is linked to some long-standing open questions in complexity theory that deal with the relationship between deterministic and nondeterministic exponential time classes. In Chapter 6 we show that proof techniques that relativize will not be able to prove anything about the existence or nonexistence of a strong separation between NP and $\mathrm{AC}^0$. The results in Chapters 4, 5 and 6 have appeared in [AG91a] and [AG93]. The characterization of uniform $\mathrm{AC}^0$ in terms of the log-bounded rudimentary reductions is presented in Chapter 7. This result has appeared in [AG91b]. Conclusion and open problems are discussed in Chapter 8.

# Chapter 2

# Preliminaries

## 2.1 Turing Machines

The reader is assumed to be familiar with deterministic and nondeterministic Turing machines, the basic notions of time and space complexity and the various notions of reducibilities like polynomial time many-one ($\leq_m^p$), polynomial time Turing ($\leq_T^p$) and logspace many-one ($\leq_m^{\log}$) etc. A good introduction to this material can be found in [HU79]. We also assume that the reader is familiar with the concept of Turing machine computation relative to an oracle. We use the following abbreviations:

P = Dtime($n^{O(1)}$)

NP = Ntime($n^{O(1)}$)

E = Dtime($2^{O(n)}$)

NE = Ntime($2^{O(n)}$)

DLOG = Dspace($\log n$)

NLOG = Nspace($\log n$)

PSPACE = Dspace($n^{O(1)}$)

We also assume familiarity with the hierarchy theorems for deterministic time [HS65] and nondeterministic time [FMS78]. We use the following reformulation of the almost-everywhere hierarchy theorem for deterministic time classes proved by Geske, Huynh and Seiferas [GHS91].

**Theorem** [GHS91] For $k \geq 2$, if $T(n)$ is the exact running time of a $k$-tape Turing machine, and if $t(n)$ is such that $t(n)\log t(n) = o(T(n))$, then there is a $\{0,1\}$-valued function computable by a $k$-tape Turing machine in time $O(T(n))$, such that every multitape Turing machine that computes this function requires time exceeding $t(n)$ almost everywhere.

The nondeterministic Turing machine model plays a very important role in this thesis. When dealing with this model, we are not only concerned with the *existence* of accepting paths on inputs, we are also concerned with the *number* of accepting paths. With that in mind, we present the following definitions:

**Definition 2.1** For a nondeterministic machine $M$, let $\#\mathrm{acc}_M : \Sigma^* \to \mathbf{N}$ be the function defined by $\#\mathrm{acc}_M(x)$ = number of accepting paths of $M$ on input $x$. Then $\#\mathrm{P} = \{\#\mathrm{acc}_M : M \text{ is an NP machine}\}$.

**Definition 2.2** Let $A = [A_{i,j}]$ be an $n \times n$ matrix. Then
$\mathrm{PERM}(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i,\sigma(i)}$
where $S_n$ is the symmetric group on $n$.

It is well known from [Val79] that PERM is complete for #P under $\leq_T^p$ (in fact, $\leq_m^p$) reducibility. The proof shows that given a propositional formula in conjunctive normal form, in polynomial time we can obtain a matrix such that if the value of the permanent of this matrix is known, then we can compute the number of satisfying assignments of the given formula from this value in polynomial time.

**Definition 2.3** A language $L$ is said to be in $\mathrm{PrTime}(t(n))$ if there exists a nondeterministic machine $M$ that runs in time $t(n)$ such that for all $x \in \Sigma^*$,
$x \in L \iff$ more than half of the computation paths of $M$ on input $x$ are accepting.

**Definition 2.4** A language $L$ is said to be in $\mathrm{C_{=}Time}(t(n))$ if there exists a nondeterministic machine $M$ that runs in time $t(n)$ such that for all $x \in \Sigma^*$,
$x \in L \iff$ exactly half of the computation paths of $M$ on input $x$ are accepting.

**Definition 2.5** A language $L$ is said to be in $\mathrm{MOD}_k\mathrm{Time}(t(n))$ if there exists a nondeterministic machine $M$ that runs in time $t(n)$ such that for all $x \in \Sigma^*$,
$x \in L \iff \#\mathrm{acc}_M(x) \not\equiv 0 \pmod{k}$.

In particular, for polynomial running times we get the well known classes PP = $\mathrm{PrTime}(n^{O(1)})$, $\mathrm{C_{=}P} = \mathrm{C_{=}Time}(n^{O(1)})$ and $\mathrm{MOD}_k\mathrm{P} = \mathrm{MOD}_k\mathrm{Time}(n^{O(1)})$. The class $\mathrm{MOD}_2\mathrm{P}$ is usually referred to as PARITY P ($\oplus$P) in literature.

The class $\mathrm{P}^{\mathrm{PP}}$ is simply the class of languages that are $\leq_T^p$ reducible to languages in PP. Toda [Tod91] proved the very important result that the polynomial hierarchy is contained in $\mathrm{P}^{\mathrm{PP}}$.

## 2.2 Alternating Turing Machines

Alternating Turing machines (ATM) are a generalization of nondeterministic Turing machines and the concept of alternation has proved to be very useful in complexity theory, particularly in explicating the difference between time and space complexity. We assume that the reader is familiar with the basic concepts like computation trees, accepting subtrees and time and space bounded ATM's. Results and precise definitions relating to alternation may be found in [CKS81] and [BDG89]. The polynomial hierarchy (PH) is the class of languages accepted by ATM's that run in polynomial time and make a constant number of alternations. An important complexity class that is defined using ATM's is the following:

**Definition 2.6** The class of *rudimentary* sets is the class of languages accepted by alternating Turing machines making $O(1)$ alternations, and running for linear time. We will usually denote this class by $\bigcup_k \Sigma_k\mathrm{Time}(n)$ (or by RUD sometimes).

Some of the results in this thesis use the model of alternating Turing machine used by Ruzzo [Ruz81] in which access to the input is provided via a special tape which we call the *input address tape* onto which an address $i$ may be written in binary, following which in unit time the $i^{\text{th}}$ input symbol is available. This convention is necessary to allow machines with sublinear running times to have access to all of the input. Consult [BIS90, BCGR92] for examples illustrating how these machines compute.

In order to prove some of the results in Chapter 3, we make use of a slightly enhanced version of the ATM. The ATM is also allowed to have MOD states, apart from the conventional (see [CKS81]) existential ($\exists$) and universal ($\forall$) states. The concept of a MOD state is defined as follows:

**Definition 2.7** For a modulus $m$, a $\mathrm{MOD}_m$ configuration (say $\sigma$) is always the root of a subtree of configurations in which all the other configurations (except the root)

are existential. This tree is called the *existential subtree* associated with $\sigma$ and is represented as $T_\sigma$. We say that $\sigma$ is *accepting* if and only if the number of leaves of $T_\sigma$ that are accepting is congruent to 0 modulo $m$. We also use the term MOD-tree at times to refer to an existential tree associated with a MOD configuration.

**Definition 2.8** There is said to be an *alternation* between two configurations $\sigma_1$ and $\sigma_2$ of an ATM if and only if $\sigma_2$ follows from $\sigma_1$ via one step of the ATM and one of the following conditions hold:

- $\sigma_1$ is of type $\exists$ and is not in the MOD-tree of any MOD configuration, and $\sigma_2$ is of type $\forall$ or MOD.
- $\sigma_1$ is of type $\exists$ and is in a MOD-tree, and $\sigma_2$ is of type $\exists$ and is not in any MOD-tree, or $\sigma_2$ is of type $\forall$ or MOD.
- $\sigma_1$ is of type $\forall$ and $\sigma_2$ is of type $\exists$ or MOD.

Let $T$ denote the computation tree of an ATM $M$ on a particular input. The root of the tree is said to have *alternation depth* 1, and a node in the tree labeled by configuration $\sigma_2$ with parent labeled by configuration $\sigma_1$ is defined to have alternation depth one greater than the alternation depth of $\sigma_1$ if there is an alternation between $\sigma_1$ and $\sigma_2$, and the alternation depth of $\sigma_2$ is equal to that of $\sigma_1$ otherwise. The alternation depth of a tree is the maximum alternation depth of all nodes in the tree. The alternation depth of an ATM is the maximum alternation depth of all its alternation trees.

## 2.3 Boolean Circuits

A Boolean circuit on $n$ variables $\{x_1, x_2, \ldots, x_n\}$ is a directed acyclic graph. The nodes of the circuit are called *gates*. Nodes of indegree 0 are labeled either by a variable $x_i$, or by a constant (0 or 1). Those labeled by a variable are called *input* gates and the ones labeled by a constant are called *constant* gates. There is exactly one node with outdegree 0 called the *output* gate. The indegree of a node is normally referred to as its fan-in. All nodes that have nonzero fan-in are labeled by different Boolean functions such as AND, OR, NOT etc. The *level* of a gate in a circuit is the length of the longest

path from an input to that gate (hence the input gates are at level 0). The *size* of a circuit is the number of gates in it and the *depth* of a circuit is the length of the longest path from an input gate to the output gate. If there are $n$ variables in the circuit, then the circuit computes a function from $\{0,1\}^n$ to $\{0,1\}$.

**Definition 2.9** Let $m$ be a positive integer. A $\mathrm{MOD}_m$ gate outputs 1 if the sum of its (binary) inputs is 0 modulo $m$; 0, otherwise. That is,

$$\mathrm{MOD}_m(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } \sum_i x_i \equiv 0 \pmod{m} \\ 0 & \text{otherwise.} \end{cases}$$

**Definition 2.10** A language $L$ is in $\mathrm{AC}^0$ if it is recognized by a family of constant depth, polynomial size circuits consisting of NOT gates, and unbounded fan-in AND and OR gates.

**Definition 2.11** ([MT89, Bar89, BT88]) A language $L$ is in ACC if there exists a positive integer $m$ such that $L$ is recognized by a family of constant depth polynomial size circuits containing NOT gates, and unbounded fan-in AND, OR and $\mathrm{MOD}_m$ gates.

ACC was first defined and studied in [MT89, Bar89, BT88] under the name $\mathrm{ACC}^0$. Yao [Yao90] defines ACC a little differently; he allows a fixed finite set of moduli $S$ instead of a single modulus $m$. However, it is easy to see that a $\mathrm{MOD}_m$ gate can simulate a $\mathrm{MOD}_k$ gate for any $k$ that divides $m$. Letting $m$ to be the least common multiple of the elements in $S$ makes the two definitions equivalent.

We now present the notions of uniformity that we use in this thesis.

**Definition 2.12** Let $\{C_n\}$ be a family of circuits. Following [Ruz81], we define the *direct connection language* $L$ of $\{C_n\}$ as:

$$L = \{\langle n, g_1, g_2\rangle : \ g_1 = g_2 \text{ and } g_1 \text{ is a gate in } C_n \text{ or}$$
$$g_1 \neq g_2 \text{ and } g_2 \text{ is an input to } g_1 \text{ in } C_n\}.$$

Here $g_1$ and $g_2$ are names of gates and $n$ is in binary notation.

**Definition 2.13** A circuit family $\{C_n\}$ is *dlogtime-uniform* if its direct connection language can be recognized in linear time by a Turing machine. The Turing machine that recognizes the direct connection language of $\{C_n\}$ will be referred to as the *uniformity machine* for $\{C_n\}$.

Most of the results in this thesis use the above notion of uniformity. However, for the results in Chapter 3, we use the following notion of uniformity which is a little less restrictive.

**Definition 2.14** A circuit family $\{C_n\}$ is *uniform* if its direct connection language can be recognized in polynomial time by a Turing machine. Note that the time is polynomial with respect to the length of the strings in the language ($|\langle n, g_1, g_2\rangle|$) and not polynomial in $n$.

In order to prove the lower bounds in Chapter 3, we need to deal with circuit families that have subexponential sized circuits. We are interested in two classes of subexponential functions that we call *subexp* and *subsubexp*. Let us call a function $f$ *constructible* if $f(n) = 2^{g(n)}$, where $g(n)$ can be computed from $n$ (in binary) in time polynomial in $g(n)$. Let *subexp* denote the class of all monotonic functions that are bounded above by some constructible function $f$ such that $\forall \epsilon > 0, f(n) = o(2^{n^\epsilon})$. Let *subsubexp* denote the class of all monotonic functions such that for any two functions $f$ and $g$ in this class, the composition of $f$ and $g$ is in *subexp*.

A typical example of a function in *subexp* is $2^{n^{1/\log^* n}}$, and typical examples of functions in *subsubexp* are $n^{\log n}$ and $2^{(\log n)^{\log\log n}}$. It is not hard to prove that if $s$ is in *subexp*, then so is $s^{(\log s)^k}$, for any constant $k$.

**Definition 2.15** Let $\mathrm{ACC}(s(n))$ denote the class of languages accepted by uniform circuit families of constant depth circuits with NOT gates, and unbounded fan-in AND, OR and $\mathrm{MOD}_m$ gates (for some integer $m \geq 2$) of size $s(n)$. Then

$$\mathrm{ACC}(subexp) = \bigcup_{s \in subexp} \mathrm{ACC}(s(n))$$

$$\mathrm{ACC}(subsubexp) = \bigcup_{s \in subsubexp} \mathrm{ACC}(s(n))$$

Note that the notion of uniformity defined in Definition 2.14 is only slightly different from the one in Definition 2.13. Our notion of uniformity can be informally referred to as polylogtime-uniformity. The reason we use this notion is that we are dealing with circuits of (sub)subexponential size and the proofs are much simpler with this uniformity condition. It should be noted that a set has uniform ACC($(sub)subexp$) circuits with respect to our notion of uniformity if and only if it has dlogtime-uniform ACC($(sub)subexp$) circuits. This can be established by "padding" a circuit with many dummy gates.

## 2.4 P-printable Sets, NE-predicates and Time-Bounded Kolmogorov Complexity

We now define the notions of P-printable sets, NE-predicates, and Time-bounded Kolmogorov complexity and show their relationships.

For a language $L$, let $L^{=n} = \{x : |x| = n \wedge x \in L\}$.

**Definition 2.16** A set $L$ is called *P-printable* if the function $1^n \longmapsto L^{=n}$ is computable in polynomial time.

P-printable sets were defined by Hartmanis and Yesha [HY84]; they have been studied in detail by Allender and Rubinstein [AR88]. An important lemma about P-printable sets that will be useful is the following:

**Lemma 2.1** Every infinite set in P has an infinite P-printable subset only if every infinite set in NP has an infinite subset in P-uniform $AC^0$.

**Proof.**
Theorem 7 in [AR88] says that every infinite set in P has an infinite P-printable subset iff every infinite set in NP has an infinite P-printable subset. Note that every P-printable set is in P-uniform $AC^0$. Hence, the lemma follows. ∎

**Definition 2.17** An *NE-predicate* is a binary predicate $R$ such that, for some NE machine $M$, $R(x, y) \iff y$ is an accepting computation of $M$ on input $x$.

**Definition 2.18** An NE-predicate $R$ is *solvable in time* $t(n)$ if there is a function $f$ computable in time $t(n)$ such that, for all $x$, $(\exists y)R(x,y) \iff R(x,f(x))$. Stated another way, $R$ is solvable in time $t(n)$ if there is a routine that, for all $x$, can find a witness for $x$ if one exists. $R$ is said to be *E-solvable* if $R$ is solvable in Dtime($2^{O(n)}$).

**Definition 2.19** An NE-predicate $R$ is said to be *immune to time* $t(n)$ if

1. the set $\{x : (\exists y)R(x,y)\}$ is infinite, and
2. for all $f$ computable in time $t(n)$, the set $\{x : R(x,f(x))\}$ is finite.

Call an NE-predicate *E-immune* if it is immune to Dtime($2^{O(n)}$).

NE-predicates and P-printability are related by the following lemma by Allender and Watanabe [AW88]:

**Lemma 2.2** Every infinite set in P has an infinite P-printable subset $\iff$ No NE-predicate is E-immune.

**Proof.**
($\Rightarrow$) Suppose every infinite set in P has an infinite P-printable subset. Let $R$ be an NE-predicate defined by the NE machine $M$ that accepts an infinite language. Let $L = \{x\#y : R(x,y)\}$ and let $k$ be such that $|x\#y| = 2^{k|x|}$. It is fairly obvious that $L \in$ P. Therefore $L$ has an infinite P-printable subset $L'$. The P-printability of $L'$ guarantees the existence of a polynomial time computable function $f$ such that $f(1^n) = L'^{=n}$. For a string $x$, let $g(x) =$ the lexicographically least string $y$ such that $x\#y \in f(1^{2^{k|x|}})$, if such an element exists; otherwise let $g(x)$ be the empty string. Note that $g$ is computable in exponential time. Since $L'$ is infinite, $g(x)$ is nonempty for infinitely many $x$. Therefore, the set $\{x : R(x,g(x))\}$ is infinite. Hence, $R$ is not E-immune.

($\Leftarrow$) Let $L$ be an infinite language in P. Consider the language $L' = \{n :$ there is a string of length $n$ in $L\}$. $L' \in$ NE because an NE machine (say $M$) can guess a string $w$ of length $n$ on input $n$, and then deterministically verify whether $w \in L$. Suppose $M$ defines the NE-predicate $R$. Since no NE-predicate is E-immune, there is a function $f$

computable in exponential time such that the set $\{x : R(x, f(x))\}$ is infinite. It is easy to see that using $f$ we can generate an infinite subset of $L$ that is P-printable. ∎

The following paragraphs introduce the notion of time-bounded Kolmogorov complexity that we use. The definitions below were introduced in [Lev84, All89c]; more formal definitions and background may be found there and in [All92].

**Definition 2.20** $Kt(x) = \min\{|y| + \log t : M_u(y) = x \text{ in at most } t \text{ steps}\}$ where $M_u$ is a "universal" Turing machine. Let $L \subseteq \{0,1\}^*$. Then $K_L(n) = \min\{Kt(x) : x \in L^{=n}\}$. If $L^{=n} = \phi$ then $K_L(n)$ is undefined.

It turns out that questions about the difficulty of NE-predicates can actually be expressed as questions about the $K_L$ complexity of sets $L$ in P. This is made precise in the following proposition.

**Proposition 2.3**

- Every NE-predicate is E-solvable iff for every set $L$ in P, $K_L(n) = O(\log n)$.
- No NE-predicate is E-immune iff for every set $L$ in P, $K_L(n) \neq \omega(\log n)$.

**Proof.**
Follows from Theorem 6 in [All92] (see also [AW88, Theorem 4]). ∎

The above proposition holds relative to an arbitrary oracle, and this fact is used in the proof of Proposition 6.3.

## 2.5 Rudimentary Reductions

In the following paragraphs, we present the definitions of Jones [Jon75] to define the class of space-bounded rudimentary predicates and reductions.

**Definition 2.21** Let $\Sigma$ be an alphabet. The ternary predicate $\sigma(x, i, a)$ is true iff $a$ is the $i^{\text{th}}$ symbol of $x$ where $x \in \Sigma^*$.

**Definition 2.22** For a function $S : \mathbf{N} \to \mathbf{N}$ such that $S(n) = \Omega(\log n)$, the class of $S(\cdot)$-bounded rudimentary predicates, $\mathrm{RUD}_S$, is the smallest class of predicates such that

1. $\sigma(x, i, a)$ and $\neg\sigma(x, i, a)$ are in $\mathrm{RUD}_S$. Note that $1 \leq |i| \leq \log|x|$.

2. If $c$ is a positive integer then both $Q(x, u, v, w)$ and $\overline{Q}(x, u, v, w)$ are in $\mathrm{RUD}_S$ where

$$Q(x, u, v, w) \iff uv = w \wedge |uvw| \leq c \cdot S(|x|)$$
$$\overline{Q}(x, u, v, w) \iff uv \neq w \wedge |uvw| \leq c \cdot S(|x|)$$

3. If $Q(x, y_1, \ldots, y_m)$ and $R(x, y_1, \ldots, y_m)$ are in $\mathrm{RUD}_S$, then so are $Q(x, y_1, \ldots, y_m) \wedge R(x, y_1, \ldots, y_m)$ and $Q(x, y_1, \ldots, y_m) \vee R(x, y_1, \ldots, y_m)$.

4. If $Q(x, z_1, \ldots, z_m)$ is in $\mathrm{RUD}_S$ and each of $\xi_1, \ldots, \xi_m$ is either a string in $\Sigma^*$ or one of the variables $y_1, \ldots, y_n$, then the predicate $R$ defined by $R(x, y_1, \ldots, y_n) \iff Q(x, \xi_1, \ldots, \xi_m)$ is in $\mathrm{RUD}_S$.

5. If $c$ is a positive integer, and $Q(x, y_1, \ldots, y_{m+1})$ is in $\mathrm{RUD}_S$, then the predicates $R$ and $R'$ defined as follows are also in $\mathrm{RUD}_S$

$$R(x, y_1, \ldots, y_m) \iff (\exists z)[|z| \leq c \cdot S(|x|) \wedge Q(x, y_1, \ldots, y_m, z)]$$
$$R'(x, y_1, \ldots, y_m) \iff (\forall z)[|z| \leq c \cdot S(|x|) \Rightarrow Q(x, y_1, \ldots, y_m, z)]$$

**Definition 2.23** A function $f : \Sigma^* \to \Sigma^*$ is $S(\cdot)$-bounded rudimentary iff the predicate $R$ defined by $R(x, i, a) \iff \sigma(f(x), i, a)$ is in $\mathrm{RUD}_S$.

That is, $f$ is $S(\cdot)$-bounded rudimentary iff the predicate "$a$ is the $i^{\text{th}}$ symbol of $f(x)$" is in $\mathrm{RUD}_S$. The fact that $|i| \leq c \cdot S(|x|)$ (remember that $S(n) = \Omega(\log n)$) for some $c$ and all $x$ implies that $|f(x)| \leq 2^{c \cdot S(|x|)}$.

Note that, as defined above, $\mathrm{RUD}_S$ is a set of *predicates*; however we can just as easily view $\mathrm{RUD}_S$ as a set of *languages*. A language $L$ will be said to be in $\mathrm{RUD}_S$ iff the characteristic function of $L$ is $S(\cdot)$-bounded rudimentary. Alternatively, each predicate $Q \in \mathrm{RUD}_S$ over alphabet $\Sigma$ defines a language $L$ over alphabet $\Sigma \cup \{,\}$ (where "," is any

symbol not in $\Sigma$) defined by $L = \{x, y_1, \ldots, y_m : Q(x, y_1, \ldots, y_m)\}$. These alternative definitions are easily seen to be equivalent. For more formal arguments along these lines, see [Wra78]. It has been proved in [Wra78] that if the space bound $S$ is linear then $\mathrm{RUD}_S$ (viewed as a set of languages) is the same as RUD (the class of rudimentary sets; also denoted as $\bigcup_k \Sigma_k\mathrm{Time}(n)$). We prove a somewhat more general result in Corollary 7.2.

