Some Results in Non-Uniform Complexity Theory

by

Iannis Tourlakis

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

Abstract

Some Results in Non-Uniform Complexity Theory

Iannis Tourlakis
Master of Science
Graduate Department of Computer Science
University of Toronto
2000

This thesis studies models and limitations of non-uniform computation. A way of quantifying the descriptional complexity of circuits due to Rackoff is introduced and analyzed. A non-uniform variant of the Turing machine is defined and analyzed. Relations with Karp-Lipton advice classes are explored.

The well-known time and space hierarchies are generalized to non-uniform classes. Moreover, diagonalization is used to show that for $t(n) \log t(n) \in o(T(n))$, deterministic machines taking $O(T(n))$ time cannot be simulated by deterministic machines taking $O(t(n))$ time and n bits of advice on inputs of length n.

The first non-trivial time-space tradeoffs for SAT on non-uniform machines are given. Strong lower bounds for the Permanent on partially non-uniform threshold circuits are shown. New separations of uniform classes are also obtained. For instance, it is shown that for all $\epsilon > 0$ and all integers k, $\mathsf{DTISP}(n^k, n^{1-\epsilon}) \subsetneq \mathsf{NTIME}(n^k)$. Most results are for random-access multi-tape Turing machines.

Acknowledgements

I would like to thank my supervisor Stephen Cook for his friendly guidance. Without Steve's thoughtful direction and help, this thesis could not have been written.

Charles Rackoff provided the groundwork for much of the work that appears in this thesis. Moreover, Charlie carefully proof-read this thesis, providing many helpful and insightful comments.

I would like to thank Dieter van Melkebeek for many interesting discussions on the material in this thesis and for allowing me to quote some of his unpublished results. I would also like to thank Montserrat Hermo for making her Ph.D. thesis on non-uniform complexity classes with sublinear advice functions available to me. Her thesis was very helpful throughout my studies.

Valentine Kabanets, Antonina Kolokolova, Steve Myers and Alan Skelley provided many helpful discussions on the work contained in thesis. Their help is gratefully acknowledged.

I would like to thank John Mylopoulos who from the beginning of my university career gave me the opportunity to work with his research group. The experiences gained while working with John have been invaluable during both my undergraduate and graduate studies.

A big hand for my office mates Jonathan Shekter and John Watkinson is in order for the surreal environment they provided while "working in the zoo". A special thanks goes out to John for his game that provided frequent relaxation from the rigours of thesis work.

Many friends in and out of the department have helped make the last two years memorable. These include both the people named above as well as Geneviève Arboit, Scott Black, Nadine Kerrigan, Jennifer Listgarten, Maria Mylopoulos, Nataša Pržulj, Eric Ruppert, Sai Saiedfar and Calum Tsang.

The financial support of the Natural Sciences and Engineering Research Council of Canada is gratefully acknowledged.

Most importantly, I would like to thank my family for their encouragement and support. And for putting up with me these last couple of years!

Contents

Chapter 1

Introduction

The main goal of computational complexity theory is to determine the minimum resources needed to solve particular problems on computers. To that end, abstract models of algorithms such as Turing machines and Boolean circuits are introduced. Subsequently, these models are used to classify problems into complexity classes. For instance, problems can be classified by measuring the resources required by a Turing machine to compute them, or alternatively by the types and sizes of circuits recognizing them. Insight on the inherent difficulty of solving problems is then obtained by determining the relations between complexity classes. It is especially of interest and of great practical importance to determine which problems have *feasible* algorithms.

While there has been considerable success in determining which problems in a given complexity class epitomize the difficulty of the hardest problems in that class, there has been little success in providing concrete lower bounds for natural problems. Indeed, amongst the class of problems solvable in exponential time, there are, with some notable exceptions, no natural problems for which non-trivial lower bounds are known for the resources needed to solve them on sufficiently realistic computing machines, such as random-access multi-tape Turing machines.

This thesis attempts to make progress by studying models and limitations of non-uniform computation. The first part of this thesis examines models of non-uniform computation. Loosely speaking, non-uniform models, such as Boolean circuits, are useful for measuring the minimum complexity of algorithms recognizing finite languages. In Chapter 2 a non-uniform variant of the general purpose random-access multi-tape Turing machine is defined and analyzed. Subsequently, complexity classes are defined with respect to this new model of computation and are then compared with Karp-Lipton advice classes. The relative merits of defining complexity classes using non-uniform Turing machines and Karp-Lipton advice functions are discussed. In addition, Chapter 2 describes a new way of quantifying the descriptional complexity of a circuit. This new method is due to Rackoff [53]. Relations with traditional methods for quantifying the complexity of a circuit are explored.