## 2.6 The First Order Framework

In order to present our results about log-bounded rudimentary reductions, we need to use the first order logic framework that has been used to define classes of languages that are subsets of $\{0,1\}^*$. The use of this framework may be traced back to McNaughton [MP71] (see also [Bar90]). Immerman and Fagin are primarily responsible for the application of this tool to complexity theory. We provide a sketch of the relevant information on the first order framework below. For more detailed definitions and background information, see [Imm83, Imm87, Imm89].

In the first order framework, we have a first order language such that every sentence $\phi$ in the language expresses a property of strings. Given a string $x$, either $x$ satisfies $\phi$ (in which case we write $x \models \phi$) or it doesn't. Thus each sentence $\phi$ defines a language, namely $\{x : x \models \phi\}$. The first order language contains variables that range over positions in strings. Even though the variables represent numeric values, the language is also designed to talk about binary representations of these numeric values. It is possible to refer to individual bits of these binary representations. In short, we can describe the framework as follows:

- The logical language has variables $i, j, k, \ldots$ that range over positions in the string under consideration.
- There are constant symbols for 1 and $n$ and there are sentences in the language that define 1 and $n$ respectively to be the first and the last positions in the string.
- The binary predicates $=$ and $<$, have their usual meaning on numbers.

- There is a unary predicate $X(i)$ for accessing the input. The value of $X(i)$ is the $i^{\text{th}}$ bit of the string under consideration.

- There is a binary predicate $BIT(i,j)$ on numbers that holds iff the $i^{\text{th}}$ bit in the binary expansion of $j$ is 1.

- Well-formed formulae are built in the usual way from the constant symbols: 1,$n$ and the relation symbols: $=, \leq, X, BIT$ using logical connectives: $\wedge, \vee, \neg$; variables: $i, j, k, \ldots$ and quantifiers: $\forall, \exists$. A formula with no free variables is called a *sentence*.

For example, the sentence $(\exists i)(\forall j)(\neg(i < j) \vee X(j)) \wedge (\neg(j < i) \vee \neg X(j))$ defines the language $1^*0^*$. Let FO denote the set of all languages expressible in first order logic in this way.

**Theorem 2.4** [BIS90] $\text{AC}^0 = \text{LH} = \text{FO}$.

Two minor complications arise from the statement of Theorem 2.4:

1. Alternating Turing machines can accept input over any alphabet $\Sigma$, whereas FO and $\text{AC}^0$ are classes of languages over the alphabet $\{0,1\}$. Thus we simply assume a binary encoding of any other alphabet. (Alternatively, the FO and circuit frameworks can be modified to allow arbitrary alphabets.)

2. It is clear what it means for a function to be computed by an $\text{AC}^0$ circuit, but it is less clear how a function can be specified in the LH or FO settings. To resolve this difficulty, we use a standard technique that has been used since at least [Jon75]. For a function $f$ with polynomial growth rate[1], let $A_f(c,i,z) \stackrel{\text{def}}{=}$ the $i^{\text{th}}$ symbol of $f(z)$ is $c$. Then we say that $f$ is in LH (or FO) if $A_f$ is in LH (or FO). It has been noticed by many authors (e.g., [BIS90]) that $f$ is computed by $\text{AC}^0$ circuits iff $A_f \in \text{AC}^0$.[2]

[1] That is, there is some polynomial $p$ such that for all $x$, $|f(x)| \leq p(|x|)$.

[2] Again, for equivalence to hold, the output of the circuit must be interpreted in such a way as to allow an "endmarker," or else all functions $f$ in $\text{AC}^0$ must have $|f(x)| = |f(y)|$ if $|x| = |y|$, which is clearly not the case for some functions in LH and FO.

# Chapter 3

# Lower bounds for the Permanent, PP and $C_{=}P$

In this chapter, we provide lower bounds for the classes of languages accepted by uniform circuit families of ACC type circuits of (sub)subexponential size. We show that PERM cannot be computed by an ACC(*subexp*) circuit family. We prove that there are languages in PP and $C_{=}P$ that cannot be recognized by ACC(*subsubexp*) circuit families. We are also able to show that ACC is properly contained in both PP and $C_{=}P$. Our main tool in proving these results is the following result:

**Main Tool** There is a set $Y$ in PP such that $\mathrm{ACC}(subexp) \subseteq \mathrm{Dtime}(n^2)^Y$.

To prove the main tool, we will first use the results of Toda [Tod91], Yao [Yao90] and Beigel and Tarui [BT91] to convert a circuit family in ACC(*subexp*) into an equivalent circuit family of depth two circuits with a symmetric gate at level two, AND gates of small fan-in at level one and the input gates at level zero. However, since we need the resulting circuit family to be uniform as well, we need to show that the above conversion process can be done uniformly. We then show that the language recognized by the new circuit family can be quickly recognized by a deterministic Turing machine that has access to a particular oracle set in PP. Results about PERM then follow from Valiant's [Val79] results about the class #P.

## 3.1 The main results

For the proof of the main tool, we will first show the following.

**Theorem 3.1** Suppose $L$ is accepted by an ACC($subexp$) circuit family $\{C_n\}$ of $s(n)$ sized circuits. Then $L$ is accepted by a uniform, depth two circuit family[1] whose circuits have the following properties:

- Level one consists of a subexponential number of AND gates each having fan-in $(\log s(n))^{O(1)}$. Furthermore, given the name of one of these AND gates, the exact fan-in of this AND gate can be computed deterministically in time $(\log s(n))^{O(1)}$.
- There is a symmetric gate at level two. Furthermore, given the number $m$ of AND gates that evaluate to one, it can be determined deterministically in time $(\log s(n))^{O(1)}$ if the symmetric gate will evaluate to one.

The above theorem is the most important part of the argument and most of the chapter hereafter is devoted to its proof. The proof uses techniques developed by Beigel and Tarui [BT91], Yao [Yao90] and Toda [Tod91]. The reader who is willing to accept the fact that the construction of [BT91] can be carried out uniformly can simply skip Section 3.2 (where Theorem 3.1 is proved). The rest of this section assumes that Theorem 3.1 is true, and uses it to prove our main results. We start with the proof of the main tool.

**Theorem 3.2** There is a set $Y$ in PP such that ACC($subexp$) $\subseteq$ Dtime$(n^2)^Y$.

**Proof.**

Let $\{C_n\}$ be a circuit family in ACC($subexp$) that accepts $L$. Using the result in Theorem 3.1, we can get a uniform family of circuits $\{D_n\}$ such that for every $n$, $D_n$ is a deterministic depth two circuit having the properties mentioned in the statement of Theorem 3.1.

Let $M_L$ be a nondeterministic Turing machine that, on input $x$, guesses the name of one of the AND gates of $D_n$ ($n = |x|$) and the names of all the inputs of $D_n$ that are connected to this gate. It verifies that the guesses are correct (using the uniformity machine for $\{D_n\}$). It then accepts if and only if the AND gate evaluates to 1 when $x$

[1] This circuit family is not an ACC($subexp$) circuit family because the circuits have arbitrary symmetric gates at their roots. When we say that it is uniform, we are using a slightly different notion of uniformity which is explained in Definition 3.13.

is the input to $D_n$. Since $\{D_n\}$ is uniform and the AND gates have $o(n)$ fan-in, $M_L$ can do this computation in linear time. Note that $\#\mathrm{acc}_{M_L}(x)$ is the number of AND gates of $D_n$ that evaluate to 1 on input $x$.

Let $M_1, M_2, \ldots$ be an enumeration of nondeterministic machines running in linear time. Define the set $Y$ to be $\{\langle i, x, l\rangle : x \in \{0,1\}^* \text{and} \#\mathrm{acc}_{M_i}(x) > l\}$. Note that $Y$ is in $\mathrm{PP}$[2]. With oracle $Y$, a deterministic machine (say $M$) can compute $\#\mathrm{acc}_{M_L}(x)$ in time $n^2$ using the binary search technique. Then, since this is the number of AND gates of $D_n$ that evaluate to 1 on input $x$, it can then in linear time determine if $D_n$ accepts $x$, using the properties guaranteed by Theorem 3.1. Thus membership of $x$ in $L$ can be determined in time $n^2$ relative to oracle $Y$. (Note that the running time can actually be brought down to $o(n)$ by modifying the oracle Turing machine model, but we choose not to do so for the sake of clarity.) ∎

Theorem 3.2 trivially gives us an important corollary (which also follows from a more general lower bound proved later in the chapter):

**Corollary 3.3** $\mathrm{ACC} \subsetneq \mathrm{PP}$.

**Proof.**

Theorem 3.2 implies that $\mathrm{ACC} \subseteq \mathrm{Dtime}(n^2)^Y$ for some $Y \in \mathrm{PP}$. Since $\mathrm{ACC} \subseteq \mathrm{PP}$, suppose for the sake of contradiction that $\mathrm{ACC} = \mathrm{PP}$. Then $\mathrm{ACC} = \mathrm{P} = \mathrm{PP}$. Therefore, $\mathrm{Dtime}(n^3)^Y \subseteq \mathrm{P}^Y \subseteq \mathrm{P} = \mathrm{ACC} \subseteq \mathrm{Dtime}(n^2)^Y$. But this contradicts the time hierarchy theorem of [HS65]. ∎

**Corollary 3.4** The following statements are true:

1. $\mathrm{ACC}(subexp) \subseteq \mathrm{Dtime}(n^2)^{\mathrm{PERM[1]}}$ where PERM[1] refers to the case when only one call is made to PERM.

2. There is a set $Z$ in $\mathrm{C_=P}$ such that $\mathrm{ACC}(subexp) \subseteq \mathrm{Ntime}(n^2)^Z$.

---

[2]Let $M$ be a nondeterministic machine that is given input $\langle i, x, l\rangle$. Suppose $t(|x|)$ is the total number of paths of $M_i$ on input $x$. The computation of $M$ will have $2t(|x|)$ paths; the first $t(|x|)$ of those consist of $t(|x|) - l$ trivially accepting and $l$ trivially rejecting paths, and the other $t(|x|)$ paths will simulate the computation of $M_i$ on $x$. It is easy to see that $\langle i, x, l\rangle \in Y$ iff $\#\mathrm{acc}_{M_i}(x) > l$ iff more than half of the paths of $M$ are accepting.

**Proof.**

1. Let $M_L$ and $M$ be the machines from the proof of Theorem 3.2. Note that if $M$ has access to PERM, it can compute $\#\mathrm{acc}_{M_L}(x)$ in time $n^2$ with just one call to PERM because PERM gives the exact number of accepting paths.

2. As before, let $M_1, M_2, \ldots$ be an enumeration of nondeterministic Turing machines running in linear time. Let $Z$ be the set $\{\langle i, x, l\rangle : x \in \{0,1\}^* \text{ and } \#\mathrm{acc}_{M_i}(x) = l\}$. It is not hard to see that $Z$ is in $\mathrm{C_{=}P}$ (much like $Y \in \mathrm{PP}$ in Theorem 3.2). Let $M_L$ be as above. A nondeterministic machine can compute $\#\mathrm{acc}_{M_L}(x)$ in time $n^2$ using $Z$ as an oracle. It guesses a value $l$ for $\#\mathrm{acc}_{M_L}(x)$ and asks the appropriate query $\langle i, x, l\rangle$ to $Z$.

∎

**Theorem 3.5** $\mathrm{ACC} \subsetneq \mathrm{C_{=}P}$.

**Proof.**

Corollary 3.4 implies that $\mathrm{ACC} \subseteq \mathrm{Ntime}(n^2)^Z$ for a set $Z \in \mathrm{C_{=}P}$. Since $\mathrm{ACC} \subseteq \mathrm{C_{=}P}$, for the sake of contradiction assume that $\mathrm{ACC} = \mathrm{C_{=}P}$. Since co-NP $\subseteq \mathrm{C_{=}P}$ and ACC is closed under complement, hence $\mathrm{ACC} = \mathrm{P} = \mathrm{NP} = \mathrm{C_{=}P}$. Therefore, $\mathrm{Ntime}(n^3)^Z \subseteq \mathrm{NP}^Z \subseteq \mathrm{NP}^{\mathrm{ACC}} = \mathrm{NP}^{\mathrm{P}} = \mathrm{NP} = \mathrm{ACC} \subseteq \mathrm{Ntime}(n^2)^Z$, which contradicts the hierarchy theorem of [FMS78] for nondeterministic time classes. ∎

**Theorem 3.6** The permanent function (PERM) does not have $\mathrm{ACC}(subexp)$ circuits.

**Proof.**

Corollary 3.4 implies that $\mathrm{ACC}(subexp) \subseteq \mathrm{Dtime}(n^2)^{\mathrm{PERM[1]}}$. By the hierarchy theorem of [HS65], we know that $\mathrm{Dtime}(n^2)^{\mathrm{PERM[1]}} \subsetneq \mathrm{Dtime}(n^3)^{\mathrm{PERM[1]}}$. Suppose PERM has $\mathrm{ACC}(subexp)$ circuits. Let $L \in \mathrm{Dtime}(n^3)^{\mathrm{PERM[1]}}$ and let $M$ be the oracle machine that accepts $L$ making at most one call to PERM. Let $L' = \{\langle x, z\rangle : M \text{ accepts } x \text{ if } z \text{ is used as the answer to the query made by } M \text{ to PERM on input } x\}$. Clearly, $L' \in \mathrm{P}$. Similarly, let $L'' = \{\langle x, i\rangle : \text{the } i^{\text{th}} \text{ bit of the query by } M \text{ on input } x \text{ is } 1\}$. Clearly,

$L'' \in$ P as well. A careful reading of Valiant's proof [Val79] reveals that the membership question for any set in P can be reduced to PERM via uniform $AC^0$ circuits. Therefore, by the hypothesis, P has ACC(*subexp*) circuits. Now we can describe an ACC(*subexp*) circuit family for $L$. On any input, the query made to PERM is constructed using the circuits for $L''$, the circuits for PERM are then used to get the answer to the query and finally we use the circuits for $L'$ to determine whether $x \in L$. Since $L'$, $L''$ and PERM all have ACC(*subexp*) circuit families, the resulting family for $L$ is also in ACC(*subexp*). Therefore, using the result in Theorem 3.2, $L \in \text{Dtime}(n^2)^{\text{PERM}[1]}$ which contradicts the hierarchy theorem of [HS65] since we started with an arbitrary $L$ in $\text{Dtime}(n^3)^{\text{PERM}[1]}$.

∎

**Theorem 3.7** PP $\not\subseteq$ ACC(*subsubexp*).

**Proof.**

We claim that if PP $\subseteq$ ACC(*subsubexp*), then PrTime(*subsubexp*) $\subseteq$ ACC(*subexp*). To see this, note that if $L \in \text{PrTime}(t(n))$ for some $t \in$ *subsubexp*, then $L' \in$ PP, where $L' = \{x10^{t(|x|)} : x \in L\}$. Since by assumption $L' \in$ ACC(*subsubexp*), one can build subexponential size circuits for $L$ because the composition of two functions in *subsubexp* is in *subexp*. This implies that PrTime(*subsubexp*) $\subseteq$ ACC(*subexp*).

Note that using the result in Theorem 3.2 and the hierarchy theorem of [HS65], we know that there are sets in $\text{P}^{\text{PP}}$ that are not in ACC(*subexp*). However, if PP is contained in ACC(*subsubexp*), then

$$\begin{aligned}
\text{P}^{\text{PP}} &\subseteq \text{P}^{\text{ACC}(subsubexp)} \\
&\subseteq \text{P}^{\text{Dtime}(subsubexp)} \\
&\subseteq \text{Dtime}(subsubexp) \\
&\subseteq \text{PrTime}(subsubexp) \\
&\subseteq \text{ACC}(subexp)
\end{aligned}$$

The last step follows from the claim above. Hence, $\text{P}^{\text{PP}} \subseteq$ ACC(*subexp*), which is a contradiction, and the theorem follows. ∎

**Theorem 3.8** $\text{C}_=\text{P} \not\subseteq$ ACC(*subsubexp*).

**Proof.**

The proof is similar to the proof of Theorem 3.7. We first claim that if $C_=P \subseteq ACC(subsubexp)$, then $C_=Time(subsubexp) \subseteq ACC(subexp)$. To see this, note that if $L \in C_=Time(t(n))$ for some $t \in subsubexp$, then $L' \in C_=P$, where $L' = \{x10^{t(|x|)} : x \in L\}$. Since by assumption $L' \in ACC(subsubexp)$, one can build subexponential size circuits for $L$. Therefore, $C_=Time(subsubexp) \subseteq ACC(subexp)$. Since $ACC(subexp)$ is closed under complement, we also have that co-$C_=Time(subsubexp) \subseteq ACC(subexp)$.

Using the result in Corollary 3.4 and the hierarchy theorem of [FMS78] for nondeterministic time, we know that there are sets in $NP^{C=P}$ that are not in $ACC(subexp)$. If $C_=P \subseteq ACC(subsubexp)$, then

$$\begin{aligned} NP^{C=P} &\subseteq NP^{ACC(subsubexp)} \\ &\subseteq NP^{Dtime(subsubexp)} \\ &\subseteq Ntime(subsubexp) \\ &\subseteq \text{co-}C_=Time(subsubexp) \\ &\subseteq ACC(subexp) \end{aligned}$$

which is a contradiction. ∎

## 3.2 Proof of Theorem 3.1

This section is devoted to the proof of Theorem 3.1. The definitions, lemmas and theorems presented in this section all lead up to the proof. Since the proof of Theorem 3.1 is fairly involved, we first start with a very high level outline.

**Outline:** Since our goal in this section is to prove that the construction of [BT91] can be done uniformly, it is necessary to prove some preliminary results about uniform constant depth circuits. To that end, we define the notions of "clean" and "nice" circuits, which are circuits that have certain properties that we find essential in presenting our uniformity results. The proof of Theorem 3.1 consists of a number of transformations of a circuit. Without loss of generality, we start out with a "nice" circuit family. After each transformation, we will have a circuit that may not obviously satisfy the "niceness"

condition, but at least satisfies the weaker notion of being "clean". We show that this clean circuit can then be transformed into a nice circuit of the same depth, and the process repeats. Our results concerning the "nice" circuit families are proved using a variation of the alternating Turing machine model.

The main steps in the transformation are:

- All the AND and OR gates in the circuits are replaced by constant depth probabilistic subcircuits. This step removes all the OR gates from the circuits and the only remaining AND gates have small fan-in. The circuits are probabilistic but the number of probabilistic bits used in each case is small and is in fact a simple function of the size of the circuit.

- All the MOD gates in the circuit with composite moduli are replaced with equivalent subcircuits so that the resultant circuits consist only of MOD gates with prime moduli.

- The circuits are now made deterministic by taking separate copies of those for each setting of the probabilistic bits and connecting all outputs to a MAJORITY gate.

- A general technique is used, showing how nice circuits with small fan-in AND gates can be replaced by equivalent circuits with the same depth, whose outputs are MOD gates.

- An induction is begun, where each step reduces the depth of the circuit. At the beginning of the inductive step, the circuit consists of a symmetric gate on the output level, where the inputs to the symmetric gate are "nice" ACC type circuits with $\mathrm{MOD}_p$ gates feeding into the symmetric gate. Then, using techniques developed by Toda [Tod91] and Yao [Yao90], we create an equivalent circuit with a new symmetric gate that "absorbs" the level of $\mathrm{MOD}_p$ gates; thus the new circuit has smaller depth.

### 3.2.1 Nice Circuits

**Definition 3.1** A circuit family $\{C_n\}$ is *well-named* if for every $n$, the name of the output gate of $C_n$ can be computed from $n$ (in binary) in polynomial time (i.e., in $(\log n)^{O(1)}$ time).

**Definition 3.2** A circuit family $\{C_n\}$ is said to have the *strong connection* property if for all $n$, for every connection $g \to h$ in $C_n$, where $i$ is the number such that $g$ is the $i^{\text{th}}$ input to $h$ (assuming lexicographic ordering), it is the case that $h$ can be computed in polynomial time from $\langle n, g, i\rangle$, and additionally, given $\langle n, h, g\rangle$, the number $i$ can be computed in polynomial time. Under the weaker assumption that this condition holds whenever $h$ is an AND gate, $\{C_n\}$ is said to have the *strong connection property for ANDs*.

**Definition 3.3** A circuit family $\{C_n\}$ is said to have *small fan-in AND gates* if for every $n$, the fan-in of each AND gate in $C_n$ is polylogarithmic in the size of $C_n$.

**Definition 3.4** Let $C$ be a circuit and let $P$ be a path in $C$ from the output gate to an input gate (say $t$). Let $G_1, G_2, \ldots, G_k$ be the sequence of the types of gates occurring on $P$ so that $G_1$ is the type of the output gate of $C$ and $G_k$ is the type of the gate that $t$ is connected to. Then the sequence $(G_1, G_2, \ldots, G_k)$ is defined as the *signature* of the path $P$.

**Definition 3.5** The *compression* of a signature $s$ is the sequence $s'$ that results from applying the following operation as many times as possible to $s$: replace "AND, AND" by "AND" and replace "OR, OR" by "OR". That is, the compression of $s$ contains no two adjacent ANDs or ORs.

**Definition 3.6** A circuit family $\{C_n\}$ is *clean* if

- It is well-named.
- It has the strong connection property for ANDs.
- Every path from an output gate to an input gate in any circuit $C_n$ has the same signature. (Note that only constant depth circuit families can be clean.)

**Definition 3.7** A circuit family $\{C_n\}$ of size $s(n)$ is *nice* if it has the following properties:

- $\{C_n\}$ is clean.
- For every $n$, the fan-in of every gate $g$ in $C_n$ can be computed from $g$ in time $(\log s(n))^{O(1)}$.
- For every $n$, the depth of a gate $g$ in $C_n$ can be computed from $g$ in time $(\log s(n))^{O(1)}$.
- Each circuit $C_n$ is in tree form (excluding the inputs and negated inputs, which may fan out to many gates at level 1).
- $\{C_n\}$ has the strong connection property.
- For all input lengths $n$, all the MOD gates in $C_n$ have the same fan-in.

The main result to be proved in this subsection is to show that any uniform ACC type circuit family can be transformed into an equivalent nice family. The actual result is stated later as Lemma 3.13 and its proof follows from a sequence of lemmas that are presented below. We use a version of the ATM model that allows existential, universal and MOD states as explained in Chapter 2 (see Definitions 2.7 and 2.8). Each configuration of an ATM has either zero, one, or two successor configurations (i.e., the fan-out of any node in the computation tree is at most two). We follow the convention that the ATM is always provided the length of the input (in binary) on the work tape as part of its initial configuration on a particular input. This convention has been introduced to simplify the proof. (It is worthwhile to note that the input length can be computed deterministically in logarithmic time (see [BCGR92]) but this requires multiple accesses to the input along a given computation path.) We consider ATMs that access their input only at the leaves. (That is, the only configurations that depend on the input are halting configurations. These are of two types: those that accept if and only if bit $i$ of the input is 1, and those that accept if and only if the complement of bit $i$ is 1 (for some $i$ that is recorded on the address tape). The results in [Sip83] show that this convention can be introduced without loss of generality.)

It is necessary for us to define a notion of "clean" ATMs corresponding to our notion of "clean" circuit families. This is accomplished using the following definitions:

**Definition 3.8** Let $\sigma$ and $\tau$ be two different configurations of an ATM. If $\tau$ is reached from $\sigma$ via a path that contains an alternation only in the step at which $\tau$ is reached, then $\tau$ is called a *primary descendent* of $\sigma$.

**Definition 3.9** For a computation path of an ATM on an input, let $C_1, C_2, \ldots C_k$ be the sequence of configurations such that $C_1$ is the initial configuration, and $C_{i+1}$ is a primary descendent of $C_i$. The *profile* of the path is the sequence $t_1, t_2, \ldots t_k$ such that if configuration $C_i$ is existential (universal, $\text{MOD}_m$), then $t_i$ = OR (AND, $\text{MOD}_m$).

**Definition 3.10** An ATM is *clean* if every path in every alternation tree of the ATM on every input has the same profile. (Note that only ATMs making $O(1)$ alternations can be clean.)

**Definition 3.11** An ATM running in time $t(n)$ has *well-behaved universal configurations* if each universal configuration has $t(n)^{O(1)}$ primary descendents, and given a universal configuration $\sigma$ and a number $i$, the $i^{\text{th}}$ primary descendent of $\sigma$ can be computed in time $t(n)^{O(1)}$.

**Lemma 3.9** Let $L \subseteq \{0,1\}^*$, let $s$ be a function in *subexp*, and let $L$ be accepted by a uniform family $\{C_n\}$ of depth $d$ circuits ($d = O(1)$) of type ACC($s(n)$). Then $L$ is accepted by an ATM $M$ that has existential ($\exists$), universal ($\forall$) and MOD states (for the same set of moduli), that runs in time $(\log s(n))^{O(1)}$ and has alternation depth $a = O(1)$. Moreover,

- If $\{C_n\}$ is clean, then the profile of $M$ is the compression of the signature of $\{C_n\}$.
- If $\{C_n\}$ is clean and has small fan-in AND gates, then $M$ has well-behaved universal configurations.

**Proof.**

Suppose $L$ is accepted by a uniform circuit family $\{C_n\}$. Let $U$ be the uniformity machine for $\{C_n\}$. $M$ behaves as follows:

On input $x$, (with $n = |x|$ on the work tape)

($\exists$) guess the name of the output gate (say $g$) of $C_n$ of length $(\log s(n))^{O(1)}$.

Use $U$ to verify that $g$ is a gate in $C_n$. (i.e., check that $U$ accepts $\langle n, g, g \rangle$.)

($\forall$) gates $h$ of length $(\log s(n))^{O(1)}$ check that $U$ rejects $\langle n, h, g \rangle$

(so that $g$ is indeed the output gate)

Call Eval($g$).

**Eval($g$)**

If $g$ is an OR gate then

($\exists$) guess $h$ (an input to $g$) of length $(\log s(n))^{O(1)}$.

If $U$ rejects $\langle n, g, h \rangle$ then reject

else call Eval($h$).

If $g$ is an AND gate then

($\forall$) guess $h$ (an input to $g$) of length $(\log s(n))^{O(1)}$.

If $U$ rejects $\langle n, g, h \rangle$ then accept

else call Eval($h$).

If $g$ is a $\text{MOD}_m$ gate then

Switch to a $\text{MOD}_m$ configuration.

($\exists$) guess $h$ (an input to $g$) of length $(\log s(n))^{O(1)}$.

If $U$ rejects $\langle n, g, h \rangle$ then reject

else call Eval($h$).

If $g$ is a constant gate then

Accept iff $g$ is the constant 1 gate.

If $g$ is an input gate then

Accept iff the corresponding input is 1.

end (Eval).

It is fairly obvious that $M$ accepts $x$ iff $C_{|x|}$ evaluates to 1 on input $x$. Note that $M$ consults its input only at the leaves. It is clear that $M$ makes a constant number

of alternations and runs in time $(\log s(n))^{O(1)}$. Indeed, the most time-consuming part of the simulation involves running the uniformity machine $U$. The constructibility conditions on $s$ are also essential here.

If $\{C_n\}$ is clean, then it is well-named, and thus the name of the output gate $g$ can be computed deterministically. Also, since all circuits in $\{C_n\}$ have the same signature, each output gate is of the same type. If the type of the output gate is $\mathrm{MOD}_m$, for instance, we can avoid the extra two levels of alternation caused by the processing outside the routine Eval, by starting out in a $\mathrm{MOD}_m$ configuration, deterministically computing $g$ (i.e., with existential nodes of fan-out 1), existentially guessing $h$, rejecting if $U$ rejects $\langle n, g, h\rangle$, and otherwise proceeding to Eval($h$). The case when the output gate is and AND or OR gate is handled similarly. Thus if $\{C_n\}$ is clean, the profile of $M$ can easily be seen to be the compression of the signature of $\{C_n\}$.

If $\{C_n\}$ has the strong connection property for ANDs and all AND gates have fan-in $(\log s(n))^c$, then instead of universally guessing an input $h$ to an AND gate $g$, universally guess a number $i \leq (\log s(n))^c$ and deterministically compute the name of the gate $h$. If $M$ is simulating $r$ consecutive levels of AND gates of $C_n$, it is not hard to see that each universal configuration of $M$ will have at most $(\log s(n))^{rc}$ primary descendents, and $M$ thus has well-behaved universal configurations.

The other claims of the Lemma are easily seen to hold. ∎

Lemma 3.9 does not guarantee the existence of a clean ATM accepting a language when the given circuit family is not already clean. This is remedied by the following lemma.

**Lemma 3.10** If $L$ is accepted by an ATM $M$ that makes a constant number of alternations between $\mathrm{MOD}_{m_1}, \mathrm{MOD}_{m_2}, \ldots, \mathrm{MOD}_{m_k}, \exists$ and $\forall$ states and runs in time $t(n)$ then $L$ is accepted by a clean ATM $N$ running in $O(t(n))$ time with a constant number of alternations between $\mathrm{MOD}_{m_1}, \mathrm{MOD}_{m_2}, \ldots, \mathrm{MOD}_{m_k}, \exists$ and $\forall$ states.

**Proof.**

Suppose $M$ makes at most $\lambda$ alternations on any input. Then $N$ has the sequence $\mathrm{MOD}_{m_1}, \mathrm{MOD}_{m_2}, \ldots, \mathrm{MOD}_{m_k}, \exists, \forall$ (repeated $\lambda$ times) hardwired into its finite control.

$N$ simply simulates $M$ but follows the profile in its finite control. If $N$ is trying to simulate a move that does not involve an alternation or that involves moving into a state that has the same type as the next type in its profile, it simply proceeds with the simulation and behaves exactly as $M$ does. In the case of a type mismatch, $N$ behaves as follows:

- If the next state in the sequence is universal (existential), then it executes a one-ary universal (existential) branch and continues the simulation. (Note that amounts to adding a "dummy" node in the alternating tree.)
- If the next state in the sequence is a $\text{MOD}_m$ state for some $m$, then it executes a $m$-way $\text{MOD}_m$ branch. It trivially accepts along $m-1$ of these branches (following the profile) and continues the simulation on the remaining one.

It is fairly obvious that $N$ is clean and for every $x$, $N$ accepts $x$ iff $M$ accepts $x$. ∎

Our main reason for introducing the ATM model is the following lemma, which enables us to construct "nice" circuits.

**Lemma 3.11** Let $2^{t(n)}$ be constructible, and suppose $L$ is accepted by a clean ATM $M$ running in time $t(n)$. Then $L$ is accepted by a clean ATM $N$ with the same profile (and hence with the same alternation depth) that runs in time $t(n)^{O(1)}$ and also has the following properties:

1. Given a configuration $\sigma$ on an input of length $n$, the number of primary descendents of $\sigma$ is computable from $\sigma$ in time $t(n)^{O(1)}$.
2. Given a configuration $\sigma$ on an input of length $n$, the alternation depth of $\sigma$ is computable from $\sigma$ in time $t(n)^{O(1)}$.
3. Given a configuration $\sigma$ and number $i \leq$ the number of primary descendents of $\sigma$, the $i^{\text{th}}$ primary descendent of $\sigma$ (under the usual lexicographic ordering) can be computed in time time $t(n)^{O(1)}$ from the encoding of $\sigma$.
4. All the MOD configurations in the computation tree have the same number of primary descendents.

5. If $M$ has well-behaved universal configurations, then $N$ also has this property.

**Proof.**

The proof is very similar to the proof of Lemma 3.9. We will need to settle on some convention of encoding paths in an alternation tree, with the property that for every path of length $i \leq t(n)$ in an alternation tree, there is exactly one string of length $2 \cdot t(n)$ that denotes that path. This can easily be accomplished by encoding sequences in {left, right, stop}$^*$ in the obvious way; note that there will be many strings that do not correspond to any path in the tree. Similarly, pick some encoding of configurations of $M$ so that any configuration $\sigma$ of $M$ on inputs of length $n$ has a unique encoding using $c \cdot t(n)$ bits (for some constant $c$). Again, many strings of length $c \cdot t(n)$ will not correspond to any configuration of $M$.

$N$ will begin its computation on $x$ by first computing (deterministically) $t(n)$. (Note that this can be done regardless of whether the initial configuration of $N$ is existential, universal, or MOD$_m$.) If $M$ has well-behaved universal configurations, then let $I(n) = b \log t(n)$ for some constant $b$; otherwise let $I(n) = t(n)$. (Note that the decision of which value to use for $I(n)$ can be encoded in the finite control of $N$.) Then $N$ will set $\sigma$ to be equal to the initial configuration of $M$, and run the routine Eval($\sigma$).

**Eval($\sigma$)**

If $\sigma$ is an existential or MOD$_m$ non-halting configuration then

existentially guess strings $w$ of length $2 \cdot t(n)$ and $\tau$ of

length $c \cdot t(n)$.

If $w$ encodes a path from $\sigma$ to configuration $\tau$, where

the last step in the path involves an alternation (so $\tau$ is

a primary descendent of $\sigma$)

then enter a configuration of the same type as $\tau$ and call Eval($\tau$)

else call Trivial(*reject*)

If $\sigma$ is a universal non-halting configuration then there are two cases:

(1) The machine $M$ has well-behaved universal configurations.

Universally guess $i \leq b \log t(n)$. Let $\tau$ be the $i^{\text{th}}$

primary descendent of $\sigma$. Call Eval($\tau$). (If there

is no such $\tau$, then call Trivial(*accept*).)

(2) Otherwise.

Universally guess strings $w$ of length $2 \cdot t(n)$ and $\tau$ of length $c \cdot t(n)$.

If $w$ encodes a path from $\sigma$ to configuration $\tau$,

where the last step in the path involves an alternation (so

$\tau$ is a primary descendent of $\sigma$)

then enter a configuration of the same type as $\tau$

and call Eval($\tau$)

else call Trivial(*accept*)

If $\sigma$ is a halting configuration, then

Accept iff $\sigma$ is accepting. (Note that this may involve accessing

the input, if $\sigma$ depends on input bit $i$ for some $i$.)

end (Eval).

The routine Trivial($d$) (for $d \in \{$*accept, reject*$\}$) used in the routine Eval is a simple routine that depends on the number of alternations executed thus far by $N$ in its simulation of $M$. If the next step in the profile calls for computation of type $\exists$ ($\forall$), then $N$ executes a $2^{(c+2)t(n)}$-way existential (universal) branch, all of which in turn call Trivial($d$). If the next step in the profile calls for computation of type $\mathrm{MOD}_m$, and $d =$ *accept* (respectively, $d =$ *reject*), then $N$ enters a $\mathrm{MOD}_m$ state, executes a $2^{(c+2)t(n)}$-way existential branch all of which call Trivial(*reject*) (respectively, the first of which calls Trivial(*accept*) and the rest of which call Trivial(*reject*)).

Machine $N$ uses its worktape to record the path in the alternation tree leading to the current configuration. Thus no configuration of $N$ will label two distinct nodes in the alternation tree.

Let us now verify the various properties claimed in the statement of the lemma.

Given $\sigma$ a configuration of $N$, one can trace through the path in the alternation tree leading to $\sigma$ (since this information is recorded in $\sigma$). This allows one to compute the alternation depth of $\sigma$, as well as to find the configuration $\tau$ reached after the last

alternation on this path, and compute the number $j$ of moves with fan-out 2 that have occurred along this path between $\tau$ and $\sigma$. If $\sigma$ is an $\exists$ or MOD configuration, the number of primary descendents of $\sigma$ is $2^{(c+2)t(n)-j}$. If $\sigma$ is a $\forall$ configuration, then this number is $2^{(c+2)I(n)-j}$. In the particular case that $\sigma$ is a MOD configuration, note that $j = 0$; thus all the MOD configurations have the same number of primary descendents. Furthermore, if $\sigma'$ is the $i^{\text{th}}$ primary descendent of $\sigma$, then the number $i$ is encoded in $(c+2)t(n) - j$ consecutive positions in the bit string encoding the path leading to $\sigma'$, thus enabling us to compute $\sigma'$ given $\langle n, \sigma, i \rangle$. The other claims of the lemma are easy to verify. ∎

**Lemma 3.12** Let $L$ be accepted by an ATM $M$ satisfying the conditions of Lemma 3.11, running in time $t(n)$. Then there is a nice ACC($2^{O(t(n))}$) circuit family $\{C_n\}$ accepting $L$, such that the signature of $\{C_n\}$ is the same as the profile of $M$. Furthermore, if $M$ has well-behaved universal configurations, then $\{C_n\}$ has small fan-in AND gates.

**Proof.**

The proof of this lemma is by a standard simulation of the sort introduced by [Ruz81]. The output gate of $C_n$ will be labeled by the initial configuration of $N$ on an input of length $n$ (i.e., with $n$ recorded on the worktape, as per the conventions of our ATM model). The inputs to any gate labeled with configuration $\sigma$ will be all of the primary descendents of $\sigma$. Universal configurations are represented by AND gates, existential configurations by OR gates, and MOD$_m$ configurations by MOD$_m$ gates. Halting configurations are either constant 1 or 0 gates (if they do not depend on the input) or are input gates connected to (negated) input $i$ (if they access input bit $i$).

It is easily verified that $\{C_n\}$ satisfies the requirements of the lemma. ∎

**Lemma 3.13** Suppose $\{C_n\}$ is an ACC(*subexp*) circuit family. Then there exists an equivalent nice family $\{D_n\}$ of subexponential size. Furthermore,

- If $\{C_n\}$ is clean, then the signature of $\{D_n\}$ is the compression of the signature of $\{C_n\}$.

- If $\{C_n\}$ is clean and has small fan-in AND gates, then $\{D_n\}$ has small fan-in AND gates.

**Proof.**

This follows immediately from Lemmas 3.9, 3.10, 3.11 and 3.12. ∎

### 3.2.2 Transformations on Circuits

In this section we present the results that enable us to make transformations on uniform circuit families. A transformation on a circuit usually involves replacing every instance of a particular kind of gate by an equivalent subcircuit.

**Definition 3.12** Suppose $G$ is a particular type of gate. Let $\{G_r\}$ denote a family of gates such that the gate $G_r$ is of type $G$ and takes $r$ inputs. Let $\{E_r\}$ be a family of subcircuits so that for every $r$, $E_r$ takes $r$ inputs and has a single output. We will assume an ordering on the inputs of $G_r$ and $E_r$ and let $x_1, x_2, \ldots, x_r$ denote the inputs to $G_r$ and $y_1, y_2, \ldots, y_r$ denote the inputs to $E_r$. We say that $E_r$ *replaces* $G_r$ in a circuit $C$ if we remove $G_r$ from $C$ and put $E_r$ in its place in such a way that the output gate of $E_r$ is connected to exactly the gates that $G_r$ is connected to in $C$, and the inputs to $G_r$ now become inputs to $E_r$ so that for all $i$, $1 \leq i \leq r$, $x_i = y_i$. In general, when we talk about replacing a gate type $G$ in a circuit, we will mean that all occurrences of $G$ in the circuit are replaced simultaneously.

**Lemma 3.14** Suppose $\{C_n\}$ and $\{E_r\}$ are nice circuit families. Let $G$ denote a particular type of gate used in the circuits of $\{C_n\}$. For every $n$, let $\{D_n\}$ denote the circuit family obtained by replacing *all* occurrences of $G$ (of the form $G_r$ for various $r$) by a subcircuit $E_r$. Then the circuit family $\{D_n\}$ is clean.

**Proof.**

It is clear that $\{D_n\}$ is well-named and that every path from output to input has the same signature. Thus we need only show that $\{D_n\}$ is uniform and has the strong connection property.

Consider the transformation from $C_n$ to $D_n$ for a particular value of $n$. Let $g$ (with fan-in $r$) be an instance of $G$ in $C_n$ and let $E_r$ be the subcircuit that replaces $g$. Suppose $E_r$ consists of the gates $h_0, h_1, \ldots, h_s$ where $h_0$ is the output gate of $E_r$. The names of these gates in the new circuit $D_n$ will be $g\#h_i$, for $0 \leq i \leq s$. Let $L_0$ be the direct connection language for $\{C_n\}$, $L_1$ for $\{E_r\}$ and $L$ for $\{D_n\}$. Similarly, let $f_0$, $f_1$, and $f$ be the functions that, given $\langle n, g, h\rangle$, compute the number $i$ such that $h$ is the $i^{\text{th}}$ input to $g$ in $C_n$, $E_n$, and $D_n$, respectively, and let $f_0'$, $f_1'$, and $f'$ be the related functions that compute $h$ given $\langle n, g, i\rangle$. To accept $L$, and to compute $f$, one has to consider the following cases:

1. Strings of the form $\langle n, g, h\rangle$ where neither $g$ nor $h$ are of type $G$. In this case $\langle n, g, h\rangle \in L \iff \langle n, g, h\rangle \in L_0$. Also, $f(n, g, h) = f_0(n, g, h)$.

2. Strings of the form $\langle n, g\#h, g\#h\rangle$. This is done as follows:

   - Check that $\langle n, g, g\rangle \in L_0$ and that $g$ has type $G$.
   - Compute the fan-in $r$ of $g$ from the description of $g$.
   - Check that $\langle r, h, h\rangle \in L_1$.

3. Strings of the form $\langle n, g\#h, g\#h'\rangle$ with $h \neq h'$. This is done as follows:

   - Check that $\langle n, g, g\rangle \in L_0$ and that $g$ has type $G$.
   - Compute the fan-in $r$ of $g$ from the description of $g$.
   - Check that $\langle r, h, h'\rangle \in L_1$.
   - Note that $f(n, g\#h, g\#h') = f_1(n, g\#h, g\#h')$.

4. Strings of the form $\langle n, g', g\#h_0\rangle$ where $G$ is not the type of $g'$. This is done as follows:

   - Check that $\langle n, g', g'\rangle$ and $\langle n, g, g\rangle \in L_0$, and that $g$ has type $G$.
   - Compute the fan-in $r$ of $g$ from the description of $g$.
   - Check that $\langle r, h_0, h_0\rangle \in L_1$ ($h_0$ is the output gate of $E_r$).
   - Check that $\langle n, g', g\rangle \in L_0$.

- Note that $f(n, g', g\#h_0) = f_0(n, g', g)$.

5. Strings of the form $\langle n, g\#h, g' \rangle$, where $G$ is not the type of $g'$. This is done as follows:

   - Check that $\langle n, g, g \rangle$ and $\langle n, g', g' \rangle$ are in $L_0$, where $g$ has type $G$.
   - Check that $\langle n, g, g' \rangle \in L_0$.
   - Compute the fan-in $r$ of $g$ from its description.
   - Check that $\langle r, h, h \rangle \in L_1$.
   - Compute the number $j$ such that $g'$ is the $j^{\text{th}}$ input to $g$ (using the strong connection property).
   - Let $x_1, x_2, \ldots, x_r$ denote the inputs to $E_r$. Check that $\langle r, h, x_j \rangle \in L_1$.
   - Note that $f(n, g\#h, g') = j$.

6. Strings of the form $\langle n, g'\#h, g\#h_0 \rangle$ where both $g$ and $g'$ are of type $G$.

   - Check that $\langle n, g, g \rangle$ and $\langle n, g', g' \rangle$ are in $L_0$.
   - Compute the fan-in $r$ of $g$ and check that $h_0$ is the output gate of $E_r$.
   - Compute the fan-in $r'$ of $g'$ and check that $\langle r', h, h \rangle \in L_1$.
   - As in the previous case, check that $g$ is the $j^{\text{th}}$ input to $g'$ and that $h$ is connected to input $j$ of $E_{r'}$.

It is not hard to see that all the above cases can be checked within the required time bounds and hence the new circuit family $\{D_n\}$ is uniform as well.

A similar analysis shows that $f'$ can also be computed in time polynomial in the length of its input, and thus $\{D_n\}$ has the strong connection property. ∎

**Lemma 3.15** Suppose $L$ is accepted by an ACC(*subexp*) family $\{C_n\}$. Then $L$ is accepted by a nice probabilistic ACC(*subexp*) circuit family $\{D_n\}$[3] such that

[3]Note that the circuits in $\{D_n\}$ are probabilistic and hence also have probabilistic inputs, but when we say $D_n$ we mean the circuit that has $n$ nonprobabilistic inputs. We follow this convention because the proof shows how to convert $C_n$ into $D_n$ for every $n$.