An important question in complexity theory is determining the amount computing resources need to be increased by before the power of a specific computation model is increased. Indeed one of the successes of complexity theory has been in establishing tight time and space hierarchies for Turing machines. Chapter 3 of this thesis is devoted to proving tight hierarchies of non-uniform complexity classes as well as to establishing tight separations of uniform and non-uniform classes. One of the main goals is to generalize the well-known deterministic and nondeterministic time and space hierarchies to non-uniform machines. Recall that the class $\mathsf{DTIME}(t(n))/a(n)$ consists of all languages accepted by deterministic machines using $t(n)$ time and $a(n)$ advice on inputs of length n. The first hierarchy theorem proved for non-uniform classes shows that

$$\mathsf{DTIME}(t(n))/a(n) \subsetneq \mathsf{DTIME}(T(n))/a(n),$$

for all functions $a(n)$, $t(n)$ and $T(n)$ such that $t(n)\log t(n) \in o(T(n))$ and T is time-constructible. Similar results are obtained for non-uniform classes defined by nondeterministic and alternating machines. In addition, it is shown in Chapter 2 that

$$\mathsf{DTIME}(T(n)) \not\subseteq \mathsf{DTIME}(t(n))/n,$$

for all functions $t(n)$ and $T(n)$ such that $t(n)\log t(n) \in o(T(n))$ and T is time-constructible. This result and the techniques used to obtain it are exploited in Chapters 4–6 to obtain lower bounds for specific natural problems on non-uniform computation models.

It is of particular interest to determine the relationship between P, the class of problems with feasible algorithms, and NP, a class that contains many important but seemingly intractable problems. Answering this question would determine whether many important combinatorial problems such as scheduling have feasible algorithms. It is strongly believed that $\mathsf{P} \neq \mathsf{NP}$. Furthermore, Cook showed [18] that to separate these two classes, it suffices to show that a deterministic machine requires superpolynomial time to recognize the set SAT of satisfiable quantifier-free Boolean formulas. Consequently, obtaining lower bounds for SAT is one of the main goals of complexity theory. Nevertheless, while it is expected that SAT requires exponential deterministic time and exponential size circuits, even superlinear lower bounds have proved elusive.

Nevertheless, Fortnow [26] and later Lipton and Viglas [46] were able to obtain non-trivial lower bounds for SAT on random-access multi-tape deterministic Turing machine provided that the machines are restricted to sublinear space. For instance, Fortnow showed that for all $\epsilon > 0$, SAT cannot be computed by a random-access multi-tape deterministic Turing machine using $n^{1+o(1)}$ time and $n^{1-\epsilon}$ space. More generally, Fortnow showed that for all $\epsilon > 0$ and any unbounded function $r(n) \in O(\frac{\log n}{\log\log n})$,

$$\overline{\mathrm{SAT}} \notin \mathsf{NTIME}(n^{1+\frac{1}{r(n)}}) \cap \mathsf{NTISP}(n^{o(r(n))}, n^{1-\epsilon}).$$

In addition, Fortnow's paper presented a proof (credited to Harry Buhrman) showing that SAT cannot be computed by logarithmic-space uniform NC^1 circuits of size $n^{1+o(1)}$. Later, Lipton and Viglas [46], using techniques similar to the ones used by Fortnow, showed that for all $\delta > 0$, SAT cannot be computed by a multi-tape deterministic Turing machine running in $n^{\sqrt{2}-\delta}$ time and using polylogarithmic space. The paper by Lipton and Viglas further claimed that for all $\delta > 0$, SAT cannot be computed by a multi-tape deterministic Turing machine using $n^{2-\delta}$ time and polylogarithmic space; however, this result was subsequently retracted [64]. Lipton and Viglas stated their results for multi-tape Turing machines, but their results hold for the more general random-access multi-tape Turing machine model.

Both the Fortnow and Lipton-Viglas results built on techniques used by Kannan [43] to show the existence of an integer k such that for all integers $j \neq k$, $\mathsf{NTIME}(n^j)$ properly contains $\mathsf{DTISP}(n^j, o(n^{j/k}))$.

Indeed, the three components in the proofs of all these results are diagonalization, arguments showing that machines using small amounts of space can be sped up using alternating machines, and finally, work showing that if SAT or, more generally, nondeterministic computations have fast co-nondeterministic algorithms, then alternating computations can be efficiently simulated by nondeterministic computations.

In Chapters 4 and 5 the arguments used by Kannan, Fortnow, and Lipton-Viglas are combined with the work from Chapters 2 and 3 to obtain the first non-trivial time-space lower bounds for SAT on non-uniform random-access Turing machines and partially non-uniform circuits. In Chapter 4 non-uniform extensions to the Fortnow-Buhrman bounds in [26] are presented. Call a function f *subpolynomial* if $f(n) \in n^{o(1)}$. Note that $\log^k n$ and $\log^{\log\log n} n$ are in $n^{o(1)}$. Buhrman's lower bound is generalized by showing that SAT does not have NC^1 circuits of size $n^{1+o(1)}$ generated by a nondeterministic log-space transducer taking subpolynomial advice. In addition, similar techniques are used to show that SAT does not have quasilinear NC^1 circuits with subpolynomial descriptors. Subsequently, Fortnow's lower bound for SAT on random-access Turing machines is generalized by showing that for all $\epsilon > 0$,

$$\overline{\mathrm{SAT}} \notin \mathsf{NTISP}(n^{1+o(1)}, n^{1-\epsilon})/n^{o(1)}.$$