- $\{D_n\}$ has no $\mathrm{MOD}_m$ gates for composite modulus $m$.
- $\{D_n\}$ has small fan-in AND gates.
- For every $n$, the number of probabilistic inputs in $D_n$ is polylogarithmic in the size of $D_n$.

**Proof.**

By Lemma 3.13, we may assume that $\{C_n\}$ is nice.

Let $n$ be fixed. The transformation $C_n \to D_n$ is carried out by performing the following sequence of steps:

- By a construction in the proof of Lemma 13 in [AH90], one can replace the AND and OR gates in the circuit by nice depth 6 probabilistic circuits with $\mathrm{MOD}_2$ gates and small fan-in AND gates. (This construction is based on an idea of Valiant and Vazirani in [VV86]; similar constructions may be found in work by Toda [Tod91] and Kannan, Venkateswaran, Vinay and Yao [KVVY92].) The size of the new circuit is only polynomially more than that of the old one. If the AND or OR gate being replaced has $r$ inputs, then the probabilistic circuit that replaces it uses $O((\log r)^3)$ random bits. The probabilistic circuits have the property that the probability of error for the whole circuit is less than $\frac{1}{4}$ after all the AND and OR gates have been replaced by these probabilistic circuits, even when the same $O((\log s(n))^3)$ probabilistic bits are fed into the probabilistic inputs of each of these subcircuits. (Even though Allender and Hertrampf discuss space uniformity, it is clear from their proof that the probabilistic circuits are uniform even in our sense of uniformity.) We can now apply Lemma 3.14 to prove that the new circuit family (now probabilistic) is clean, and thus by Lemma 3.13 there is an equivalent nice circuit family $\{C_n^1\}$. Note that $\{C_n^1\}$ has small fan-in AND gates and has no OR gates.
- Suppose the circuit $C_n^1$ contains a $\mathrm{MOD}_m$ gate (call it $G$) where $m$ is composite. Let

$$m = \prod_{i=1}^{t} a_i^{e_i}$$

where $a_i < a_{i+1}$ for all $i$ such that $1 \leq i \leq t-1$ and for all $i$, $1 \leq i \leq t$, $a_i$ is prime and $e_i > 0$. We use the elementary fact that $x \equiv 0 \pmod{m}$ $\iff x \equiv 0 \pmod{a_i^{e_i}}$ for all $i$, $1 \leq i \leq t$ to change $G$ into an AND of $\text{MOD}_{a_i^{e_i}}$'s. Suppose $G$ has $r$ inputs. For each $m$, the subcircuit family $\{E_r\}$ that replaces the $\text{MOD}_m$ gates is easily seen to be nice. The subcircuit $E_r$ has depth two, with an AND gate at the top level and $\text{MOD}_{a_i^{e_i}}$ gates at the bottom level for all $i$, $1 \leq i \leq t$. The top level AND gate has fan-in $t$ and is connected to each of the MOD gates at the second level. All the MOD gates at the second level have fan-in $r$ and are all connected to each of the inputs of the gate $G$. We can now use the result of Lemma 3.14 to conclude that the new circuit family is clean. Moreover, the family contains MOD gates with only prime power moduli. The subcircuit $E_r$, other than its input gates, contains only a constant number of gates that depends on $m$. Since the original circuit family $\{C_n\}$ only has MOD gates for a fixed set of moduli, the size of the circuit after this step goes up by at most a constant factor. We again use Lemma 3.13 to get a nice family of probabilistic circuits $\{C_n^2\}$ that has no composite MOD gates, no OR gates, and small fan-in AND gates.

- This step eliminates all the MOD gates that have moduli of the form $p^e$ where $p$ is prime and $e > 1$ from $C_n^2$ and replaces them with subcircuits consisting of AND and $\text{MOD}_p$ gates. Suppose $C_n^2$ contains a $\text{MOD}_{p^e}$ gate $G$ for some prime $p$ and $e > 1$. This step uses the following result (for references, see e.g. [BT91]):
  $x$ is congruent to 0 (mod $p^e$) if and only if each of $x, \binom{x}{p}, \binom{x}{p^2}, \ldots, \binom{x}{p^{e-1}}$ are congruent to 0 (mod $p$). If $x = \sum_{i=1}^{r} x_i$, then for $1 \leq j \leq e-1$:

$$\binom{x}{p^j} = \binom{x_1+x_2+\cdots+x_r}{p^j} = \sum_{S \subseteq \{1,2,\ldots,r\}, |S|=p^j} \bigwedge_{k \in S} x_k$$

  The subcircuit that replaces $G$ is a three level subcircuit that is described as follows:

  - The top level consists of an AND gate that has fan-in $e$.

- The middle level consists of $e$ $\mathrm{MOD}_p$ gates and each of those is connected to the top level AND gate. For all $j$, $0 \leq j \leq e-1$, the $j^{\text{th}}$ $\mathrm{MOD}_p$ gate outputs 1 if and only if $\binom{x}{p^j} \equiv 0 \pmod{p}$. If $G$ has fan-in $r$, then the $j^{\text{th}}$ $\mathrm{MOD}_p$ gate at this level has fan-in $\binom{r}{p^j}$, one corresponding to each subset of the inputs of size $p^j$.
- The bottom level consists of $\sum_{j=1}^{e-1} \binom{r}{p^j}$ AND gates divided into $e-1$ groups. For all $j$, $1 \leq j \leq e-1$, the $j^{\text{th}}$ group consists of $\binom{r}{p^j}$ AND gates, one corresponding to each subset of the inputs of size $p^j$. The inputs to a particular gate in the $j^{\text{th}}$ group are the $p^j$ inputs in the subset to which it corresponds and it fans out to the $j^{\text{th}}$ $\mathrm{MOD}_p$ gate at the middle level. Note that all the AND gates introduced here have constant fan-in.

It is not hard to see that the subcircuit family described above is nice for every prime power $p^e$. (The only point that is not completely obvious is checking that the strong connection property holds, but this is straightforward to verify.) Using Lemma 3.14 we can now replace every MOD gate with a prime power modulus with a subcircuit that consists only of MOD gates with prime moduli and we now get a clean circuit family that only has AND gates and MOD gates with prime moduli. The size of the subcircuit that replaces a $\mathrm{MOD}_{p^e}$ gate is $O(\sum_{j=1}^{e-1} \binom{r}{p^j})$ which is a polynomial in the size of the circuit $C_n^2$, and thus the new circuit family also has subexponential size. The proof is completed by appeal to Lemma 3.13.

■.

### 3.2.3 Circuits with Symmetric Gates

In order to prove Theorem 3.1 we need to show how to convert an ACC(*subexp*) circuit family into a uniform deterministic depth two circuit family that has a symmetric gate at the root and AND gates of small fan-in at the bottom level. So far we have only dealt with ACC type circuits. To proceed, we need to deal with circuits that have arbitrary symmetric gates (but only at the root). However, since most of the results proved so far only deal with uniform ACC type circuits, we need to expand the notion of uniformity

a little so that the results can also be used with circuits that have arbitrary symmetric gates at the root. The idea here is to treat the symmetric gate in a special way. We think of the $n^{\text{th}}$ circuit as one with a symmetric gate at the root, that takes as inputs the outputs of $f(n)$ different ACC type circuits, where $f(n)$ is an easily computable function of $n$. We require that each of these ACC type circuits be uniform according to the notion of uniformity that we have used so far. In other words, for every $n$, we now define a uniform sequence of ACC type circuits. The sequence consists of $f(n)$ circuits that are indexed by a variable $t$. In addition, we require that the symmetric function at the top gate be easily computable. The circuit family consists of a sequence and a symmetric function for every $n$. The new notion of uniformity is formally explained in the following definition:

**Definition 3.13** Let $f : \mathbf{N} \rightarrow \mathbf{N}$ be a function. Then $\{C_{n,t} : n \in \mathbf{N}, 1 \leq t \leq f(n)\}$ is a *uniform family of ACC sequences* if there is a constant $d$ and a finite set $S$ such that for all $n$ and for all $t$, $C_{n,t}$ is a circuit of depth $d$ taking inputs from the set $\{x_1, x_2, \ldots, x_n\}$ and having AND, OR, and $\text{MOD}_m$ gates (for $m \in S$) and the direct connection language defined as

$$\{\langle n, t, g_1, g_2\rangle : \quad g_1 = g_2 \text{ and } g_1 \text{ is a gate in } C_{n,t} \text{ or}$$
$$g_1 \neq g_2 \text{ and } g_2 \text{ is an input to } g_1 \text{ in } C_{n,t}\}$$

can be recognized in polynomial time. A uniform family of ACC sequences $\{C_{n,t} : n \in \mathbf{N}, 1 \leq t \leq f(n)\}$ together with a function $\text{SYM} : \mathbf{N} \times \mathbf{N} \rightarrow \{0, 1\}$, defines a uniform SYMACC circuit family $\{D_n\}$ such that for every $n$,

- $D_n$ is a circuit with a symmetric gate at the output level that computes $\text{SYM}(n, i)$ where $i$ is the number of its inputs that evaluate to 1.
- The symmetric gate has fan-in $f(n)$ and the output gates of $C_{n,t}$, $1 \leq t \leq f(n)$, are connected to it.
- Given $n$ and $i$, $f(n)$ and $\text{SYM}(n, i)$ can be computed in time polylogarithmic in the size of $D_n$.

Note that the results proved so far also hold with this new notion of uniformity. In particular, letting $f(n) = 1$ for all $n$ and letting SYM be the identity function reduces this to the old notion of uniformity. Also, we will use the fact that Lemma 3.13 also holds in this new setting. That is, given a uniform *clean* family of ACC sequences, there is an equivalent *nice* family of ACC sequences with the same profile and of approximately the same size. (In proving the analog of Lemma 3.13 in this new setting, the index $t$ of circuit $C_{n,t}$ would be provided to the ATM as an additional parameter on the worktape, along with $n$.)

**Lemma 3.16** Let $L$ be accepted by an ACC(*subexp*) circuit family $\{C_n\}$. Then there is a constructible subexponential function $s$ and there is a constant $c$ such that $L$ is accepted by a deterministic circuit family $\{D_n\}$ where for every $n$, $D_n$ has a MAJORITY gate at the root, connected to the output gates of $C_{n,t}$, $1 \leq t \leq 2^{(\log s(n))^c}$ where $\{C_{n,t} : n \in \mathbf{N}, 1 \leq t \leq 2^{(\log s(n))^c}\}$ is a uniform family of ACC sequences with small fan-in AND gates, no OR gates, and no $\text{MOD}_m$ gates for composite $m$.

**Proof.**

By Lemma 3.15, if $L$ is accepted by an ACC(*subexp*) circuit family, then $L$ is accepted by a nice ACC(*subexp*) family of *probabilistic* circuits with small fan-in AND gates, no OR gates, and no $\text{MOD}_m$ gates for composite $m$, using at most $(\log s(n))^c$ probabilistic bits (for some constant $c$), where $s(n)$ bounds the size of $C_n$.

Now construct the sequence of circuits $\{C_{n,t}\}$ where $t$ is a bit string of length $(\log s(n))^c$. The gates in $\{C_{n,t}\}$ will have names of the form $\langle t, g\rangle$ where $g$ is a gate in $C_n$, and the connections among all gates are the same, except that if gate $g$ in $C_n$ is connected to probabilistic bit number $j$, then gate $\langle t, g\rangle$ will be connected to the $j^{\text{th}}$ bit of $t$. (i.e., the new circuit sequence consists of identical copies of $C_n$, with particular choices of probabilistic bits hardwired in.)

Let $D_n$ consist of a MAJORITY gate with inputs from the various $C_{n,t}$. It is clear that the new circuit accepts the same language as $\{C_n\}$. The size of $D_n$ is $O(s(n)2^{(\log s(n))^c})$ which is subexponential. It is immediate that the other required properties also hold. ∎

The following lemma shows how one can in effect "push" an AND gate below a level of MOD gates (much as multiplication distributes over addition).

**Lemma 3.17** Let $\{C_{n,t}\}$ be a nice family of ACC sequences of subexponential size, having small fan-in AND gates, no OR gates, and no $\mathrm{MOD}_m$ gates where $m$ is composite, where the output gate of each circuit is an AND gate, and the inputs to that AND gate are $\mathrm{MOD}_p$ gates. Then there is an equivalent nice sequence $\{D_{n,t}\}$ with the same depth, also of subexponential size with small fan-in AND gates, no OR gates, and no $\mathrm{MOD}_m$ gates where $m$ is composite, where the output gate of each circuit is a $\mathrm{MOD}_p$ gate, and the inputs to that $\mathrm{MOD}_p$ gate are AND gates.

**Proof.**

Our proof again follows the outline given in [BT91], where we must be careful to see that the transformation can be done uniformly.

Suppose $G$ is an AND gate (the output gate of some $C_{n,t}$ that has $r$ $\mathrm{MOD}_p$ gates $G_1, G_2, \ldots, G_r$ as inputs). Note that $r$ is polylogarithmic in $s(n)$, where $s(n)$ bounds the size of $C_{n,t}$. Let the fan-in of $G_i$ be $n_i$ for all $i$, $1 \leq i \leq r$ and let $\{x_{ij}\}$, $1 \leq j \leq n_i$ denote the set of inputs to $G_i$. Finally, let $\boldsymbol{x}_i = \sum_{1 \leq j \leq n_i} x_{ij}$. Consider the AND of $G_1, G_2, \ldots, G_r$. By Fermat's Little Theorem, for $1 \leq i \leq r$,

$$1 - \boldsymbol{x}_i{}^{p-1} \equiv \begin{cases} 0 \pmod{p} & \text{if } \boldsymbol{x}_i \not\equiv 0 \pmod{p} \\ 1 \pmod{p} & \text{otherwise} \end{cases}$$

Therefore,

$$\bigwedge_{i=1}^{r} [\boldsymbol{x}_i \equiv 0 \pmod{p}] \iff 1 - \prod_{i=1}^{r} (1 - \boldsymbol{x}_i{}^{p-1}) \equiv 0 \pmod{p}$$

Note that $1 - \prod_{i=1}^{r} (1 - \boldsymbol{x}_i{}^{p-1})$ is a polynomial of degree $r(p-1)$ in the variables $x_{ij}$, $1 \leq i \leq r$, $1 \leq j \leq n_i$. Let $[r]$ denote the set $\{1, 2, \ldots, r\}$.

$$\begin{aligned} 1 - \prod_{i=1}^{r} (1 - \boldsymbol{x}_i{}^{p-1}) &= 1 - \Big(\prod_{i=1}^{r} \big(1 - \big(\sum_{j=1}^{n_i} x_{ij}\big)^{p-1}\big)\Big) \\ &= \sum_{k=1}^{r} \sum_{I \subseteq [r], |I|=k} (-1)^{k-1} \prod_{i \in I} \big(\sum_{j=1}^{n_i} x_{ij}\big)^{p-1} \end{aligned}$$

$$= \sum_{k=1}^{r}(-1)^{k-1} \sum_{I\subseteq[r],|I|=k} \prod_{i\in I} (\sum_{j=1}^{n_i} x_{ij})^{p-1}$$

$$= \sum_{k=1}^{r}(-1)^{k-1} \sum_{I\subseteq[r],|I|=k} \prod_{i\in I} \sum_{J_i=\langle j_{i,1},j_{i,2},\ldots,j_{i,p-1}\rangle\in[n_i]^{p-1}} \prod_{l=1}^{p-1} x_{ij_{i,l}}$$

$$= \sum_{k=1}^{r}(-1)^{k-1} \sum_{I\subseteq[r],I=\{i_1,i_2,\ldots,i_k\}} \sum_{J_{i_1},J_{i_2},\ldots,J_{i_k}} \prod_{s=1}^{k} \prod_{l=1}^{p-1} x_{i_s j_{i_s,l}} \quad (*)$$

This expression can be realized by a $\mathrm{MOD}_p$ gate (call it $g$) with AND gates of fan-in at most $r(p-1)$ as inputs. Since $r$ is $(\log s(n))^{O(1)}$, the fan-in of these AND gates is also polylogarithmic in $s(n)$. The only thing we need to take care of are the negative coefficients in the above expression. That is done by multiplying[4] the negative coefficients by $(1-p)$. The expression is changed slightly and the term $(-1)^{k-1}$ is replaced by $c_k$ where $c_k = 1$ if $k$ is even and $c_k = p-1$ if $k$ is odd. Now we interpret scalar multiplication as repeated addition and the multiplication of variables is realized by AND gates. The expression $(*)$ that we have derived above is in the most general form. For our situation, we can simplify things by noticing that all the $\mathrm{MOD}_p$ gates $G_1, G_2, \ldots, G_r$ have the same fan-in since the sequence $\{C_{n,t}\}$ is nice. If we let this fan-in be denoted by $n_0$, then it is not hard to see that the number of AND gates that are input to the new $\mathrm{MOD}_p$ gate $g$ is $\sum_{k=1}^{r} c_k \binom{r}{k} (n_0^{p-1})^k$. Note that since $r$ is polylogarithmic in $s(n)$, this expression (i.e., the fan-in of the new MOD gate) can be computed in time $(\log s(n))^{O(1)}$ from $\langle G, G_1, \ldots, G_r\rangle$.

To show that this step can be done uniformly, we must show how the new gates created in this step should be named so that the direct connection language of the new circuit family can be recognized within the required time bound. The name of the new $\mathrm{MOD}_p$ gate is $g = \langle G\#\langle G_1, G_2, \ldots, G_r\rangle, \mathrm{MOD}_p\rangle$. Looking at the expression $(*)$, it is clear that a typical AND gate has $k(p-1)$ inputs where $1 \le k \le r$. The $k(p-1)$ inputs can be divided up into $k$ groups of size $(p-1)$ each. Every group represents a distinct gate in the set $\{G_1, G_2, \ldots, G_r\}$. The $(p-1)$ inputs in a particular group (representing say $G_i$) are simply some of the input gates to $G_i$ (with repetitions allowed) in $C_{n,t}$. Depending on the value of $c_k$, such an AND gate either appears once or $(p-1)$ times.

[4]Note that this does not change the value of the expression mod $p$.

The name of such an AND gate is $\langle G\#\langle\langle H_1\#L_1\rangle,\langle H_2\#L_2\rangle,\ldots,\langle H_k\#L_k\rangle\rangle\#m,\mathrm{AND}\rangle$ where $H_1, H_2, \ldots, H_k$ are distinct gates from the set $\{G_1, G_2, \ldots, G_r\}$ and each $L_i$, $1 \leq i \leq k$ is a list of $(p-1)$ of the gates that are input to $H_i$ in the original circuit. Note that $L_i$ is allowed to have repetitions. The number $m$ is either 0 (indicating only one copy of the gate; this will be the case if $k$ is even) or between 1 and $(p-1)$ and is used for indexing the $(p-1)$ different copies.

We now show how to recognize the direct connection language for the new circuit family that we get after applying this transformation. Let $L_0$ be the direct connection language before the step and $L$ the one after the step. Note that the strings in $L$ conform to the naming scheme discussed above. The following cases must be considered:

1. Let $g$ be a new[5] gate. To check if $\langle n, t, g, g\rangle \in L$, we have the following two subcases:

   (a) $g$ is a $\mathrm{MOD}_p$ gate of the form $\langle G\#\langle G_1, G_2, \ldots, G_r\rangle, \mathrm{MOD}_p\rangle$. We do the following:

   - check that $G$ is the output gate of $C_{n,t}$. (This can be done because the circuits are well-named.)
   - check that $\langle n, t, G, G_i\rangle \in L_0$ for all $i$, $1 \leq i \leq r$, where $r$ is the fan-in of $G$. (Recall that $r$ can be computed from $G$, by one of the niceness properties.)

   (b) $g$ is an AND gate; $g = \langle G\#\langle\langle H_1\#L_1\rangle,\langle H_2\#L_2\rangle,\ldots,\langle H_k\#L_k\rangle\rangle\#m,\mathrm{AND}\rangle$. We do the following:

   - check that $G$ is the output gate of $C_{n,t}$.
   - verify that $H_1, H_2, \ldots, H_k$ are all distinct.
   - check that for all $i$, $1 \leq i \leq k$, $\langle n, t, G, H_i\rangle \in L_0$.
   - check that $m$ has the right value based on the parity of $k$.
   - For all $i$, $1 \leq i \leq k$, verify that $L_i$ is indeed a list of $(p-1)$ gates that are all input to $H_i$.

[5]The word "new" will hereafter be used to refer to gates that were created in the current step.

2. Let $g_1$ be an old gate and $g_2$ a new gate. Then $\langle n, t, g_1, g_2 \rangle \notin L$.

3. Let $g_1$ be a new gate and $g_2$ an old gate. The only way for $\langle n, t, g_1, g_2 \rangle \in L$ to hold is that $g_1$ is a new AND gate created in this step.
   Hence $g_1$ has the form $\langle G \# \langle \langle H_1 \# L_1 \rangle, \langle H_2 \# L_2 \rangle, \ldots, \langle H_k \# L_k \rangle \rangle \# m, \text{AND} \rangle$. We do the following to check if $\langle n, t, g_1, g_2 \rangle \in L$:

   - check that $\langle n, t, g_1, g_1 \rangle \in L$.
   - check that $\langle n, t, g_2, g_2 \rangle \in L_0$.
   - verify that $\exists\, i$, $1 \leq i \leq k$, such that $g_2$ belongs to the list of gates $L_i$.

4. Let $g_1$ and $g_2$ both be new gates. For $g_2$ to be an input to $g_1$, $g_1$ must be a new $\text{MOD}_p$ gate and $g_2$ a new AND gate. Let $g_1 = \langle G \# \langle G_1, G_2, \ldots, G_r \rangle, \text{MOD}_p \rangle$ and $g_2 = \langle G \# \langle \langle H_1 \# L_1 \rangle, \langle H_2 \# L_2 \rangle, \ldots, \langle H_k \# L_k \rangle \rangle \# m, \text{AND} \rangle$ where $\{H_1, \ldots, H_k\} \subseteq \{G_1, \ldots, G_r\}$. This is obviously easy to check.

The only remaining property that needs to be checked is the strong connection property for ANDs. However this is immediate using the naming system that we use, since the name of each new AND gate explicitly lists the names of each of its inputs.

Let us now consider the size of the new circuit after a single level of AND gates has been pushed below a level of MOD gates. The increase in size comes mainly because of all the new AND gates that get created. For a circuit of size $s$, the number of new AND gates created to change an AND of $r$ $\text{MOD}_p$ gates is $\leq \sum_{k=1}^{r} c_k \binom{r}{k} s^{(p-1)k} \leq (p-1)2^r s^{(p-1)r}$. Therefore, the overall size of the new circuit is at most $O(2^r s^{(p-1)r+1})$. Since $s$ is subexponential and $r$ is polylogarithmic in $s$, the size of the new circuits is still subexponential.

Note that this step does not preserve the tree structure of the circuit so we use Lemma 3.13 to produce an equivalent nice circuit sequence.

∎

**Lemma 3.18** Let $L$ be accepted by a uniform nice SYMACC circuit family $\{C_n\}$ of subexponential size, with small fan-in AND gates, no OR gates, and no $\text{MOD}_m$ gates

for composite $m$, such that each path from the output gate to an input passes through $k \geq 1$ MOD gates. Then there is an equivalent SYMACC circuit family $\{D_n\}$ satisfying the same conditions, such that each path from the output gate to an input gate passes through $k-1$ MOD gates.

**Proof.**

Our proof follows the outline in [BT91], using techniques developed in [Yao90, Tod91].

Let $L$ and $\{C_n\}$ be as in the statement of the lemma, where the output gate of $C_n$ computes the function $A(n, \eta)$, where $\eta$ is the number of elements of $\{C_{n,i} : 1 \leq i \leq f(n)\}$ that evaluate to 1. By Lemma 3.17, we may assume without loss of generality that the output of each circuit $C_{n,i}$ is a $\text{MOD}_p$ gate. Since $\{C_n\}$ is nice, for each $n$ there is some $n_0$ so that each $\text{MOD}_p$ gate in $C_{n,t}$ has fan-in $n_0$ (where $n_0$ can be computed in $(\log s(n))^{O(1)}$ time from $n$). For each $i \leq f(n)$, let the inputs to the $i^{\text{th}}$ of these $\text{MOD}_p$ gates be denoted by $x_{i,j}$, $1 \leq j \leq n_0$. Then the value of $\{C_n\}$ can be expressed as $A(n, \sum_{1 \leq i \leq f(n)} \text{MOD}_p(x_{i,1}, x_{i,2}, \ldots, x_{i,n_0}))$.

Let $k(n) = 1 + \lfloor \log_p f(n) \rfloor$ so that $p^{k(n)} > f(n)$. Note that $k(n)$ is computable in time $(\log s(n))^{O(1)}$. For the rest of this discussion, fix $n$, and let $k$ denote $k(n)$.

It is shown in [BT91] that the polynomial $P_k$ defined by

$$P_k(y) = (-1)^{k+1}(y-1)^k\left(\sum_{j=0}^{k-1} \binom{k+j-1}{j} y^j\right) + 1$$

satisfies the property that for every $m \geq 1$ and $y \geq 0$,

$$y \equiv 0 \pmod{m} \Rightarrow P_k(y) \equiv 0 \pmod{m^k}$$

and

$$y \equiv 1 \pmod{m} \Rightarrow P_k(y) \equiv 1 \pmod{m^k}$$

Let $Q_k(y) = 1 - P_k(y^{p-1})$. Then

$$Q_k(y) \equiv \begin{cases} 1 \pmod{p^k} & \text{if } y \equiv 0 \pmod{p} \\ 0 \pmod{p^k} & \text{otherwise.} \end{cases}$$

If $y = \sum_{i=1}^{r} y_i$ then

$$Q_k\left(\sum_{i=1}^{r} y_i\right) \equiv \text{MOD}_p(y_1, y_2, \ldots, y_r) \pmod{p^k}$$

Thus, recalling that the value of $C_n$ is $A(n, \sum_{1 \le i \le f(n)} \mathrm{MOD}_p(x_{i,1}, x_{i,2}, \ldots, x_{i,n_0}))$, we see that this can also be expressed as

$$A(n, \sum_{1 \le i \le f(n)} (Q_k(\sum_{1 \le j \le n_0} x_{i,j}) \pmod{p^k})).$$

Since $f(n) < p^k$ and $Q_k$ is always 0 or 1 (mod $p^k$), we can bring the outer sum inside the modulus to obtain the equivalent expression

$$A(n, (\sum_{1 \le i \le f(n)} Q_k(\sum_{1 \le j \le n_0} x_{i,j})) \pmod{p^k}).$$

Let $B(n, i)$ be defined to be $A(n, (i \pmod{p^k}))$. Thus the value of $\{C_n\}$ is equal to

$$B(n, \sum_{1 \le i \le f(n)} Q_k(\sum_{1 \le j \le n_0} x_{i,j})).$$

Note that $B$ is computable in time polylogarithmic in $s(n)$.

Note that $(\sum_{1 \le i \le f(n)} Q_k(\sum_{1 \le j \le n_0} x_{i,j}))$ is a low degree polynomial in the variables $\{x_{i,j}\}$. As in the proof of Lemma 3.17, our strategy will be to implement scalar multiplication with AND gates, and multiply the negative coefficients by $(1 - p^k)$ to make them positive[6], to obtain a realization of this polynomial in terms of circuits.

We first have to compute the coefficients of the polynomial $(\sum_{1 \le i \le f(n)} Q_k(\sum_{1 \le j \le n_0} x_{i,j}))$. Since this is just a sum of $f(n)$ similar polynomials, we can consider each of them separately.

Recall that $Q_k(y) = 1 - P_k(y^{p-1})$. Let $z = y^{p-1}$. After a little simplification we get $1 - P_k(z) = (1-z)^k(\sum_{j=0}^{k-1} \binom{k+j-1}{j} z^j)$. This is a polynomial of degree $2k-1$. For $i \ge 0$, let $b_i = 1$ if $i$ is even, and $b_i = p^k - 1$ if $i$ is odd. The coefficients of $z^m$, say $c_m$, are given by

$$c_m = \begin{cases} 1 & \text{if } m = 0. \\ 0 & \text{if } 1 \le m \le k-1. \\ \sum_{0 \le i \le k, 0 \le j \le k-1, i+j=m} b_i \binom{k}{i} \binom{k+j-1}{j} & \text{if } k \le m \le 2k-1. \end{cases}$$

These coefficients $c_m$ can be computed in $(\log s(n))^{O(1)}$ time, because we only need to compute $O(k)$ binomial coefficients each involving numbers that are $O(k \log k)$ bits

[6]This does not change the value of the expression mod $p^k$.

long. It can be verified that $O(k^4 \log k)$ time suffices which is polylogarithmic in the size of the circuit since $k$ is logarithmic in the size.

Now observe that the value of circuit $\{C_n\}$ is given by

$$\begin{aligned} B(n, \sum_{i=1}^{f(n)} Q_k(\sum_{j=1}^{n_0} x_{i,j})) &= B(n, \sum_{i=1}^{f(n)} 1 - P_k((\sum_{j=1}^{n_0} x_{i,j})^{p-1})) \\ &= B(n, \sum_{i=1}^{f(n)} \sum_{m=0}^{2k-1} c_m (\sum_{j=1}^{n_0} x_{i,j})^{(p-1)m}) \\ &= B(n, \sum_{i=1}^{f(n)} \sum_{m=0}^{2k-1} \sum_{c=1}^{c_m} \sum_{\langle j_1, j_2, \ldots, j_{p-1} \rangle \in [n_0]^{(p-1)m}} \bigwedge_{l=1}^{(p-1)m} x_{i,j_l}) \end{aligned}$$

In place of each circuit $C_{n,t}$ in the original sequence of circuits, there will be several new circuits, each of the form $D_{n,\langle i,m,c,j_1,\ldots,j_{(p-1)m}\rangle}$ where $0 \leq m \leq 2k-1, 1 \leq c \leq c_m$, and each $j_l$ is in $[n_0]$. (Of course, by our conventions, there will also be circuits $D_{n,t}$ where $t$ is not of this form; each such circuit $D_{n,t}$ will be a trivial rejecting circuit that will therefore have no effect on the output of the symmetric gate.)

The output gate of each circuit $D_{n,\langle i,m,c,j_1,\ldots,j_{(p-1)m}\rangle}$ will be an AND gate with the name $g = \langle n, i, m, c, j_1, \ldots, j_{(p-1)m}, \text{AND}\rangle$. The inputs to $g$ will be the $(p-1)m$ gates that are the $j_l^{\text{th}}$ inputs to the $\text{MOD}_p$ gate $G_i$ in the original circuit $C_{n,i}$. Note that since $C_{n,i}$ has the strong connection property, one can show that $D_{n,\langle i,m,c,j_1,\ldots,j_{(p-1)m}\rangle}$ does, too.

Note that the number of bits needed to write any such index $\langle i, m, c, j_1, \ldots, j_{(p-1)m}\rangle$ is bounded by $(\log s(n))^b$ for some constant $b$, and thus if we define $f'(n)$ to be equal to $2^{(\log s(n))^b}$, it follows that the symmetric gate computing $B$ in circuit $D_n$ has fan-in $f'(n)$, where $f'(n)$ is computable in time polylogarithmic in $s(n)$.

Since the new circuit consists of a subexponential number of circuits, each of which is of subexponential size, the new circuit is also of subexponential size.

The depth of the new circuit family is the same as $\{C_n\}$ but the top layer of $\text{MOD}_p$ gates has been "absorbed" into the symmetric gate computing $B$ and been replaced by a layer of AND gates of small fan-in. Now, by an appeal to Lemma 3.13, the circuit can be converted into nice form, which completes the proof. ∎

**Proof.**

(of Theorem 3.1) By Lemma 3.16, every language in ACC(*subexp*) is accepted by a deterministic SYMACC circuit family of subexponential size, with small fan-in AND gates, no OR gates, and no $\mathrm{MOD}_m$ gates for composite $m$. Successive applications of Lemma 3.17 and Lemma 3.18 remove all MOD gates from the circuit, while maintaining the property that all AND gates have small fan-in. This suffices to prove the theorem.

∎

# Chapter 4

# Immunity with respect to $AC^0$

In this chapter we investigate the notion of immunity with respect to $AC^0$. Recall that an infinite set $L$ is immune to a complexity class $\mathcal{C}$ if neither $L$ nor any of its infinite subsets are in $\mathcal{C}$. Our primary interest in initiating this study was to find sets that are almost-everywhere complex to $AC^0$. Since almost-everywhere complexity is well-known to be connected to immunity [BS85, GHS91, GK90, ABHH90], it is necessary to study sets that are immune to $AC^0$. In this chapter we focus on finding sets that are immune to $AC^0$.

It is easy to find sets with small space complexity that are immune to $AC^0$. Merely note that $AC^0$ is contained in Dspace($\log n$), and then use the almost-everywhere hierarchy theorem of [GHS91] to obtain the following result:

**Proposition 4.1** Let $S$ be a space-constructible function such that $\log n = o(S(n))$. Then there is a set in Dspace($S(n)$) that is immune to $AC^0$.

An essentially identical proof yields the following:

**Proposition 4.2** Let $T$ be a time-constructible function such that $n^k = o(T(n))$ for all $k$. Then there is a set in Dtime($T(n)$) that is immune to $AC^0$.

Neither of the preceding two propositions makes any use of any properties of $AC^0$ at all, other than the trivial observation that $AC^0$ is contained in Dspace($\log n$). On the other hand, the results that we present do rely on the special characteristics of constant depth circuits. In fact, the immunity result that follows hold also for the class ACC(*subexp*).

**Theorem 4.3** There is a set in $P^{PP}$ that is immune to ACC(*subexp*) (and hence is also immune to ACC and $AC^0$).

**Proof.**

The proof is direct consequence of Theorem 3.2. Using Theorem 3.2, we have that there is a set $Y \in \mathrm{PP}$ such that $\mathrm{ACC}(\mathit{subexp}) \subseteq \mathrm{Dtime}(n^2)^Y$. Now observe merely that the almost-everywhere hierarchy theorem of [GHS91] relativizes, so that there is a set in $\mathrm{Dtime}(n^3)^Y$ that is immune to $\mathrm{Dtime}(n^2)^Y$, and thus is also immune to $\mathrm{ACC}(\mathit{subexp})$. ∎

Even though Theorem 4.3 provides us the strongest immunity result that we can prove, we present a different proof to show that $\mathrm{P}^{\mathrm{PP}}$ is strongly separated from $\mathrm{AC}^0$ that is of independent interest. The proof is interesting because it follows very simply from Toda's proof [Tod91] that $\mathrm{PH} \subseteq \mathrm{P}^{\mathrm{PP}}$.

**Theorem 4.4** There is a set in $\mathrm{P}^{\mathrm{PP}}$ that is immune to $\mathrm{AC}^0$.

For the proof, we first need to introduce some notation. Let $X$ be a finite set of strings and $R$ a predicate over strings. We denote by $\mathrm{Prob}(\{w \in X : R(w)\})$ the probability that $R(w)$ is true for a randomly chosen $w$ from $X$ under uniform distribution. We now define a few operators that act on complexity classes and produce new ones. These operators are very similar to the ones defined by Toda [Tod91]. We then state and prove some lemmas about the complexity classes that we get using the operators.

**Definition 4.1** Let $t : \mathbf{N} \longrightarrow \mathbf{N}$, and let $\mathcal{C}$ be a class of sets. Then

- $\oplus_t \cdot \mathcal{C}$ is the class of languages $L$ for which there is some language $A \in \mathcal{C}$ such that $x \in L^{=n}$ iff $|\{y \in \{0,1\}^{t(n)} : \langle x, y\rangle \in A\}|$ is odd.
- $\mathrm{BP}_t \cdot \mathcal{C}$ is the class of languages $L$ for which there is some language $A \in \mathcal{C}$ such that $x \in L^{=n} \Rightarrow |\{y \in \{0,1\}^{t(n)} : \langle x, y\rangle \in A\}|/2^{t(n)} > 3/4$, and $x \notin L^{=n} \Rightarrow |\{y \in \{0,1\}^{t(n)} : \langle x, y\rangle \in A\}|/2^{t(n)} < 1/4$.
- $\mathrm{Pr}_t \cdot \mathcal{C}$ is the class of languages $L$ for which there is some language $A \in \mathcal{C}$ such that $x \in L^{=n} \iff |\{y \in \{0,1\}^{t(n)} : \langle x, y\rangle \in A\}|/2^{t(n)} > 1/2$.

For convenience, we shall suppress the subscripts whenever possible. For instance, $\mathrm{Pr}\cdot\oplus\cdot\mathrm{Dtime}(n^k)$ denotes $\mathrm{Pr}_{n^k}\cdot\oplus_{n^k}\cdot\mathrm{Dtime}(n^k)$. Note that for any function $t$ and any complexity class $\mathcal{C}$, $\mathrm{BP}_t\cdot\mathcal{C} \subseteq \mathrm{Pr}_t\cdot\mathcal{C}$.

The operators BP and Pr produce probabilistic analogs of a given complexity class $\mathcal{C}$ much in the way BPP and PP are probabilistic analogs of P. Applying the $\oplus$ operator on $\mathcal{C}$ produces a class that is related to $\mathcal{C}$ just like $\oplus$P is related to P. In particular, $L \in \oplus_t\cdot\mathcal{C}$ means that for every string $x \in L$ there is an odd number of witnesses of length $t(|x|)$, and for any given string of length $t(|x|)$ the procedure that verifies that the string is indeed a witness is in $\mathcal{C}$.

To prove Theorem 4.4, we first show that $\mathrm{AC}^0 \subseteq \mathrm{BP}\cdot\oplus\cdot\mathrm{Dtime}((\log n)^{O(1)})$. We then show that $\mathrm{Pr}\cdot\oplus\cdot\mathrm{Dtime}((\log n)^{O(1)})$ is contained in a small subclass of $\mathrm{P}^{\mathrm{PP}}$, such that $\mathrm{P}^{\mathrm{PP}}$ contains sets that are immune to this subclass.

**Lemma 4.5** $\mathrm{AC}^0 \subseteq \mathrm{BP}\cdot\oplus\cdot\mathrm{Dtime}((\log n)^{O(1)})$.

**Proof.**

It is shown in [AH90] that every language in uniform $\mathrm{AC}^0$ is accepted by a uniform family of depth two probabilistic circuits of size $2^{(\log n)^{O(1)}}$, consisting of a single PARITY gate on level two, and AND gates of fan-in $(\log n)^{O(1)}$ on level one, where the circuits use only $(\log n)^{O(1)}$ probabilistic bits. Although [AH90] claims only $\mathrm{Dspace}((\log n)^{O(1)})$ uniformity for these circuits, an examination of the proof of [AH90] shows that the circuits are uniform according to the notion of uniformity in Definition 2.13 as well.

To prove the lemma, we only need to show that every set accepted by a uniform family of depth two probabilistic circuits of the type described above is in $\mathrm{BP}\cdot\oplus\cdot\mathrm{Dtime}((\log n)^{O(1)})$.

Let $L$ be accepted by such a family of circuits $\{C_n\}$, and let $A$ be the set $\{\langle y, v, x\rangle : v$ is the name of an AND gate in $C_{|x|}$, and $v$ takes on the value 1 when $x$ is input to the circuit with probabilistic bits $y\}$. Since $x \in L$ iff for a majority of $y$ there are an odd number of $v$ such that $\langle y, v, x\rangle \in L$, and $A \in \mathrm{Dtime}((\log n)^{O(1)})$ because $\{C_n\}$ is uniform, it follows that $L \in \mathrm{BP}\cdot\oplus\cdot\mathrm{Dtime}((\log n)^{O(1)})$. ∎

**Lemma 4.6** Let $X$ be any set in $\oplus \cdot \mathrm{Dtime}(n^k)$ and let $q$ be any polynomial. Then, there exists an NP machine $M$ satisfying the following conditions. For each input $y$ of length $n$,

1. $\#\mathrm{acc}_M(y) \equiv -1 \pmod{2^{q(n)}}$ if $y \in X$,
2. $\#\mathrm{acc}_M(y) \equiv 0 \pmod{2^{q(n)}}$ if $y \notin X$, and
3. $M$ runs in time $(q(n))^2 \cdot n^k$.

**Proof.**

The proof follows directly from the proof of Lemma 4.2 in [Tod89]. ∎

**Lemma 4.7** There exists $Y \in \mathrm{PP}$ such that $\mathrm{Pr} \cdot \oplus \cdot \mathrm{Dtime}(n^k) \subseteq \mathrm{Dtime}(n^{2k^2+4k})^Y$ for every $k \geq 1$.

**Proof.**

(of Theorem 4.4) Lemmas 4.5 and 4.7 imply that $\mathrm{AC}^0$ is contained in $\mathrm{Dtime}(n^6)^Y$ for some $Y \in \mathrm{PP}$ (letting $k = 1$). By the almost-everywhere hierarchy theorem of [GHS91] there is a set in $\mathrm{Dtime}(n^7)^Y$ that is immune to $\mathrm{Dtime}(n^6)^Y$ and hence to $\mathrm{AC}^0$. ∎

**Proof.**

(of Lemma 4.7) Let $L \in \mathrm{Pr} \cdot \oplus \cdot \mathrm{Dtime}(n^k)$. Then there exists a set $X \in \oplus \cdot \mathrm{Dtime}(n^k)$ such that

For each $x$,

$$\mathrm{Prob}(\{w \in \{0,1\}^{|x|^k} : x\#w \in X \text{ iff } x \in L\}) > \tfrac{1}{2}$$

$$(\text{i.e., } x \in L \iff |\{w \in \{0,1\}^{|x|^k} : x\#w \in X\}| > 2^{|x|^k - 1}).$$

Let $q$ be a polynomial satisfying $q(n + 1 + n^k) > n^k$ for each $n \geq 0$. Note that the polynomial $q(m) = m$ has the above mentioned property and so we use $q(m) = m$ hereafter. By Lemma 4.6, there exists an NP machine $M$ satisfying the following conditions. For each input $y$ of length $n$,

1. $\#\mathrm{acc}_M(y) \equiv -1 \pmod{2^n}$ if $y \in X$,
2. $\#\mathrm{acc}_M(y) \equiv 0 \pmod{2^n}$ if $y \notin X$, and

3. $M$ runs in time $n^{k+2}$.

Without loss of generality assume that each computation path of $M$ on an input of length $n$ is of length exactly $n^{k+2}$.

Let $t(n) = n^{k+2}$. Let $M_1, M_2, \ldots$ be an enumeration of nondeterministic Turing machines, each running in time $t(n)$. Let $\#$ be a special pairing symbol. For each $i$, define the function $h_i : \{0,1\}^* \to \mathbf{N}$ as follows:

$$h_i(x) = |\{wv \in \{0,1\}^* : \quad |w| = |x|^k, |v| = t(|x\#w|) \text{ and } v \text{ is an accepting computation of } M_i \text{ on input } x\#w\}|.$$

Define the set $Y$ to be $\{\langle i, x, l\rangle : x \in \{0,1\}^* \text{ and } h_i(x) \leq l\}$. Toda [Tod91] has observed that $Y$ is in PP (for essentially the same reason that the set $Y$ in the proof of Theorem 3.2 is in PP).

As before, for any $L$ in $\Pr\cdot\oplus\cdot\mathrm{Dtime}(n^k)$, there exists a set $X \in \oplus\cdot\mathrm{Dtime}(n^k)$ and a machine $M_i$ such that

1. $\#\mathrm{acc}_{M_i}(y) \equiv -1 \pmod{2^n}$ if $y \in X$,

2. $\#\mathrm{acc}_{M_i}(y) \equiv 0 \pmod{2^n}$ if $y \notin X$, and

3. $M_i$ runs in time $t(n)$.

Now define the function $g$ such that for each $x$, $g(x) = |\{w \in \{0,1\}^{|x|^k} : x\#w \in X\}|$.

Let $x$ be any string of length $n$. By definition, $x \in L \iff g(x) > 2^{n^k - 1}$. Toda has proved in [Tod91] that $h_i(x) \equiv -g(x) \pmod{2^{(n+1+n^k)}}$.

It is easy to see that

- $g(x) \leq 2^{n^k}$ and
- $h_i(x) \leq 2^{n^k} \cdot 2^{t(n+1+n^k)} = 2^{n^k} \cdot 2^{n^{k(k+2)}} = 2^{n^k} \cdot 2^{n^{k^2+2k}} = 2^{O(n^{k^2+2k})}$ (since $k \geq 1$).