More generally, it is shown that for all unbounded functions $r(n) \in O(\frac{\log n}{\log\log n})$ and any $\epsilon > 0$,

$$\overline{\mathrm{SAT}} \notin \mathsf{NTIME}(n^{1+\frac{1}{r(n)}})/n^{o(1)} \cap \mathsf{NTISP}(n^{o(r(n))}, n^{1-\epsilon})/n^{o(1)}.$$

In Chapter 5 the work of Lipton and Viglas [46] is extended. The proof framework of [46] is combined with the diagonal results from Chapter 3 to obtain time-space tradeoffs for SAT and nondeterministic time on deterministic machines taking subpolynomial advice. In addition, time-space tradeoffs for $\overline{\mathrm{SAT}}$ and co-nondeterministic time on nondeterministic machines taking subpolynomial advice are obtained. The specific results shown in Chapter 5 are as follows. First a general theorem is proved showing that for all rational $r \geq 1$ and all ϵ, $0 < \epsilon \leq 1/r$, if

$$\mathsf{NTIME}(n^r) \subseteq \mathsf{DTISP}(n^{\alpha r}, n^{\epsilon r})/n^{o(1)},$$

then $\alpha \geq \frac{1}{2}(\sqrt{\epsilon^2 + 4 + 4/r} - \epsilon)$. One corollary of this result is that for all $\delta > 0$ and any ϵ, $0 < \epsilon < 1$,

$$\mathrm{SAT} \notin \mathsf{DTISP}(n^{\frac{1}{2}(\sqrt{\epsilon^2+8}-\epsilon)-\delta}, n^{\epsilon})/n^{o(1)}.$$

This result generalizes both the Fortnow and Lipton-Viglas time-space tradeoffs for SAT. In particular, this corollary implies that $\mathrm{SAT} \notin \mathsf{DTISP}(n^{1.186}, \sqrt{n})/n^{o(1)}$. In addition, this corollary says that if M is a deterministic random-access multi-tape Turing machine solving SAT using $t(n)$ time, $s(n)$ space and subpolynomial advice, then $t(n)s(n) \in \Omega(n^{(\sqrt{17}+1)/4})$. A second corollary of the general theorem is that for all rational $r \geq 1$ and all $\epsilon > 0$,

$$\mathsf{DTISP}(n^r, n^{1-\epsilon}) \subsetneq \mathsf{NTIME}(n^r).$$

Note the similarity of this result to Kannan's result [43] mentioned above. Since the constant in Kannan's result is unknown, the exact relationship between Kannan's result and this corollary is uncertain. For co-nondeterministic time, a general theorem is given showing that for all rational $r \geq 1$ and all sufficiently small positive ϵ, if

$$\mathsf{coNTIME}(n^r) \subseteq \mathsf{NTISP}(n^{\alpha r}, n^{\epsilon r})/n^{o(1)},$$

then $r\alpha^3 + \epsilon r\alpha^2 - \alpha - r \geq 0$. Time-space tradeoffs for $\overline{\text{SAT}}$ on nondeterministic machines taking subpolynomial advice are subsequently derived. The results in Chapters 4 and 5 are all shown for random-access multi-tape Turing machines.

While obtaining strong lower bounds for SAT seems quite hard, considerably more progress can be made for problems, such as computing the Permanent of a matrix, that are strongly believed to be harder than SAT. Indeed, Allender and Gore [5] were able to show that computing the Permanent requires exponential size on uniform ACC^0 circuits. Later Caussinus, McKenzie, Thérien and Vollmer [15] proved the existence of problems in the counting hierarchy requiring superpolynomial size uniform TC^0 circuits. Subsequently, Allender [4] improved on their result by showing that if T is a function such that for all k, $T^{(k)}(n) \in o(2^n)$, then the Permanent cannot be computed by uniform TC^0 circuits of size $T(n)$. Note that while the lower bounds obtained in [15] and [4] for the Permanent are not as strong as those obtained in [5], the bounds in [15] and [4] are for more powerful circuits. In Chapter 6, the techniques from Chapters 2 and 3 are used to extend Allender's lower bound for the Permanent to TC^0 circuits of subpolynomial descriptive size.

1.1 Related work

In the area of non-uniform complexity, the paper by Karp and Lipton [45] occupies a seminal position. In this paper, the idea of quantifying non-uniformity using advice functions was first proposed. In addition, this paper contained the first results showing how upper bounds for non-uniform classes can imply upper bounds for uniform classes. For instance, Karp and Lipton showed that if SAT has polynomial size circuits, then the polynomial hierarchy collapses to the second level. In addition, they showed that if SAT $\in \mathsf{P}/log$, then $\mathsf{P} = \mathsf{NP}$. Later, Yap [69] generalized these results to show, for instance, that if the non-uniform polynomial hierarchy collapses to level i, then the (uniform) polynomial hierarchy collapses to level $i+2$. Another important source of results relating uniform and non-uniform classes is Kannan's paper [42] showing that for each integer k there is a language in the second level of the polynomial hierarchy that cannot be computed by circuits of size n^k. In addition, [42] contains a proof of the fact that if $s(n)$ is superpolynomial, then $\mathsf{DSPACE}(s(n))$ contains a language that requires circuits of superpolynomial size. The ideas in the proof of the latter result are used in several of the proofs in Chapter 3.