Consider an oracle machine $N$ that has access to the oracle $Y$ and behaves as follows: On input $x$ ($|x| = n$),

- $N$ computes the value of $h_i(x)$ using binary search on the set $Y$. $N$ needs to ask at most $O(n^{k^2+2k})$ questions of about the same size. This step requires time $O(n^{2k^2+4k})$.

- $N$ computes the value of $g(x)$ using the fact[1] that $h_i(x) \equiv -g(x) \pmod{2^{(n+1+n^k)}}$. This step can be done in time $(O(n^{k^2+2k}) + O(n^k))^2$ which is $O(n^{2k^2+4k})$.

- After computing $g(x)$, $N$ decides whether $g(x) > 2^{n^k-1}$. This step requires time $O(n^k)$.

Therefore, $N$ runs in time $O(n^{2k^2+4k})$, and thus $L \in \text{Dtime}(n^{2k^2+4k})^Y$. ∎

[1]Note that the computation $a \pmod{b}$ where $|a| = m$ and $|b| = n$ can easily be done in time $O(m+n)^2$.

# Chapter 5

# NP and Immunity with respect to $AC^0$

In the previous chapter we proved that $P^{PP}$ contains sets that are immune to $AC^0$ (Theorem 4.4). This result is not so surprising because $AC^0$ is a tiny complexity class, unable to even compute the parity of an input string, whereas $P^{PP}$ is a very powerful complexity class, containing the entire polynomial hierarchy. However, the result is fairly interesting and establishes a big gap between $AC^0$ and $P^{PP}$, as is widely believed. The downside of this result is that since $AC^0 \subsetneq NC^1$, it would seem reasonable to hope to be able to present sets in $NC^1$, or at least in P (classes believed to be far less powerful than $P^{PP}$), that are immune to $AC^0$.

However, the results of this chapter show that even for the fairly "large" complexity class NP, the question of whether there are sets in that class that are immune to $AC^0$ cannot be answered without resolving some long-standing issues in complexity theory. In fact, it is still unknown if PP or even $PP^{PH}$ has sets that are immune to $AC^0$. As far as finding sets in NP that are immune to $AC^0$ is concerned, we are able to prove the following:

**Theorem 5.1** If there is a set in E that is immune to $\bigcup_k \Sigma_k\text{Time}(n)$ then there is a set in P that is immune to $AC^0$.

**Theorem 5.2** If $E = \bigcup_k \Sigma_k\text{Time}(n)$ and all NE-predicates are E-solvable, then no set in NP is immune to $AC^0$.

Recall that $\bigcup_k \Sigma_k\text{Time}(n)$ denotes the rudimentary sets (Definition 2.6). To prove Theorem 5.1 we need Proposition 5.3 that draws some elementary connections between $AC^0$ and RUD. We would like to remind the reader that all the languages considered in this thesis are subsets of $\{0,1\}^*$. We assume a standard mapping from $\{0,1\}^*$onto the

positive integers; namely the string $x$ will denote the integer whose binary representation is $1x$. We will use strings and the integers they denote interchangeably. This leads us to the following definition:

**Definition 5.1** For a language $L$, the *unary version* of $L$, denoted $un(L)$ is defined as $un(L) = \{0^n : n \in L\}$.

**Proposition 5.3** $L \in \bigcup_k \Sigma_k\text{Time}(n) \iff un(L) \in \text{AC}^0$.

**Proof.**

This result is proved using elementary translational (i.e, "padding") techniques. The proof is somewhat easier if one observes (as is done in [Sip83]) that, if $L$ is accepted by an alternating Turing machine $M$ in time $O(\log n)$ with $O(1)$ alternations, then $L$ is accepted by a similar machine that, along any computation path, uses its address tape to access the input only once.

($\Rightarrow$) Let $L \in \bigcup_k \Sigma_k\text{Time}(n)$. Therefore, there exists an alternating Turing machine $M$ and some $k$ such that $M$ accepts $L$ in linear time and makes $k$ alternations. Consider the machine $M'$ that behaves as follows:

On input $0^n$,

Compute $n$ (in binary)

Check that $M$ accepts $n$ (by simulating $M$ on $n$)

end

It is obvious that $M'$ accepts $un(L)$ in time $O(\log n)$ and makes at most $k$ alternations. Therefore, $un(L) \in \bigcup_k \Sigma_k\text{Time}(\log n)$. Since $\text{AC}^0 = \bigcup_k \Sigma_k\text{Time}(\log n)$ by the result of [BIS90], $un(L) \in \text{AC}^0$.

($\Leftarrow$) Suppose there exists an alternating Turing machine $M$ that accepts $un(L)$ in $O(\log n)$ time and makes a constant number of alternations. Consider the machine $M'$ that behaves as follows:

On input $n$, $M'$ starts simulating $M$. If, at some point during this simulation, $M$ queries the input via its input address tape, $M'$ compares the number $m$ stored on the address tape of $M$ with the number $n$, and continues the simulation of $M$ either from

the state saying "the $m^{\text{th}}$ input symbol is 0" (if $m \leq n$), or from the state saying "the input has length shorter than $m$" (if $m > n$); thus $M'$ on input $n$ effectively simulates $M$ on input $0^n$. Also, since we are assuming without loss of generality that $M$ uses its random access only once along any computation path, the running time of $M$ is $O(\log n)$, which is $O(|n|)$. Thus $M'$ accepts $L$ in linear time and makes a constant number of alternations. ∎

**Proof.**

(of Theorem 5.1) If $L$ is in E and is immune to $\bigcup_k \Sigma_k\text{Time}(n)$, then $un(L)$ is in P and is immune to $\text{AC}^0$. ∎

**Corollary 5.4** If NP is not strongly separated from $\text{AC}^0$(i.e., no set in NP is immune to $\text{AC}^0$) then P $\neq$ NP.

Though the above statement sounds counterintuitive, we can infer that if NP is not large enough to have sets immune with respect to $\text{AC}^0$, then this somehow puts a severe restriction on the power of P.

**Proof.**

(of Corollary 5.4) It is a well known result that P = NP $\Rightarrow$ P = PH. Therefore, P = NP $\Rightarrow \bigcup_k \Sigma_k\text{Time}(n) \subseteq$ P. The hierarchy theorem in [HS65] implies that E has a set that is immune to P, therefore E has a set that is immune to $\bigcup_k \Sigma_k\text{Time}(n)$. By Theorem 5.1 this implies that P has a set that is immune to $\text{AC}^0$ which means that NP is strongly separated from $\text{AC}^0$. ∎

To prove Theorem 5.2 we need the following lemmas:

**Lemma 5.5** If all NE-predicates are E-solvable, then no set in NP is immune to P-uniform $\text{AC}^0$.

**Proof.**

By Lemma 2.2, if all NE-predicates are E-solvable then every infinite set in P has an infinite P-printable subset. By Lemma 2.1, this implies that every infinite set in NP has an infinite set in P-uniform $\text{AC}^0$. ∎

Lemma 5.5 does not say anything about dlogtime-uniform $AC^0$, speaking instead only of P-uniform $AC^0$. The next lemma shows under what conditions these classes coincide.

**Lemma 5.6** $E = \bigcup_k \Sigma_k \text{Time}(n) \iff$ P-uniform $AC^0 =$ dlogtime-uniform $AC^0$.

**Proof.**

($\Leftarrow$) Suppose P-uniform $AC^0$ = dlogtime-uniform $AC^0$. It suffices to show that E $\subseteq \bigcup_k \Sigma_k \text{Time}(n)$. Let $L \in$ E. Note that $un(L)$ is a tally set in P. Any tally set in P is trivially in P-uniform $AC^0$. Using the hypothesis, $un(L) \in$ dlogtime-uniform $AC^0$. Hence, by Proposition 5.3, $L \in \bigcup_k \Sigma_k \text{Time}(n)$.

($\Rightarrow$) To prove this direction, we first need the simple fact that for all $k$, the language

$$L_k = \{\langle x, C\rangle : C \text{ is a depth } k \text{ circuit of AND and OR gates that evaluates to 1 on input } x\}$$

is in $AC^0$. To see this, note that $\langle x, C\rangle \in L_2$ iff

($\exists i$) bit position $i$ in $C$ is the start of the name of the output gate $g$ and

[$g$ is an AND gate and

($\forall j$)

[(if $j$ is the start of the name of an AND gate $h$ connected to $g$, then

($\forall k$) (if (the negation of) bit $k$ of $x$ is an input to $h$,

then bit $k$ is 1 (0)))

and (if $j$ is the start of the name of an OR gate $h$ connected to $g$, then

($\exists k$) ((the negation of) bit $k$ of $x$ is an input to $h$

and bit $k$ is set to 1 (0)))]]

or [$g$ is an OR gate and

($\exists j$)

[($j$ is the start of the name of an AND gate $h$ connected to $g$, and

($\forall k$) (if (the negation of) input bit $k$ is an input to $h$,

then bit $k$ is set to 1 (0)))

or ($j$ is the start of the name of an OR gate $h$ connected to $g$, and

$(\exists k)$ ((the negation of) bit $k$ of $x$ is an input to $h$
and bit $k$ is set to 1 (0)))]].

Using the characterization of $AC^0$ in terms of first-order logic as presented by [BIS90], it is clear that $L_2$ is in $AC^0$. It is easy to see how to generalize this to show that each $L_k$ is in $AC^0$.

To proceed, assume that $E = \bigcup_k \Sigma_k Time(n)$, and let $L$ be a language in P-uniform $AC^0$. Thus there is a family of depth $k$ circuits $\{C_n\}$ recognizing $L$, such that the set $A = \{0^{\langle n,i \rangle} : \text{bit } i \text{ of the description of } C_n \text{ is } 1\}$ is in P. Let $B = \{r : 0^r \in A\}$. It is easy to see that $B$ is in E. Under the assumption $E = \bigcup_k \Sigma_k Time(n)$, $B \in \bigcup_k \Sigma_k Time(n)$, and since $A = un(B)$, from Proposition 5.3 it follows that $A$ is in dlogtime-uniform $AC^0$.

An alternating log-time Turing machine for $L$ can now be described. On input $x$, simulate the $AC^0$ algorithm for $L_k$ on input $\langle x, C_n \rangle$. That is, when the algorithm for $L_k$ uses its address tape to look at input position $m$, look at the $m^{\text{th}}$ bit of $x$ if $m \leq |x|$, and otherwise test if $0^{\langle n, m-n \rangle} \in A$. ∎

**Proof.**

(of Theorem 5.2) Immediate from Lemmas 5.5 and 5.6. ∎

As we have stated before, we conjecture that the hypothesis to Theorem 5.2 is false. Assuming that $E = \bigcup_k \Sigma_k Time(n)$ puts very low limits on the power of E and also implies that PH collapses. At the same time, assuming that E is powerful enough to solve every NE-predicate seems contradictory. However, in the next chapter we show that this hypothesis cannot be shown to be false by any proof technique that relativizes.

# Chapter 6

## Some relativization results

In the previous chapter we showed that the issue of strong separation between NP and $AC^0$ is linked to some long-standing open questions in complexity theory. If it can be shown that NP is not strongly separated from $AC^0$ then that gives us a non-relativizing proof technique to show that P $\neq$ NP (Theorem 5.1). On the other hand, if NP has sets that are immune to $AC^0$ then either E is different from RUD or there are NE-predicates that are E-solvable (Theorem 5.2). However, in this chapter we provide evidence that the standard techniques of simulation and diagonalization will not be able to answer whether or not NP is strongly separated from $AC^0$. We demonstrate the existence of oracles relative to which the hypotheses of Theorems 5.1 and 5.2 hold.

We should emphasize that the results in this chapter do not indicate that the problem of strongly separating NP from $AC^0$ is hard. We hope that the new proof techniques developed in [FKLN90], [Sha90] and [BFL91] can be used to answer this question. We think that our oracle results can point to some of the problems that would have to be overcome in answering this question in the unrelativized setting.

**Proposition 6.1** There is an oracle relative to which E has a set that is immune to $\bigcup_k \Sigma_k \mathrm{Time}(n)$.

**Proof.**

If $A$ is any oracle relative to which P = NP, then relative to $A$, $\bigcup_k \Sigma_k \mathrm{Time}(n) \subseteq$ P = PH. Since the almost-everywhere hierarchy theorem of [GHS91] relativizes, there is a set in E that is immune to P relative to any oracle. Therefore, relative to $A$, E has a set that is immune to $\bigcup_k \Sigma_k \mathrm{Time}(n)$. ∎

**Theorem 6.2** There is an oracle relative to which E = $\Sigma_2$Time($n$) and every NE-predicate is E-solvable.

We would like to remind the reader that we have only one reason for being interested in this unusual combination of conditions on the complexity of exponential time: it implies that no set in NP is immune to $AC^0$. Hence, finding sets in P or in NP that are immune to $AC^0$ will necessarily involve the development of non-relativizing proof techniques for attacking questions concerning deterministic and nondeterministic time classes. To prove the above theorem, we need the following:

**Definition 6.1** Let $\langle i, x \rangle$ denote a standard one-one pairing function mapping $\mathbf{N} \times \Sigma^*$ into $\mathbf{N}$ (e.g., let $\langle i, x \rangle$ denote the number whose binary representation is the ASCII code of the string "$i, x$", where $i$ is written in binary). Let $M_1, M_2, \ldots$ be an enumeration of nondeterministic exponential-time oracle Turing machines; note that without loss of generality we may assume that for every NE-predicate there is a machine $M_i$ realizing the predicate, such that on every input $x$, the computations of $M_i$ on input $x$ have length exactly $\langle i, x \rangle$. For any set $A$, define $S(A)$ to be the set:
$\{y : |y| = \langle i, x \rangle$ and $y$ encodes an accepting path of the NE machine $M_i^A$ on input $x\}$.

We have already seen that questions about the difficulty of NE-predicates can actually be expressed as questions about the time-bounded Kolmogorov complexity of sets in P (Proposition 2.3). In our oracle construction, we replace the condition "every NE-predicate is E-solvable" by an equivalent condition about the time-bounded Kolmogorov complexity of certain sets in P. Recall that the time-bounded Kolmogorov complexity of a string $x$ is defined as $Kt(x) = \min\{|y| + \log t : M_u(y) = x$ in at most $t$ steps$\}$ where $M_u$ is a "universal" Turing machine. Also recall that for a set $L \subseteq \{0,1\}^*$, $K_L(n) = \min\{Kt(x) : x \in L^{=n}\}$ if $L^{=n}$ is nonempty; undefined otherwise. The following proposition is an extension of Proposition 2.3.

**Proposition 6.3** For any oracle $A$, the following are equivalent:

(a) Every $NE^A$-predicate is $E^A$-solvable.

(b) For every set $L$ in $P^A$, $K_L^A(n) = O(\log n)$.

(c) $K^A_{S(A)}(n) = O(\log n)$.

**Proof.**

By Proposition 2.3, (a) $\iff$ (b) and the result holds relative to any oracle. Since $S(A)$ is clearly in $\mathrm{P}^A$, it follows that (b) $\Rightarrow$ (c). In order to show that (c) $\Rightarrow$ (a), suppose $K^A_{S(A)}(n) \leq c \log n$ for some constant $c$. To solve the $\mathrm{NE}^A$-predicate defined by the $\mathrm{NE}^A$ machine $M_i^A$, we can use the following algorithm:

On input $x$, let $m = \langle i, x \rangle$. Since $K^A_{S(A)}(m) \leq c \log m$, the universal Turing machine $M$ will produce a string of $S(A)$ of length $m$ (if one exists) on some input of length $\leq c \log m$ with oracle $A$. Try running $M^A$ on all possible inputs of length $\leq c \log m$ to see if a string $y$ of length $m$ is ever produced. If so, check if $y$ encodes an accepting path of $M_i^A$ on $x$. It is rather easy to see that the above computation requires $2^{O(|x|)}$ time. ∎

**Proof.**

(of Theorem 6.2) For the proof it suffices to build an oracle $A$ such that $K^A_{S(A)}(n) = O(\log n)$ and $\mathrm{E}^A = \Sigma_2\mathrm{Time}^A(n)$.

Recall that for every $A$, $S(A) = \{y : |y| = \langle i, x \rangle$ and $y$ encodes an accepting computation of the $\mathrm{NE}^A$ machine $M_i^A$ on input $x\}$. It is fairly obvious that there is a deterministic oracle machine $M_S^A$ running in time $n^2$ such that $M_S^A$ accepts $S(A)$.

For every $A$, let $L(A) = \{i\#x10^k :$ the $i^{\text{th}}$ $\mathrm{E}^A$ machine $M_i^A$ accepts $x$ in $\leq 2^k$ steps$\}$. It is easy to see that there is a deterministic oracle machine $M_L^A$ running in time $2^{2n}$ such that $M_L^A$ accepts $L(A)$. It is also obvious that $L(A)$ is complete for $\mathrm{E}^A$ under linear time reductions.

Thus to prove the theorem, it suffices to build an oracle $A$ such that $K^A_{S(A)}(n) = O(\log n)$ and $L(A) \in \Sigma_2\mathrm{Time}^A(n)$. To obtain such an $A$, two kinds of encoding need to be performed. The encoding done to ensure $\mathrm{E}^A = \Sigma_2\mathrm{Time}^A(n)$ will henceforth be called $L$-encoding and the encoding done to ensure $K^A_{S(A)}(n) = O(\log n)$ will be referred to as $S$-encoding.

The oracle will be constructed in stages. Within each stage we have a substage in which we do $L$-encoding followed by a substage in which we do $S$-encoding. These

encodings reserve certain strings for $A$ and $\overline{A}$. Note that once a string has been reserved for $A$ or $\overline{A}$, its fate does not change later. Let $A_i$ and $\overline{A_i}$ denote the sets of strings reserved for the oracle and its complement respectively after the completion of stage $i$. Define $A = \cup_i A_i$. We will also use $A_i'$ and $A_i''$ to denote the oracle at certain points while stage $i$ is in progress.

In a particular $L$-encoding substage, we ensure that

$$(\forall x)\,(x \in L(A) \iff (\exists y)(\forall z)\,1xyz \in A) \text{ where } |y| = |z| = 5|x|.$$

This is done for all $x$ of a particular length under consideration in that substage. The above condition ensures that $L(A) \in \Sigma_2\text{Time}^A(n)$. To test for membership of a string $x$ in $L(A)$, a $\Sigma_2\text{Time}^A(n)$ machine existentially guesses $y$, universally guesses $z$ (both of length $5|x|$) and then asks whether the string $1xyz$ belongs to $A$.

In the $S$-encoding substage, we try to ensure that $K^A_{S(A)}(m) = O(\log m)$ for the length $m$ currently under consideration by encoding a string $w \in S(A)$ by the strings $0^{m^2+1}, 0^{m^2+2}, 0^{m^2+3}, \ldots, 0^{m^2+m}$. The way this encoding works is that $(\forall j)\,1 \leq j \leq m$, the answer to the question "$0^{m^2+j} \in A$?" specifies the $j^{\text{th}}$ bit of $w$. Note that any string $w$ of length $m$ encoded in this way can be reconstructed very quickly from the description "$m$" (of length logarithmic in $|w|$) and thus $Kt^A(w) = O(\log |w|)$.

Let $A_0 = \emptyset$, and assume that no strings have been reserved so far. The construction is done as follows:

During stage $i$,

do the $L$-encoding for length $i$;

do the $S$-encoding for lengths $m$ where $2^i \leq m < 2^{i+1}$

end of construction.

We will now describe a typical stage (say $i^{\text{th}}$) of the construction process in detail. This is done in two parts:

**$L$-encoding substage**

During this substage all strings of length $i$ are processed. Let $A_i'$ denote the oracle that has been constructed up to a certain point in the current $L$-encoding substage, and let $x$ be the next string under consideration. Run $M_L^{A_i'}$ on input $x$, and reserve all of the strings that were queried on that computation for $A_i$ or $\overline{A_i}$, according to whether or

not they are in $A_i'$. If $M_L^{A_i'}$ accepts, find some $y$ of length $5|x|$ such that no string of the form $1xyz$ where $|z| = 5|x|$ is reserved for $\overline{A_i}$, and put all of the $2^{5|x|}$ strings of the form $1xyz$ into $A_i$. On the other hand, if $M_L^{A_i'}$ rejects, then for each $y$ of length $5|x|$ find some $z$ with $|z| = |y|$ and reserve $1xyz$ for $\overline{A_i}$.

**$S$-encoding substage**

During this substage all strings of length $m$ where $2^i \leq m < 2^{i+1}$ are processed. Let $A_i'$ denote the oracle that has been constructed thus far in the current stage (i.e., after the completion of the $L$-encoding substage and part of the $S$-encoding substage that has been completed so far). Let $m$ denote the length of the strings being considered. There are two possibilities:

1. $M_S^{A_i'}$ accepts a string $w$ of length $m$.
2. $M_S^{A_i'}$ does not accept any string of length $m$.

In the first case, we simply go ahead and encode one such $w$ as described above. In the second case, try to determine if there is a possible extension[1] of $A_i'$, say $A_i''$ (this extension might require us to reserve certain strings for $\overline{A_i}$ as well), so that $M_S^{A_i''}$ accepts some string $w$ of length $m$. There are two possibilities again:

- $M_S^{A_i''}$ does not accept any string of length $m$ for any possible extension $A_i''$; in this case, no encoding is required.
- $M_S^{A_i''}$ does accept a string $w$ of length $m$ for some extension $A_i''$; in this case, extend $A_i'$ to $A_i''$ (i.e., add the strings in $A_i'' - A_i'$ to $A_i'$ and possibly reserve certain strings for $\overline{A_i}$) and proceed with the encoding of $w$. (Note that $A_i''$ need differ from $A_i'$ on at most $m^2$ strings, since only those strings queried by $M_S$ on input $w$ need be reserved.)

**Proof of correctness of the construction**

Consider stage $i$ of the construction:

---

[1] Note that any extension that is considered is not allowed to change the status of any string that has already been reserved for $A_i$ or $\overline{A_i}$.