Important work on the structure of non-uniform complexity classes has been done by several authors. Balcázar, Díaz and Gabarró [6] studied the internal structure of $\mathsf{PSPACE}/poly$. Balcázar and Schöning [10] studied classes with logarithmic advice. Balcázar, Hermo and Mayordomo [9] and Hermo and Mayordomo [36] gave circuit characterizations of logarithmic advice classes that are further discussed in this thesis. Hermo [35] and Balcázar and Hermo [8] further studied the structure of classes with logarithmic, and more generally, sublinear advice. Results and techniques from [8,9,35] are touched upon in Chapters 2 and 3 in this thesis.

There is a rich literature on time-space tradeoffs. The first such result was Cobham's time-space tradeoff for the recognition of the set of palindromes [17]. Another early result was contained in [20] where Cook addresses the question of whether P is in $\mathsf{DSPACE}(\log^2 n)$. To that end, Cook showed that certain problem "path systems", complete for P, require $\sqrt{n}$ pebbles to solve in a restricted setting. In [21] Cook showed that deterministic CFLs are accepted simultaneously in polynomial time and log squared space. Dúriś and Galil [23], Karchmer [44], and Gurevich and Shelah [31] obtain time-space tradeoffs by taking advantage of the inefficiencies of computation models that read their input sequentially. In addition, time-space lower bounds for computation models such as Jumping Automata for Graphs (JAGs) have also been shown by Edmonds [25], Barnes and Edmonds [11], and Edmonds and Poon [24]. Time-space tradeoffs have been proved for Comparison models of computation by Yao [67]. Beame [12], Beame, Saks and Thathachar [13] and Ajtai [2,3] have obtained non-uniform time-space tradeoffs on branching programs.

There are several separations related to the time-space tradeoffs discussed in this thesis. In many cases, the proofs of these separations are the foundations of many of the time-space tradeoffs mentioned above and presented in this thesis. Hopcroft, Paul and Valiant [37] showed that for sufficiently "nice" $t(n)$, $\mathsf{DTIME}(t)$ is properly contained in $\mathsf{DSPACE}(t)$. Later, Kannan [43] showed that there exists an integer k such that for all integers $j \neq k$, $\mathsf{DTISP}(n^j, o(n^{j/k}))$ is properly contained in $\mathsf{NTIME}(n^j)$. Almost concurrently, Paul, Pippenger, Szemerédi and Trotter [51] showed that $\mathsf{DTIME}(n)$ is properly contained in $\mathsf{NTIME}(n)$ for sequential multi-tape Turing machines. Later, Gupta [30] showed that for time-constructible functions $t(n)$, $\mathsf{DTIME}(t(n)\log^* t(n))$ is properly contained in $\Sigma_2^{t(n)}$. As a corollary, it follows that either $\mathsf{NTIME}(n) \neq \mathsf{coNTIME}(n)$ or $\mathsf{DTIME}(t(n)) \subsetneq \mathsf{NTIME}(t(n))$ for all time-constructible $t(n)$. However, Gupta's results only hold for standard multi-tape Turing machines rather than random-access machines. There are also, of course, the results of Fortnow [26] and Lipton and Viglas [46] which were mentioned in the introduction. Recently, Fortnow and van Melkebeek [27] have improved on the results in [26] and [46]. These improvements are discussed in Chapter 5.

As mentioned in the introduction, an exponential lower bounds for computing the Permanent of a matrix on uniform bounded-depth uniform circuits with AND, OR, NOT and MODm gates (ACC^0 circuits) was given by Allender and Gore [5]. Caussinus, McKenzie, Thérien and Vollmer [15] later showed that there are problems in the counting hierarchy that require superpolynomial size bounded-depth circuits with majority gates (TC^0 circuits). The first non-trivial lower bounds for circuits were obtained by Ajtai [1] and Furst, Saxe and Sipser [28] who showed that the parity of n bits cannot be computed by polynomial size bounded-depth circuits with unbounded fan-in AND, OR and NOT gates. Håstad [32] and Yao [68] extended these results to show that in fact exponential size is required to compute the parity of n bits using circuits of bounded-depth and unbounded fan-in AND, OR and NOT gates. Subsequently, Razborov [54] followed by Smolensky [59] proved similar lower bounds for constant-depth circuits with unbounded fan-in AND, OR, NOT and MODp gates computing MODq where p and q are distinct primes. It remains open whether any problem in $\mathsf{NTIME}(2^{n^{O(1)}})$ does not have linear-size constant-depth circuits consisting of only MOD6 gates.