• In the $L$-encoding substage we want that

for all $x$ of length $i$, $x \in L(A) \iff (\exists y)(\forall z)\, 1xyz \in A$ where $|y| = |z| = 5i$

Consider the encoding for a typical $x$ of length $i$. We need to show that there are enough unreserved strings beginning with $1x$ to enable us to do the required encoding. The number of strings that have already been considered for $L$-encoding is $< (2 + 2^2 + \cdots + 2^i) < 2^{i+1}$. For each of these strings, it is possible that $M_L^A$ might have queried the oracle. But since $M_L^A$ runs in time $2^{2n}$, the maximum number of strings that could have been reserved due to queries made while processing those strings is $< (2^{i+1} - 1) \cdot 2^{2i} < 2^{3i+1}$.

The $L$-encoding substage also reserves strings for the oracle that are of the form $1uvw$. But since we are only interested in counting strings of the form $1xyz$ for the string $x$ currently under consideration, these strings can be ignored.

Note that the $S$-encoding has been done for lengths $\leq 2^i - 1$. The machine $M_S^A$ could possibly have queried strings for all lengths $\leq 2^i - 1$. Since $M_S^A$ runs in time $n^2$, the maximum number of strings that could have been reserved due to queries made while doing that $S$-encoding is $< \sum_{m=1}^{2^i-1} m^2 < 2^{3i}$. In each $S$-encoding substage, the strings that are specially reserved to encode strings always start with 0 so they can be ignored too. Therefore, the total number of strings with 1 in the first position that could have been queried or reserved so far is $< 2^{3i+1} + 2^{3i} < 2^{5i}$, $\forall\, i \geq 1$. Since less than $2^{5i}$ strings have been reserved we can find a string $y$ of length $5i$ such that for all $z$ of length $5i$, none of $1xyz$ has been reserved. Similarly, for each string $y$ of length $5i$ there is some $z$ of length $5i$ such that $1xyz$ is not reserved. Thus it is possible to find the unreserved strings that are required in order to carry out the $L$-encoding.

• Now consider the $S$-encoding substage. During stage $i$ we do the $S$-encoding for each length $m$ such that $2^i \leq m < 2^{i+1}$. Encoding is necessary only if there is some $w$ of such a length $m$ such that $M_S$ accepts $w$ with the oracle constructed so far.

Since $|w| = m$, it should be encoded by strings $0^{m^2+j}$ where $1 \leq j \leq m$. But we must make sure that none of these strings has been reserved as yet.

Since $i \leq \log m$, the $L$-encoding has only been done for lengths $\leq \log m$. Therefore

the longest string with a 0 in the first position that could have been queried during any of the $L$-encoding substages has length $\leq 2^{2\log m} = m^2$. The strings of the form $1xyz$ that are reserved during the $L$-encoding substages start with a 1 in their first positions so they do not interfere with the encoding in the current substage. $S$-encoding has been done for lengths $\leq m-1$, so the longest strings queried or reserved have length $\leq (m-1)^2 + (m-1) < m^2$ whenever $m \geq 1$.

During the current substage, to make $M_S^{A_i'}$ accept an input of length $m$ (by possibly extending the oracle), the longest string that could have been queried and reserved has length $\leq m^2$ (because the machine $M_S^{A_i'}$ runs in time $n^2$). Therefore, any string queried so far has length $\leq m^2$. Hence we can encode $w$ with $0^{m^2+j}$ $1 \leq j \leq m$.

This completes the proof of correctness of the construction and we have an oracle $A = \cup_i A_i$ such that $\mathrm{E}^A = \Sigma_2\mathrm{Time}^A(n)$ and $K^A_{S(A)}(n) = O(\log n)$. ∎

# Chapter 7

# A new characterization of $AC^0$

In [Jon75], Jones introduced logspace reductions to study the relative complexity of problems in P (see [SM73] for independent work on logspace reductions). In the same paper, he also introduced a restricted version of logspace reductions called log-bounded rudimentary reductions and went on to demonstrate a lot of complete problems for various complexity classes under these reductions. Logspace reducibility has proved to be a very useful notion in complexity theory in the intervening years, whereas log-bounded rudimentary reducibility has been mentioned explicitly only seldom in subsequent work – although, as we demonstrate, it has been considered implicitly many times, under different names. In this chapter we show that the log-bounded rudimentary reductions provide yet another characterization of dlogtime-uniform $AC^0$.

## 7.1 The Characterization Theorem

Recall the definition of the class of predicates $\mathrm{RUD}_S$ where $S$ is a space bound (Definition 2.22). In Chapter 2 we mentioned that $\mathrm{RUD}_S$ can also be viewed as a class of languages. In this chapter we show that $\mathrm{RUD}_{\log} = \mathrm{AC}^0$. For the proof we use the first order framework discussed in Chapter 2. We use the well-known fact that the composition of two functions in $\mathrm{AC}^0$ is[1] also in $\mathrm{AC}^0$ and we work with alternating Turing machines running in sublinear time. Let $\Sigma_k\mathrm{time}(\log n)$ denote the class of languages accepted by alternating Turing machines running in $O(\log n)$ time, beginning in an existential configuration, and making at most $k-1$ alternations between existential and universal configurations on any computation path. Let LH (the logtime hierarchy of [Sip83]) denote $\bigcup_k \Sigma_k\mathrm{time}(\log n)$.

[1] A circuit can easily compute a function by having multiple output gates.

**Theorem 7.1** $\mathrm{AC}^0 = \mathrm{RUD}_{\log}$.

**Proof.**

($\supseteq$) This containment is mentioned in Proposition 5 of [Vol83] (where the converse containment is left as an open question). We present a complete proof here.

The proof proceeds by induction on the definition of $\mathrm{RUD}_{\log}$, showing that each language in $\mathrm{RUD}_{\log}$ is in LH. The input to an ATM is assumed to have both ends marked by the special symbol \$.

Consider the inductive definition of $\mathrm{RUD}_{\log}$ :

1. $\sigma(x, i, a)$ and $\neg\sigma(x, i, a)$ are in $\mathrm{RUD}_{\log}$.

   Consider an ATM $M$ that behaves as follows:

   On input $\$x, i, a\$$,

   - $M$ existentially guesses the positions of all the commas (there is only a constant number of them).
   - $M$ universally verifies that that the guessed positions contain the appropriate markers and the other positions contain 0's or 1's.
   - Using the marker positions, $M$ calculates $|x|$ and $|i|$.
   - $M$ existentially guesses the bits of $i$ and writes them on a worktape, and then universally checks that for each $j \leq i$, the guessed value for position $|x| + 1 + j$ is correct.
   - $M$ again uses marker positions to read $a$.
   - $M$ checks that the $i^{\text{th}}$ bit of $x$ is $a$.

2. If $c$ is a positive integer then both $Q(x, u, v, w)$ and $\overline{Q}(x, u, v, w)$ are in $\mathrm{RUD}_{\log}$ where

$$Q(x, u, v, w) \iff uv = w \wedge |uvw| \leq c \cdot \log|x|$$
$$\overline{Q}(x, u, v, w) \iff uv \neq w \wedge |uvw| \leq c \cdot \log|x|$$

Consider an ATM $M$ that behaves as follows:

On input $\$x, u, v, w\$$,

- $M$ first guesses and verifies the positions of all the markers as in the previous case.
- $M$ uses the marker positions to find $|x|$ and since $|x|$ in binary has length $\log|x|$, $M$ can easily compute $c \cdot \log|x|$.
- Using the marker positions, determine whether $|uvw| \leq c \cdot \log|x|$.
- If the above condition holds, then read each of $u$,$v$ and $w$ from the input, as in the previous case.
- Verify whether $uv = w$ (or $uv \neq w$ for $\overline{Q}$).

3. If $Q(x, y_1, \ldots, y_m)$ and $R(x, y_1, \ldots, y_m)$ are in $\mathrm{RUD}_{\log}$, then so are $Q(x, y_1, \ldots, y_m) \wedge R(x, y_1, \ldots, y_m)$ and $Q(x, y_1, \ldots, y_m) \vee R(x, y_1, \ldots, y_m)$.

   This case is trivial, using universal and existential branching, respectively.

4. If $Q(x, z_1, \ldots, z_m)$ is in $\mathrm{RUD}_{\log}$ and each of $\xi_1, \ldots, \xi_m$ is either a string in $\Sigma^*$ or one of the variables $y_1, \ldots, y_r$, then the predicate $R$ defined by $R(x, y_1, \ldots, y_r) \iff Q(x, \xi_1, \ldots, \xi_m)$ is in $\mathrm{RUD}_{\log}$.

   Inductively, $Q$ can be accepted by an alternating TM $M$ in $\bigcup_k \Sigma_k\mathrm{Time}(\log n)$. Note that $r$ and $m$ are constants here and $\xi_1, \ldots, \xi_m$ depend only on $y_1, \ldots, y_n$ and not on $x$. Therefore, the conversion of $y_1, \ldots, y_n$ into $\xi_1, \ldots, \xi_m$ only requires a finite amount of information that can be supplied to an ATM in its finite control.

   Let $f$ be a function that converts strings of the form $x, y_1, \ldots, y_r$ into $x, \xi_1, \ldots, \xi_m$. If we can show that $f$ can be computed in LH, then we can use the fact that LH functions are closed under composition to conclude the proof for this case, using the inductive hypothesis. That is, we need to show that the language $A_f = \{c\#i\#z : \text{the } i^{\text{th}} \text{ symbol of } f(z) \text{ is } c\}$ is in LH. (Note that, for well-formed inputs, $z$ will be of the form $x, y_1, \ldots, y_r$.)

Consider an ATM $M$ that behaves as follows:

On input $\$c\#i\#z\$$

- $M$ guesses all the marker positions in $\$c\#i\#z\$$ as well as in $z$ itself (there is a constant number of these) and verifies them as in the previous cases.
- $M$ uses the marker positions to compute $|x|$ and $|i|$. As in the previous cases, $M$ guesses the contents of the $c$, $i$, and $y_1, \ldots, y_r$ fields of the input, and checks that the guesses are correct.
- If $i \leq |x|$, accept iff the $i^{\text{th}}$ bit of $x$ is $c$. Otherwise, compute $\xi_1, \ldots, \xi_m$. Reject if $i > |x, \xi_1, \ldots, \xi_m|$; otherwise accept iff $c$ is in position $i - |x|$ of $\xi_1, \ldots, \xi_m$. (It is easy to verify that there is ample time to perform these operations.)

5. If $c$ is a positive integer, and $Q(x, y_1, \ldots, y_{m+1})$ is in $\text{RUD}_{\log}$, then the predicates $R$ and $R'$ defined as follows are also in $\text{RUD}_{\log}$

$$R(x, y_1, \ldots, y_m) \iff (\exists z)[|z| \leq c \cdot \log|x| \wedge Q(x, y_1, \ldots, y_m, z)]$$
$$R'(x, y_1, \ldots, y_m) \iff (\forall z)[|z| \leq c \cdot \log|x| \Rightarrow Q(x, y_1, \ldots, y_m, z)]$$

Inductively, $Q$ is accepted by some LH machine $M$. We construct a machine $M'$ accepting $R'$. On input $\$x, y_1, \ldots, y_m\$$, $M'$ starts off by getting the marker positions as in the previous cases. Then it universally verifies that for all $z$ with $|z| \leq c \cdot \log|x|$, $M$ accepts $\$x, y_1, \ldots, y_m, z\$$. To see that this is possible, note that $M$ consults its input only at the end of a computation branch. As soon as $M$ goes into input mode and has $j$ written on its index tape, $M'$ either consults the $j^{\text{th}}$ symbol of its input, or reads the symbol in position $j - |x, y_1, \ldots, y_m|$ of $z$.

A similar argument, using existential branching, shows that $R$ is in LH.

($\subseteq$) To prove this inclusion, let $L \in$ FO. Therefore, there exists a sentence in Prenex Normal Form

$$\eta = (Q_1 j_1)(Q_2 j_2) \ldots (Q_m j_m)\, \phi(j_1, \ldots, j_m)$$

such that $\forall x$

$$x \in L \iff x \models (Q_1 j_1)(Q_2 j_2) \ldots (Q_m j_m)\, \phi(j_1, \ldots, j_m)$$

where each $Q_i$ is a quantifier ($\exists$ or $\forall$).

We will show that for each such $\eta$ (and its associated $\phi$) there is a $\mathrm{RUD}_{\log}$ predicate $R_\phi$ such that for each string $x$,

$$x \models (Q_1 j_1)(Q_2 j_2) \ldots (Q_m j_m)\, \phi(j_1, \ldots, j_m)$$
$$\text{if and only if}$$
$$(Q'_1 y_1)(Q'_2 y_2) \ldots (Q'_m y_m)\, R_\phi(x, y_1, \ldots, y_m)$$

where each $Q'_i$ ranges over all binary strings of length at most $\lceil \log |x| \rceil + 1$ and is the same type (existential or universal) as $Q_i$. By the closure properties of $\mathrm{RUD}_{\log}$, and by the fact that the length of each $y_i$ is bounded, it follows that since $R_\phi$ is in $\mathrm{RUD}_{\log}$, $L$ is also in $\mathrm{RUD}_{\log}$, and the proof is complete. It remains only to construct $R_\phi$.

The formula $\phi$ is built up from the primitive predicates $=, <$, $X$, and $BIT$, and from the constants 1 and $n$. Clearly, it will suffice to show that each of these primitive predicates is "equivalent" in the same sense to a relation in $\mathrm{RUD}_{\log}$. For example, we will show that there is a relation $S$ in $\mathrm{RUD}_{\log}$ such that, for every string $x$, and for any numbers $j_1$ and $j_2$ (that represent positions in $x$), the primitive predicate $j_1 < j_2$ is true iff the relation $S(x, y_1, y_2)$ holds, where $y_1$ and $y_2$ are binary strings of length at most $\lceil \log |x| \rceil + 1$ representing the numbers $j_1$ and $j_2$, respectively.

The constant 1 is explicitly definable in $\mathrm{RUD}_{\log}$, and Jones [Jon75] has already observed that the relations $u + v = w, u \cdot v = w, u^v = w$ and $|u| = w$ are all in $\mathrm{RUD}_{\log}$, so long as $u, v, w$ are constrained to have length $\leq c \cdot \log |x|$ for some $c$. Thus the constant $n$ is definable, since the relation $|x| = j$ is equivalent to $(\sigma(x, j, 0) \vee \sigma(x, j, 1)) \wedge \neg(\sigma(x, j+1, 0) \vee \sigma(x, j+1, 1))$. It follows easily that the following set is in $\mathrm{RUD}_{\log}$: $\{x, y_1, \ldots, y_m$ : for each $i$, $|y_i| = \lceil \log |x| \rceil + 1\}$. Therefore for the rest of this proof, we will assume that all variables $y_i$ refer to strings of exactly this length. (This will result in a few simplifications.)

Now we consider each of the primitive predicates in turn.

- $(j_1 = j_2) \leftrightarrow (\forall i)[\sigma(y_1, i, 0) \iff \sigma(y_2, i, 0)]$
- $(j_1 < j_2) \leftrightarrow (\exists u)(\exists v)(\exists w)[Q(x, u, v, y_1) \wedge Q(x, u, w, y_2) \wedge \sigma(v, 1, 0) \wedge \sigma(w, 1, 1)]$
  This says that the binary representation of $j_1$ is $uv$ and the binary representation $j_2$ is $uw$ such that the most significant bits of $v$ and $w$ are 0 and 1 respectively. Note that the values taken by each of the variables $u, v, w$ have lengths $\leq \log |x|$.
- $X(j_1) \leftrightarrow \sigma(x, y_1, 1)$.
- $BIT(j_1, j_2) \leftrightarrow \sigma(y_2, y_1, 1)$.

This completes the proof. ∎

We remark that standard translational techniques may be used to prove the following generalization of Theorem 7.1. Note that for the case of $S(n) = n$, this is the well-known theorem of Wrathall [Wra78].

**Corollary 7.2** For functions $S$ such that the binary representation of $S(n)$ can be computed from $n$ in time $O(S(n))$, $\mathrm{RUD}_S = \bigcup_k \Sigma_k \mathrm{time}(S(n))$.

## 7.2 Reducibilities

Jones defined $\mathrm{RUD}_{\log}$ as a first step towards defining a very restrictive notion of reducibility. Jones took the familiar notion of a many-one reduction (namely, $f$ reduces $A$ to $B$ if $x \in A \iff f(x) \in B$), and imposed the restriction that $f$ be log-bounded rudimentary. There was one additional restriction that Jones imposed in defining his logspace-rudimentary reducibility (denoted $\leq_{\log}^{\mathrm{rud}}$); he required that there be some constant $c$ such that for all $x$, $\log |f(x)| = c \cdot \log |x|$. In [Jon75], Jones comments that this final restriction is stronger than the more natural restriction of simply having $f$ have polynomial growth rate, but explains that it "seems to be necessary" in order to have the $\leq_{\log}^{\mathrm{rud}}$ relation be transitive (i.e., in order for the composition of two log-bounded rudimentary functions to be log-bounded rudimentary). However by Theorem 7.1 we see that for functions $f$ with polynomial growth rate, $f$ is log-bounded rudimentary iff $f$ is in LH. Since the composition of two functions in LH is easily seen to be in

LH, it follows that Jones' rationale for imposing the stronger length requirement was unfounded.

Another closely related type of reducibility is the class of *first order translations* introduced by Immerman in [Imm83]. (A more complete definition, modified to allow the $BIT$ predicate as a primitive, is presented in [Imm87]. We use that definition in the following discussion.) In Immerman's formulation, a first-order translation takes as input a structure on $\{1, n\}$ and outputs a structure on $\{1, n^k\}$, where elements of $\{1, n^k\}$ are identified with $k$-tuples of elements from $\{1, n\}$. Since the input and output structures may consist of many relations of different arity, there are polynomials $p$ and $q$ such that the length of the encoding of the input structure is $p(n)$ and the length of the encoding of the output structure is $q(n)$. As Immerman has observed in [Imm87], a first-order translation may be viewed as a many-one reduction computed by dlogtime-uniform $\mathrm{AC}^0$ circuits. Indeed, the only differences among (1) logspace-rudimentary reducibility as defined by Jones, (2) first-order translations as defined by Immerman, and (3) functions computed by dlogtime-uniform $\mathrm{AC}^0$ circuits, is in the allowable relations between the input and output lengths. For (1) inputs of length $n$ are mapped to inputs of length $n^k$, for (2) inputs of length $n$ are mapped to inputs of length $q(p^{-1}(n))$ for some polynomials $p$ and $q$, and for (3) the only limitations are the implicit limitations imposed by the uniformity condition.

# Chapter 8

# Conclusion

In this thesis we have studied the power of uniform constant depth circuits. In particular, we studied the classes $AC^0$ and ACC and proved upper and lower bounds for them. We have shown that uniform ACC type circuits of subexponential size cannot compute the permanent function. We have also proved a somewhat weaker bound for certain sets in PP and in C$_=$P by showing that these sets cannot be recognized by uniform ACC type circuit families of susbsubexponential size. The proofs are based on a simulation of ACC given by Beigel and Tarui in [BT91]. We have shown how to carry out this simulation in the setting of uniform circuits. Some of the obvious open problems are:

- Is uniformity really necessary? Our lower bound proofs work only in the uniform setting. Can we prove a lower bound for the permanent with respect to nonuniform ACC circuits? Note that it is still unknown if Ntime $(2^{n^{O(1)}})$ contains sets that are not accepted by nonuniform ACC circuit families.

- The lower bound that we have for PP is not quite as strong as the one for the permanent. Can it be improved? Even though the permanent function seems to provide more information about the number of accepting paths of NP machines (the permanent gives us all the bits whereas PP only gives us the most significant bit) we still think that a subexponential lower bound can be proved for PP as well.

- In Corollary 3.3 we prove that ACC is properly contained in PP. It would be interesting to see if this result translates upwards to exponential analogs of ACC and PP. The exponential size analog of ACC has been considered before. In

[GKT92], Green, Köbler and Torán define the class ModPH (see Definition 2.10). It is not hard to see that ModPH = ACC($2^{n^{O(1)}}$). We can also show that ModPH is precisely the class of languages accepted by polynomial time ATM's with $\exists$, $\forall$ and MOD states (cf. Definition 2.7) that make a constant number of alternations. It would be interesting if we can show that ModPH $\subsetneq$ PrTime($2^{n^{O(1)}}$) using the fact that ACC $\subsetneq$ PP. Our results (as well as the results in [GKT92]) imply that ModPH $\subseteq$ $\mathrm{P}^{\mathrm{PP}}$.

It should be emphasized that our results about the complexity of PERM do not rely on any unproven complexity-theoretic assumptions. This is in contrast to other results such as [FL92], which proves stronger intractability results about PERM under the hypothesis that the polynomial hierarchy is infinite.

Let $\mathrm{SYM}^+$ denote the class of Boolean functions computable by depth two circuit families of size $2^{(\log n)^{O(1)}}$ where the circuits have a symmetric gate at the root and AND gates of fan-in $(\log n)^{O(1)}$ at the next level. This is exactly the class of Boolean functions $f$ such that $f(x_1, x_2, \ldots, x_n)$ can be expressed as $h_n(p_n(x_1, x_2, \ldots, x_n))$ for some polynomial $p_n$ over $\mathbf{Z}$ of degree $(\log n)^{O(1)}$ and norm (the sum of the absolute values of its coefficients) $2^{(\log n)^{O(1)}}$ and some function $h_n : \mathbf{Z} \to \{0, 1\}$. Beigel and Tarui [BT91] showed[1] that ACC $\subseteq$ $\mathrm{SYM}^+$ in the nonuniform setting. The techniques used in the proof of Theorem 3.1 enable us to show that ACC $\subseteq$ $\mathrm{SYM}^+$ holds in the setting of uniform circuits as well. Theorem 3.2 then gives us the following immediate corollary:

**Corollary 8.1** There is a set $Y \in \mathrm{PP}$ such that uniform $\mathrm{SYM}^+ \subseteq \mathrm{Dtime}(n^2)^Y$.

It would be interesting to see if our techniques can be used to prove lower bounds for uniform $\mathrm{SYM}^+$. For example, it is not obvious that our techniques can be used to show that there is something in PP that is not in uniform $\mathrm{SYM}^+$.