1.2 Preliminaries

As is common in complexity theory, "solving a problem" is rephrased as asking whether a given string belongs in some language or set corresponding to the problem. For example, instead of asking whether a propositional formula ϕ is satisfiable, one considers the set SAT of all satisfiable propositional formulas and asks whether ϕ is in SAT. Some brief comments about languages are therefore in order. An *alphabet*

Σ is a non-empty finite set. In this thesis, Σ will typically be $\{0, 1\}$, unless stated otherwise. A *word* or *finite string* over an alphabet Σ is a finite sequence of elements from Σ. If Σ is an alphabet, Σ^n denotes all finite strings over the alphabet Σ of length n, while Σ^* denotes all finite strings over the alphabet Σ. A *language* over an alphabet Σ is a subset of Σ^*. If L is a language over the alphabet Σ, then $\bar{L}_\Sigma$ denotes the complement of L. When Σ can easily be inferred by context, the complement of L is simply denoted by $\bar{L}$.

1.2.1 Uniform computation models and complexity classes

Most of the results in this thesis, such as those in Chapters 4 and 5 can be seen to hold for any general sequential model of computation that can efficiently simulate a machine given a description of that machine as input. Nevertheless, a particular computation model must be chosen to present and prove the results in this thesis. The main model used is an extension of the standard multi-tape Turing machine [7,38,49]. This model is the *random-access multi-tape Turing machine* and is defined much like the computation model of the same name used in [27]. Such machines are like normal multi-tape Turing machines except that they have two types of tapes: index and non-index tapes. With the exception of the output tape, the tapes come in pairs. Each non-index tape T (other than the output tape) has an associated index tape I_T such that the machine can move its head on T to the position indicated by I_T in one step. The heads on the index tapes can only be moved one unit to the left or right in one step. One of the random-access non-index tapes is read-only and contains the input for the machine. If the machine has output, it is equipped with a unidirectional write-only output tape. The length of an index tape is restricted to $O(\log t)$ where t is the running time of the machine. The space used by a machine is the largest non-index tape location indexed. Machines with output are sometimes called *transducers*.

Why use the random-access Turing machine model? First of all, random-access machines must be used, instead of standard multi-tape Turing machines, since many of the results in this thesis have trivial proofs on standard multi-tape Turing machines. Indeed, time-space tradeoffs stronger than the ones given in this thesis can be obtained for standard Turing machines by exploiting the known strong time-space tradeoffs for recognizing, say, the set of palindromes [17] on standard Turing machines. So a random-access model of computation is necessary to be more convinced that the results in this thesis truly say something about the powers of, say, nondeterminism rather than just commenting on the primitiveness of the underlying computation model.

The particular choice of the random-access Turing machine as the model of random-access computation is then made since these machines are essentially as easy to work with as standard multi-tape Turing machines while still being able to simulate, as far as can be determined, more realistic machines (like RAMs) with only polylogarithmic factor losses in efficiency. This again addresses the desire to obtain the most general results possible. For more remarks on the necessity of using at the very least a model with random-access to its input, see also [27]. In the sequel all references to Turing machines will be to random-access multi-tape Turing machines unless explicitly stated otherwise.

Let M be a Turing machine. The language accepted by M is denoted by $L(M)$. Let $\overline{M}$ be the machine that computes just like M except that if M enters an accepting (rejecting) state then $\overline{M}$ enters a rejecting (accepting) state, and if M enters an existential (universal) state then $\overline{M}$ enters a universal (existential) state. Note that $L(\overline{M}) = \overline{L(M)}$. A *configuration* of M consists of a "snapshot" of the machine's work-tapes, state and head positions at some given step M's computation. Let $A \vdash_{M,x} B$ be true if M on input x can reach configuration B from configuration A in one step. More generally, $A \vdash^k_{M,x} B$ is true if M on input x can reach configuration B from configuration A in k steps, and $A \vdash^*_{M,x} B$ is true if M on input x can reach configuration B from configuration A in some finite number of steps. The input x will often be omitted when using the $\vdash$ notation if it can easily be determined from context. Let $M \downarrow x$ be true if M halts and accepts on input x, and let $M \uparrow x$ be true otherwise. A Turing machine M takes $t(n)$ *time* or, equivalently, is $t(n)$ *time-bounded* if on all inputs of length n, M takes at most $t(n)$ steps to halt. A Turing machine M uses $s(n)$ *space* or, equivalently, is $s(n)$ *space-bounded* if on all inputs of length n the largest position read by M on a non-index tape, excluding the input tape, is at most $s(n)$.

The reader is assumed to be familiar with the concept of nondeterminism. In addition, familiarity with standard complexity classes such as P, NP, L, NL and PSPACE is assumed. For completeness, the definitions of classes recurring throughout the thesis now follow. Let $\mathsf{DTIME}(t(n))$ ($\mathsf{DSPACE}(s(n))$) denote the class of all languages accepted by $O(t(n))$ time-bounded ($O(s(n))$ space bounded, respectively) deterministic Turing machines. The classes $\mathsf{NTIME}(t(n))$ and $\mathsf{NSPACE}(s(n))$ are the nondeterministic versions. The following classes are of particular importance:

$$\mathsf{L} = \mathsf{DSPACE}(\log n)$$

$$\mathsf{NL} = \mathsf{NSPACE}(\log n)$$

$$\mathsf{P} = \mathsf{DTIME}(n^{O(1)})$$

$$\mathsf{NP} = \mathsf{NTIME}(n^{O(1)})$$

$$\mathsf{PSPACE} = \mathsf{DSPACE}(n^{O(1)})$$

Note that in the above list, each class contains all the classes listed above it. Note also that whereas it is an easy consequence of the standard time hierarchy theorem [33] that L is properly contained in PSPACE, none of the adjacent inclusions in the above list is known to be proper; nevertheless, each of the inclusions is strongly believed to be proper. Note that the basic complexity classes $\mathsf{DTIME}(t(n))$, $\mathsf{DSPACE}(s(n))$, together with their nondeterministic analogues, are usually defined using standard multi-tape Turing machines rather than using random-access Turing machines as done here. Nevertheless, since standard multi-tape Turing machines can simulate random-access Turing machines with only a polynomial factor slowdown, the five special complexity classes defined above are the same regardless of whether the basic time and space classes are defined using standard or random-access Turing machines.

If $\mathcal{C}$ is a class of languages, let $\mathsf{co}\mathcal{C} = \{L : \bar{L} \in \mathcal{C}\}$. Let $\mathsf{DTISP}(t(n), s(n))$ be the class of all languages accepted by Turing machines running simultaneously in time $O(t(n))$ and space $O(s(n))$, and let $\mathsf{NTISP}(t(n), s(n))$ be the nondeterministic version.

The reader is assumed to be familiar with the concepts of relativization and oracle machines. Let A be an arbitrary set. An *oracle Turing machine* $M^?$ is a Turing machine M that is provided with an extra write-only query tape and three extra states q_{QUERY}, the query state, and $q_{\mathrm{YES}}, q_{\mathrm{NO}}$, the answer states. Note that random access to the query tape is no allowed. Let A be an arbitrary language. The machine

M^A computes like a normal Turing machine except for transitions from the query state q_{QUERY}. From the query state, the machine proceeds to state q_{YES} or q_{NO} depending on whether the current string written on the query tape is in A. The query tape is instantly erased upon entering either answer state. The point is that M^A can answer membership questions about A in one step. Time complexity for oracle Turing machines is defined analogously as for ordinary Turing machines. If $\mathcal{C}$ is a time complexity class, the class $\mathcal{C}^A$ is defined to be the class of languages accepted by machines of the same sort as those accepting the languages in $\mathcal{C}$ only now the machines have access to an oracle deciding A. If $\mathcal{F}$ is a class of languages, then $\mathcal{C}^{\mathcal{F}}$ is the union of all $\mathcal{C}^A$ where A ranges over all languages in $\mathcal{F}$.

The reader is assumed to be familiar with the concept of alternation [16] and with the polynomial-time hierarchy [47,60]. Recall that the polynomial-time hierarchy is equal to the union of the classes Σ_i^p where the classes Σ_i^p are defined inductively as follows:

1. $\Sigma_0^p = \mathsf{P}$.

2. $\Sigma_{i+1}^p = \mathsf{NP}^{\Sigma_i^p}$, for $i \geq 0$.

The polynomial-time hierarchy is sometimes denoted by PH. Recall that $\mathsf{PH} \subseteq \mathsf{PSPACE}$. Note that the complement of each class Σ_i^p is denoted by Π_i^p. A $\Sigma_{a(n)}^{\mathcal{F}}$ *machine* ($\Pi_{a(n)}^{\mathcal{F}}$ *machine*) is an alternating Turing machine that runs in time bounded by a function $t(n)$ in $\mathcal{F}$ and which makes at most $a(n) - 1$ alternations beginning with $\exists$ ($\forall$) on inputs of length n. Note then that an alternate characterization of the polynomial hierarchy is as the class of languages accepted by $\Sigma_{O(1)}^p$ machines, where the p superscript indicates that the machines run in polynomial time. More generally, let $\Sigma_{a(n)}^{\mathcal{F}}$ ($\Pi_{a(n)}^{\mathcal{F}}$) denote the class of all languages accepted by $\Sigma_{a(n)}^{\mathcal{F}}$ ($\Pi_{a(n)}^{\mathcal{F}}$) machines. When $a(n)$ is unbounded and $\mathcal{F}$ only contains functions bounded by polynomials, the classes $\Sigma_{a(n)}^{\mathcal{F}}$ and $\Pi_{a(n)}^{\mathcal{F}}$ are sometimes referred to as non-constant levels of the polynomial-time hierarchy.

More information about the classes and concepts discussed above can be found in standard structural complexity texts [7,38,49]. With the exception of the material found in the section on circuit complexity below, all other uniform complexity classes and models of computation not mentioned above but required in this thesis are defined as they are introduced.

1.2.2 Circuit complexity

Combinatorial circuits are an alternate way to model computation that will be frequently used in this thesis. Circuits are useful for measuring the size of algorithms recognizing finite sets. This is in contrast to Turing machines which are used to measure the resources required to recognize infinite sets. Circuits are a "non-uniform" model of computation whereas Turing machines are a "uniform" model of computation. The meaning of these concepts will soon become apparent.

The reader is assumed to be familiar with Boolean functions and the circuit model of computation. Unless otherwise specified, gates in a circuit are assumed to be either AND, OR, NOT, constant or input gates. The *size* of a circuit is the number of gates it contains. The *depth* of a circuit is the length of its longest path. The *width* of a circuit can be defined as follows. Fix a circuit C, and let W_0 denote the set of input gates in C, and for $j > 0$ let W_j be the set of gates in C that are in none of the W_i, $i < j$, and such that all predecessors of any gate in W_j are in some W_i, $i < j$. Then the width of C is equal to

$\max_{j>0} |W_j|$. A family of circuits are said to have *bounded fan-in* if the number of inputs to each gate in each circuit is bounded; otherwise, the circuit family has *unbounded fan-in*.

Given a language $L \in \{0,1\}^*$, a family of circuits $\{C_n\}_{n=1}^{\infty}$ recognize L if for all n, C_n has n inputs and for all $x \in \{0,1\}^n$, $x \in L$ if and only if C_n outputs **true** on input x. Note that there need be no relation between different circuits in the family $\{C_n\}$ other than the fact that they compute the language L. This contrasts with a Turing machine where the same algorithm is used on all input lengths. This is why circuits are said to be a *non-uniform* computational model and Turing machines are said to be a *uniform* model of computation. A circuit family $\{C_n\}$ is said to have *size* $s(n)$ if for all n, the number of gates in C_n is $s(n)$. Width and depth for a family of circuits are defined analogously. Languages that have polynomial-size circuits are sometimes said to have *small circuits.*

Languages can now be classified by the types of circuit families required to recognize them. For instance, $\mathsf{P}/poly$ turns out to be the class of all languages with polynomial-size circuits while the class P corresponds to the class of all languages with *uniform* polynomial-size circuits [56] (the notion of uniform circuits is discussed below). It is an important conjecture that NP is not contained in $\mathsf{P}/poly$. That is, it is strongly believed that NP-complete problems do not have polynomial-size circuits. Also important are the circuit classes NC^1 and AC^0. A language L is in NC^i, $i \geq 0$, if there exists a family $\{C_n\}$ of bounded fan-in circuits with AND, OR and NOT gates of polynomial size and $O(\log^i n)$ depth recognizing L. A language L is in AC^i, $i \geq 0$, if there exists a family $\{C_n\}$ of unbounded fan-in circuits with AND, OR and NOT gates of polynomial size and $O(\log^i n)$ depth recognizing L.

Other circuit complexity classes that are considered in this thesis are ACC^0 and TC^0. A language L is in ACC^0 if there exist an integer m and a family $\{C_n\}$ of unbounded fan-in circuits with AND, OR, NOT and MODm gates of polynomial size and bounded depth recognizing L. A language L is in TC^0 if there exists a family $\{C_n\}$ of unbounded fan-in circuits with MAJORITY, constant, input and negated input gates of polynomial size and bounded depth recognizing L. One final type of circuit that will occasionally be used is a *projection* circuit. A projection circuit is a circuit consisting only of constant, input and negated input gates.

Note that even some undecidable languages have trivial circuit complexity. Indeed, any unary non-recursive language can be decided by a circuit of depth 1. Consequently, the notion of *uniform* circuits has been developed to avoid these pathological cases. Briefly, a circuit family $\{C_n\}$ is uniform if there exists an algorithm (i.e., a Turing machine) that on input 1^n outputs a description of C_n. For instance, a circuit family is called *logarithmic-space uniform* if there exists a logarithmic space-bounded multi-tape Turing machine that outputs C_n on input 1^n. In this thesis, the notation $\mathsf{NC}^1[\mathsf{L}]$ will be used to denote the class of all languages accepted by such circuits. This notation is generalized to other circuit and complexity classes in the obvious way. Other notions of uniformity are discussed in Chapter 6.

Note that the circuit classes defined above are used to classify *languages.* Occasionally, however, the names of circuit classes are used to describe circuit families. For example, an AC^0 family of circuits is a family of unbounded fan-in, bounded-depth and polynomial size circuits. Or, for instance, a set has "NC^1 circuits" if the set is in NC^1.

As mentioned in the introduction, this thesis studies ways for quantifying the descriptional complexity of a circuit. The discussion of such matters as well as the introduction of other models of non-uniform computation are deferred to Chapter 2.

More information on circuits and Boolean functions can be found in [66].

1.2.3 Functions and constructibility

In this thesis, $\log n$ will be used to denote the integer-valued function $\lceil \log_2 n \rceil$ whose value at 0 is defined to be 0. More generally, integer-valued functions will primarily be discussed. In particular, for every real-valued function f, $f(n)$ will typically denote $\lceil f(n) \rceil$.