In this thesis we also began an investigation into the immunity properties of $\mathrm{AC}^0$ and other circuit complexity classes. In addition to observing that there are sets in

[1] In [BT91], the class $\mathrm{SYM}^+$ was called SYMMC. The name $\mathrm{SYM}^+$ has been suggested by Barrington in his recent survey [Bar92]. Beigel, Tarui and Toda have used the name $\mathrm{SYM}^+$ in their recent paper [BTT92] as well.

superlogarithmic deterministic space classes and superpolynomial deterministic time classes, we have also shown that there are sets in $\mathrm{P}^{\mathrm{PP}}$ that are immune to $\mathrm{AC}^0$. In fact, we have shown that $\mathrm{P}^{\mathrm{PP}}$ contains sets that are immune to $\mathrm{ACC}(subexp)$ (Theorem 4.3). Corollary 8.1 trivially implies that $\mathrm{P}^{\mathrm{PP}}$ contains sets that are immune to uniform $\mathrm{SYM}^+$. In [BTT92], Beigel, Tarui and Toda have considered augmented ACC circuit families where the circuits are allowed to have probabilistic inputs and are also allowed to have an exact-threshold gate as the output gate (an exact-threshold gate outputs 1 if exactly $k$ of its inputs are 1, where $k$ is a parameter; it outputs 0 otherwise). They are able to show that every Boolean function computed by these augmented ACC circuits is still in $\mathrm{SYM}^+$. Since it seems that the results of [BTT92] also hold in the setting of uniform circuits, it follows that $\mathrm{P}^{\mathrm{PP}}$ contains sets that are immune to the augmented version of ACC as well.

We also show that it would represent a significant advance if one were able to prove the existence (or nonexistence) of sets in NP that are immune to $\mathrm{AC}^0$. *Any* answer to the question

*Are there sets in NP that are immune to $AC^0$?*

yields non-relativizing proof techniques for attacking questions concerning deterministic and nondeterministic time-bounded computation.

Earlier work by Impagliazzo and Naor [IN88] showed that many relationships among very small complexity classes (such as $\mathrm{Dtime}((\log n)^{O(1)})$ and $\mathrm{Ntime}((\log n)^{O(1)})$) cannot be resolved without first answering long-standing questions such as whether $\mathrm{P} = \mathrm{NP} \cap \text{co-NP}$. However, this is the first time that questions about very small complexity classes like $\mathrm{AC}^0$ have been shown to have a bearing on questions about fairly large classes like E and NE.

In Theorem 3.2, we have shown that there is a set $Y$ in PP such that $\mathrm{ACC}(subexp)$ is contained in $\mathrm{Dtime}(n^2)^Y$ and we used this result to prove Theorem 4.3. It is an interesting open question if there is any set $Y'$ in the polynomial hierarchy such that $\mathrm{ACC}(subexp)$ (or even $\mathrm{AC}^0$) can be recognized in deterministic time $n^k$ relative to $Y'$, for some $k$. If so, then there are sets in the polynomial hierarchy that are immune to $\mathrm{ACC}(subexp)$ ($\mathrm{AC}^0$). However, this would also have significant consequences concerning

the complexity of the rudimentary sets.

**Proposition 8.2** If there is a set $L$ in the polynomial hierarchy and a constant $b$ such that $\mathrm{AC}^0 \subseteq \mathrm{Dtime}(n^b)^L$, then there exist constants $c$ and $l$ such that $\bigcup_k \Sigma_k\mathrm{Time}(n) \subseteq \Sigma_l\mathrm{Time}(2^{cn})$.

The proof follows quite easily from the result in Proposition 5.3. However, the conclusion of Proposition 8.2 is a much better inclusion than is known to hold, and it is reasonable to conjecture that this inclusion fails relative to some oracle, but this is not known to be the case.

Recall that $\mathrm{AC}^0 = \mathrm{FO} + < + BIT$. One reason it seems unlikely that this class would be contained in $\mathrm{Dtime}(n^k)$ for any fixed $k$, is that a first order formula with $k+1$ quantifiers, with variables ranging over the set $\{1, \ldots, n\}$ cannot be evaluated in any straightforward way in time less than $n^{k+1}$. However, it should be noted that the class $\mathrm{FO} + <$ is exactly the star-free regular sets, and thus each language in that class has a linear time algorithm [MP71] (see also [Lad77]). Some (but clearly not all) of the combinatorial techniques that are used to prove this characterization of $\mathrm{FO} + <$ apply also to the system $\mathrm{FO} + < + BIT$. It seems possible that techniques could be developed in this setting to prove non-relativizing results, using the logic-based characterizations of uniform $\mathrm{AC}^0$.

In this thesis we also considered the log-bounded rudimentary predicates defined by Jones in [Jon75] and we have shown that they correspond exactly to the class of sets accepted by dlogtime-uniform $\mathrm{AC}^0$ circuits. Thus Jones was probably the first to study this complexity class, which has loomed large in importance in recent years. We believe that this augments the (already compelling) arguments of [BIS90] in favor of using the dlogtime-uniformity condition when studying small circuit complexity classes.

It is instructive to note some of the open questions posed by Jones in [Jon75]. He mentions on more than one occasion the question of whether or not $\mathrm{RUD}_S = \mathrm{Dspace}(S(n))$, and he explicitly considers the possibility that $\mathrm{RUD}_{\log} = \mathrm{Nspace}(\log n)$ ([Jon75], Theorem 23). However, the results of [Hås87, Yao85] show that for any function $S$ such that for all $\epsilon$, $S(n) = o(n^\epsilon)$, the PARITY language is in $\mathrm{REG} - \mathrm{RUD}_S$.

(Here, REG denotes the class of regular sets.) However, any generalization of this result for larger functions $S$ will require entirely different techniques and will have to take uniformity into account in some way, since the techniques of [Nep70] (see also [Vol83]) show that for each $\epsilon > 0$, NLOG $\subseteq$ RUD$_{n^\epsilon}$, and it follows from [Wra78] that RUD$_{n^\epsilon}$ contains complete sets for each level of the polynomial hierarchy. Thus the question of whether or not RUD$_{n^\epsilon}$ = Dspace($n^\epsilon$) remains an interesting open problem. Note, however, that an affirmative answer would imply that the polynomial hierarchy is equal to PSPACE.

# References


[ABFR91] J. Aspnes, R. Beigel, M. Furst, and S. Rudich. The expressive power of voting polynomials. In *Proc. 23rd ACM Symposium on Theory of Computing*, pages 402–409, 1991.

[ABHH90] E. Allender, R. Beigel, U. Hertrampf, and S. Homer. A note on the almost-everywhere hierarchy for nondeterministic time. In *Proc. 7th Symposium on Theoretical Aspects of Computer Science,* Lecture Notes in Computer Science 415, pages 1–11. Springer-Verlag, 1990.

[AG91a] E. Allender and V. Gore. On strong separations from $AC^0$. In *Proc. 8th International Symposium on Fundamentals of Computation Theory (FCT '91),* Lecture Notes in Computer Science 529, pages 1–15. Springer-Verlag, 1991.

[AG91b] E. Allender and V. Gore. Rudimentary reductions revisited. *Inform. Process. Lett.*, 40:89–95, 1991.

[AG92] E. Allender and V. Gore. A uniform circuit lower bound for the permanent. Submitted to *SIAM J. Comput.* (in review), 1992.

[AG93] E. Allender and V. Gore. On strong separations from $AC^0$. Accepted for publication in the *DIMACS-AMS Special Volume on Complexity Theory*, 1993.

[AH90] E. Allender and U. Hertrampf. On the power of uniform families of constant depth threshold circuits. In *Proc. 15th International Symposium on Mathematical Foundations of Computer Science,* Lecture Notes in Computer Science 452. Springer-Verlag, 1990.

[Ajt83] M. Ajtai. $\Sigma^1_1$-formulae on finite structures. *Annals of Pure and Applied Logic*, 24:1–48, 1983.

[AKS83] M. Ajtai, J. Komlós, and E. Szemerédi. Sorting in $c \log n$ parallel steps. *Combinatorica*, 3:1–19, 1983.

[All89a] E. Allender. A note on the power of threshold circuits. In *Proc. 30th IEEE Symposium on Foundations of Computer Science*, pages 580–584, 1989.

[All89b] E. Allender. P-uniform circuit complexity. *J. Assoc. Comput. Mach.*, 36:912–928, 1989.

[All89c] E. Allender. Some consequences of the existence of pseudorandom generators. *J. Comput. System Sci.*, 39(1):101–124, 1989.

[All92] E. Allender. Applications of time-bounded Kolmogorov complexity in complexity theory. In O. Watanabe, editor, *Kolmogorov Complexity: Theory and Relations to Computational Complexity*. Springer-Verlag, 1992. To appear.

[AR88] E. Allender and R. Rubinstein. P-printable sets. *SIAM J. Comput.*, 17(6):1193–1202, 1988.

[AW88] E. Allender and O. Watanabe. Kolmogorov complexity and degrees of tally sets. In *Proc. 3rd Structure in Complexity Theory Conference*, pages 102–111, 1988.

[Bab87] L. Babai. Random oracles separate PSPACE from the polynomial time hierarchy. *Inform. Process. Lett.*, 26:51–53, 1987.

[Bar86] D. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in $NC^1$. In *Proc. 18th ACM Symposium on Theory of Computing*, pages 1–5, 1986.

[Bar89] D. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in $NC^1$. *J. Comput. System Sci.*, 38(1):150–164, 1989.

[Bar90] D. Barrington. Extensions of an idea of McNaughton. *Math. Sys. Theory*, 23:147–164, 1990.

[Bar92] D. Barrington. Quasipolynomial size circuit classes. In *Proc. 7th Structure in Complexity Theory Conference*, pages 86–93, 1992.

[BCGR92] S. Buss, S. Cook, A. Gupta, and V. Ramachandran. An optimal parallel algorithm for formula evaluation. *SIAM J. Comput.*, 21(4):755–780, 1992.

[BCH84] P. Beame, S. Cook, and H. Hoover. Log depth circuits for division and related problems. In *Proc. 25th IEEE Symposium on Foundations of Computer Science*, pages 1–11, 1984.

[BDG89] J. Balcázar, J. Díaz, and J. Gabarró. Structural Complexity II. In W. Brauer, G. Rozenberg, and A. Salomaa, editors, EATCS Monographs on Theoretical Computer Science, volume 22. Springer-Verlag, 1989.

[BFL91] L. Babai, L. Fortnow, and C. Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1:3–40, 1991.

[BGS75] T. Baker, J. Gill, and R. Solovay. Relativizations of the P =? NP question. *SIAM J. Comput.*, 4:431–442, 1975.

[BIS90] D. Barrington, N. Immerman, and H. Straubing. On uniformity within $NC^1$. *J. Comput. System Sci.*, 41:274–306, 1990.

[BNS89] L. Babai, N. Nisan, and M. Szegedy. Multiparty protocols and logspace-hard pseudorandom sequences. In *Proc. 21st ACM Symposium on Theory of Computing*, pages 1–11, 1989.

[Boo78] R. Book. Simple representations of certain classes of languages. *J. Assoc. Comput. Mach.*, 25:23–31, 1978.

[Bor77] A. Borodin. On relating time and space to size and depth. *SIAM J. Comput.*, 6(4):733–743, 1977.

[BS85] J. Balcázar and U. Schöning. Bi-immune sets for complexity classes. *Math. Sys. Theory*, 18:1–10, 1985.

[BT88] D. Barrington and D. Thérien. Finite monoids and the fine structure of $NC^0$. *J. Assoc. Comput. Mach.*, 35(4):941–952, 1988.

[BT91] R. Beigel and J. Tarui. On ACC. In *Proc. 32nd IEEE Symposium on Foundations of Computer Science*, pages 783–792, 1991.

[BTT92] R. Beigel, J. Tarui, and S. Toda. On probabilistic ACC circuits with an exact-threshold output gate. In *Proc. 3rd Annual International Symposium on Algorithms and Computation,* Lecture Notes in Computer Science 650, pages 420–429. Springer-Verlag, 1992.

[Bus87] S. Buss. The boolean formula value problem is in ALOGTIME. In *Proc. 19th ACM Symposium on Theory of Computing*, pages 123–131, 1987.

[Cai89] J. Cai. With probability 1, a random oracle separates PSPACE from the polynomial-time hierarchy. *J. Comput. System Sci.*, 38:68–85, 1989.

[CKS81] A. Chandra, D. Kozen, and L. Stockmeyer. Alternation. *J. Assoc. Comput. Mach.*, 28:114–133, 1981.

[Clo90] P. Clote. Bounded Arithmetic and Computational Complexity. In *Proc. 5th IEEE Structure in Complexity Theory Conference*, pages 196–199, 1990.

[Coo85] S. Cook. A taxonomy of problems with fast parallel algorithms. *Inform. and Cont.*, 64:2–22, 1985.

[FKLN90] L. Fortnow, H. Karloff, C. Lund, and N. Nisan. Algebraic methods for interactive proof systems. In *Proc. 31st IEEE Symposium on Foundations of Computer Science*, pages 2–10, 1990.

[FL92] U. Feige and C. Lund. On the hardness of computing the Permanent of random matrices. In *Proc. 24th ACM Symposium on Theory of Computing*, pages 643–654, 1992.

[FMS78] M. Fischer, A. Meyer, and J. Seiferas. Separating nondeterministic time complexity classes. *J. Assoc. Comput. Mach.*, 25:146–167, 1978.

[FS88] L. Fortnow and M. Sipser. Are there Interactive Protocols for co-NP Languages? *Inform. Process. Lett.*, 28:249–251, 1988.

[FSS84] M. Furst, J. Saxe, and M. Sipser. Parity, circuits and the polynomial time hierarchy. *Math. Sys. Theory*, 17:13–27, 1984.

[GHS91] J. Geske, D. Huynh, and J. Seiferas. A note on almost-everywhere-complex sets and separating deterministic time complexity classes. *Inform. and Comput.*, 92:97–104, 1991.

[GK90] J. Geske and D. Kakihara. Almost-everywhere complexity, bi-immunity and nondeterministic space. In *Advances in Computing and Information (ICCI '90)*, Lecture Notes in Computer Science 468, pages 44–51. Springer-Verlag, 1990.

[GKT92] F. Green, J. Köbler, and J. Torán. The power of the middle bit. In *Proc. 7th Structure in Complexity Theory Conference*, pages 111–117, 1992.

[Hås87] J. Håstad. *Computational limitations for small depth circuits.* PhD thesis, Massachusetts Institute of Technology, 1987.

[HG90] J. Håstad and M. Goldmann. On the power of small-depth threshold circuits. In *Proc. 31st IEEE Symposium on Foundations of Computer Science*, pages 610–618, 1990.

[HMP+87] A. Hajnal, W. Maass, P. Pudlak, M. Szegedy, and G. Turan. Threshold circuits of bounded depth. In *Proc. 28th IEEE Symposium on Foundations of Computer Science*, pages 99–110, 1987.

[HS65] J. Hartmanis and R. Stearns. On the computational complexity of algorithms. *Trans. AMS*, 117:285–306, 1965.

[HU79] J. Hopcroft and J. Ullman. Introduction to Automata Theory, Languages and Computation. Addison-Wesley, Reading, Massachusetts, 1979.

[HY84] J. Hartmanis and Y. Yesha. Computation times of NP sets of different densities. *Theoret. Comput. Sci.*, 34:17–32, 1984.

[IL89] N. Immerman and S. Landau. The complexity of iterated multiplication. In *Proc. 4th IEEE Structure in Complexity Theory Conference*, pages 104–111, 1989.

[Imm83] N. Immerman. Languages which capture complexity classes. In *Proc. 15th ACM Symposium on Theory of Computing*, pages 347–354, 1983. Also appeared in revised form in *SIAM J. Comput.*, 16(4), 1987.

[Imm87] N. Immerman. Expressibility as a complexity measure: Results and directions. In *Proc. 2nd Structure in Complexity Theory Conference*, pages 194–202, 1987.

[Imm89] N. Immerman. Expressibility and parallel complexity. *SIAM J. Comput.*, 18(3):625–638, 1989.

[IN88] R. Impagliazzo and M. Naor. Decision trees and downward closures. In *Proc. 3rd IEEE Structure in Complexity Theory Conference*, pages 29–38, 1988.

[IT89] R. Impagliazzo and G. Tardos. Decision versus search in super-polynomial time. In *Proc. 30th IEEE Symposium on Foundations of Computer Science*, pages 222–227, 1989.

[Jon75] N. Jones. Space-bounded reducibility among combinatorial problems. *J. Comput. System Sci.*, 11:68–85, 1975. Also see corrigendum on page 241 in *J. Comput. System Sci.* 15, 1977.

[KVVY92] R. Kannan, H. Venkateswaran, V. Vinay, and A. Yao. A circuit-based proof of Toda's theorem. To appear in *Information and Computation*, 1992.

[Lad77] R. Ladner. Application of model theoretic games to discrete linear orders and finite automata. *Inform. and Comput.*, 33:281–303, 1977.

[Lev84] L. Levin. Randomness conservation inequalities; information and independence in mathematical theories. *Inform. and Cont.*, 61:15–37, 1984.

[Lip78] R. Lipton. Model theoretic aspects of computational complexity. In *Proc. 19th IEEE Symposium on Foudations of Computer Science*, pages 193–200, 1978.

[Lis90] G. Lischke. Impossibilities and possibilities of weak separation between NP and exponential time. In *Proc. 5th IEEE Structure in Complexity Theory Conference*, pages 245–253, 1990.

[MP71] R. McNaughton and S. Papert. Counter-free Automata. MIT Press, 1971.

[MT89] P. McKenzie and D. Thérien. Automata theory meets circuit complexity. In *Proc. 16th Annual International Colloquium on Automata, Languages and Programming,* Lecture Notes in Computer Science 372. Springer-Verlag, 1989.

[Nep70] V. Nepomnjaščiĭ. Rudimentary predicates and Turing calculations. *Soviet Math. Dokl.*, 11:1462–1465, 1970.

[PD80] J. Paris and C. Dimitracopoulos. Truth definitions for $\Delta_0$ formulae. In *Logic and Arithmetic: an Internatinal Symposium Held in Honour of Ernst Specker, Monographie no. 30 de l'Enseignement Mathématique*, pages 317–329, 1980.

[PHW85] J. Paris, W. Handley, and A. Wilkie. Characterizing some low arithmetic classes. In L. Lovász and E. Szemerédi, editors, *Theory of Algorithms, Colloquia Mathematica Societatis János Bolyai, 44*, pages 353–364. North-Holland, 1985.

[Pip79] N. Pippenger. On simultaneous resource bounds (preliminary version). In *Proc. 20th IEEE Symposium on Foundations of Computer Science*, pages 307–311, 1979.

[Raz87] A. Razborov. Lower bounds for the size of circuits of bounded depth with basis $\{\wedge, \oplus\}$. *Math. notes of the Academy of Sciences of the USSR*, 41(4):333–338, 1987.

[Ruz80] W. Ruzzo. Tree-size bounded alternation. *J. Comput. System Sci.*, 21:218–235, 1980.

[Ruz81] W. Ruzzo. On uniform circuit complexity. *J. Comput. System Sci.*, 21(2):365–383, 1981.

[Sal73] A. Salomaa. Formal Languages. Academic Press, 1973.

[Sha90] A. Shamir. IP = PSPACE. In *Proc. 31st IEEE Symposium on Foundations of Computer Science*, pages 11–15, 1990.

[Sip83] M. Sipser. Borel sets and circuit complexity. In *Proc. 15th ACM Symposium on Theory of Computing*, pages 61–69, 1983.

[SM73] L. Stockmeyer and A. Meyer. Word problems requiring exponential time: Preliminary report. In *Proc. 5th ACM Symposium on Theory of Computing*, 1973.

[Smo87] R. Smolensky. Algebraic methods in the theory of lower bounds for Boolean circuit complexity. In *Proc. 19th ACM Symposium on Theory of Computing*, pages 77–82, 1987.

[Smu61] R. Smullyan. Theory of formal systems. In *Annals of Math. Studies* 47. Princeton University Press, 1961.

[Sud75] I. Sudborough. A note on tape-bounded complexity classes and linear context-free languages. *J. Assoc. Comput. Mach.*, 22(4):499–500, 1975.

[Tod89] S. Toda. On the computational power of PP and $\oplus$P. In *Proc. 30th IEEE Symposium on Foundations of Computer Science*, pages 514–519, 1989.

[Tod91] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991.

[Val79] L. Valiant. The complexity of computing the Permanent. *Theoret. Comput. Sci.*, 8:189–201, 1979.

[Ven92] H. Venkateswaran. Circuit definitions of nondeterministic complexity classes. *SIAM J. Comput.*, 21(4):655–670, 1992.

[Vol83] H. Volger. Rudimentary relations and Turing machines with linear alternation. In *Logic and Machines : Decision Problems and Complexity,* Lecture Notes in Computer Science 171, pages 131–136. Springer-Verlag, 1983.

[VV86] L. Valiant and V. Vazirani. NP is as easy as detecting unique solutions. *Theoret. Comput. Sci.*, 47:85–93, 1986.

[Wil79] A. Wilkie. Applications of complexity theory to $\Sigma_0$-definability of problems in arithmetic. In Lecture Notes in Mathematics, volume 834, pages 363–369. Springer-Verlag, 1979.

[Wil83] R. Wilber. Randomness and the density of hard problems. In *Proc. 24th IEEE Symposium on Foundations of Computer Science*, pages 335–342, 1983.

[Wra78] C. Wrathall. Rudimentary predicates and relative computation. *SIAM J. Comput.*, 7:194–209, 1978.

[Yao85] A. Yao. Separating the polynomial-time hierarchy by oracles. In *Proc. 26th IEEE Symposium on Foundations of Computer Science*, pages 1–10, 1985.

[Yao89] A. Yao. Circuits and local computation. In *Proc. 21st ACM Symposium on Theory of Computing*, pages 186–196, 1989.

[Yao90] A. Yao. On ACC and threshold circuits. In *Proc. 31st IEEE Symposium on Foundations of Computer Science*, pages 619–627, 1990.

# Vita

## Vivek Gore

**1983** Graduated from Colvin Taluqdars' College, Lucknow, India.

**1983-87** Attended Indian Institute of Technology, Kanpur, India.

**1987** B.Tech. in Computer Science, Indian Institute of Technology, Kanpur, India.

**1987-92** Graduate work in Computer Science, Rutgers, The State University of New Jersey, New Brunswick, New Jersey.

**1989** M.S. in Computer Science, Rutgers, The State University of New Jersey, New Brunswick, New Jersey.

**1993** Ph.D. in Computer Science, Rutgers, The State University of New Jersey, New Brunswick, New Jersey.