The reader is assumed to be familiar with "Big-O" notation and related "order-of-magnitude" notations. Recall that $f(n) \in O(g(n))$ if there exist natural numbers c and N such that $f(n) \leq cg(n)$ for all $n \geq N$, while $f(n) \in o(g(n))$ if,

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0.$$

Recall that $f \in \Omega(g(n))$ if and only if $g(n) \in O(f(n))$ and $f(n) \in \omega(g(n))$ if and only if $g(n) \in o(f(n))$. Occasionally sets of the form $\bigcup O(f(n)^{cg(n)})$, where the union is over all natural numbers c, are denoted by $f(n)^{O(g(n))}$

Certain sets of functions recur throughout this thesis. The sets $\log^{O(1)} n$, $n^{o(1)}$, $o(n)$, $n \log^{O(1)} n$ and $\omega(n)$ are called the sets of *polylogarithmic*, *subpolynomial*, *sublinear*, *quasilinear* and *superlinear* functions, respectively. Note that f is subpolynomial if and only if $f(n^k) \in o(n)$ for all $k \in \mathbb{N}$. This alternate characterization of subpolynomial functions is repeatedly exploited in this thesis. Note also that the product of two subpolynomial functions is also subpolynomial. Examples of subpolynomial functions are $\log^k n$ and $\log^{\log \log n} n$. Note that

$$\log^{O(1)} n \subseteq n^{o(1)} \subseteq o(n) \subseteq n \log^{O(1)} n \subseteq \omega(n).$$

Observer that the inclusions are all proper.

Occasionally, several strings will need to be efficiently encoded into one string. Define $\langle x, y \rangle$ to be the string formed by doubling each bit in x and appending $01y$ to the resulting string. Note that $|\langle x, y \rangle| = 2|x| + |y| + 2$. Note also that computing $\langle , \rangle$ and its inverse can be done in linear time and constant auxiliary space.

Throughout this thesis, only functions that have some form of time-constructibility are considered. The one exception is in Chapter 3 where, consequently, all constructibility assumptions are explicitly mentioned in the statements of all results. Recall that a function f such that $f(n) \in \Omega(\log n)$ is *time-constructible* if given an input of length n, $f(n)$ can be written in binary in deterministic $O(f(n))$ time. More generally, a function f is called *t-time-constructible* if given an input of length n, $f(n)$ can be written in binary in deterministic $O(t(n))$ time. Most natural functions are time-constructible. Indeed, polynomials with rational exponents enjoy an even stronger property than time-constructibility:

Proposition 1.1 *Given an input of length n, the number n in binary is constructible in linear time and logarithmic auxiliary space.*

Proof. This is Proposition 1 in Kannan [43]. ■

Proposition 1.2 *For all rational numbers $r > 0$, the number n^r in binary is constructible from an input of length n in linear time and logarithmic auxiliary space.*

Proof. The proof uses standard arguments; a similar result appears in [43]. Let $r = k/\ell$ where k and ℓ are integers. Construct n in linear time and logarithmic space with the aid of Proposition 1.1. Then compute n^k (in binary) in time polynomial with respect to $\log n$ by repeated multiplication. Note that the resulting number (in binary) has length $O(\log n)$. Now use binary search on the numbers between 1 and n^k to find $n^{k/\ell}$. That the latter can be done using logarithmic space with respect to n and time polynomial with respect to $\log n$ is clear. ■

These strong constructibility properties of polynomials with rational exponents are exploited in Chapter 5 in this thesis.

1.2.4 Boolean formulas, satisfiability and nondeterministic computation

Recall that a quantified Boolean formula is a formula with an initial block of quantifiers followed by a quantifier-free formula. Let $\mathrm{QBF}_{a(n)}$ be the set of true quantified formulas having at most $a(n) - 1$ quantifier alternations beginning with $\exists$. Recall that $\mathrm{QBF}_{a(n)}$ is complete for $\Sigma^p_{a(n)}$.

Definition 1.1 *The* size *of a quantified Boolean formula is the number of occurrences of variables it contains.*

It will be useful to encode formulas over $\{0, 1\}$. The standard way of doing this is by describing each literal in a formula of size n using a bit string of length $\log n + 1$. By assuming in addition that all encodings begin with 1, the encoding of a formula can thereby be padded by prefixing the standard encoding with a string of 0's. This padding property will frequently be exploited in this thesis.

Definition 1.2 *Given a representation of a Boolean formula in the above paddable system, the* length *n of the (encoded) formula.*

In particular, a quantified Boolean formula will therefore be said to have length n if it can be expressed using the paddable encoding system with n bits. Note then that a formula can therefore have multiple lengths. Indeed, a formula can have any length greater than or equal to the length of its smallest encoding in the above paddable encoding system.

In the sequel, unless explicitly stated otherwise, any reference to a quantified Boolean formula is in fact a reference to a bit string encoding the quantified Boolean formula in question. Note that Boolean formulas will sometimes be referred to as propositional formulas.

To obtain lower bounds for SAT, a lemma showing that nondeterministic computations can be efficiently reduced to SAT questions is required. Lemma 1.3 below will serve in this regard. Note that the arguments in [46] also seem to require Lemma 1.3 (or at least the weaker version Lemma 1.1 given below) to obtain lower bounds for SAT. Indeed, Lipton and Viglas obtain time-space tradeoffs for $\mathsf{NTIME}(n)$ in [46], but do not explicitly show how these imply a lower bound for SAT. Lemma 1.3 can be used to provide the missing argument.

Of course, the first result showing a tight relationship between SAT and nondeterministic computation is Cook's Theorem [18]. Later, Cook [22] refined his original result by showing that a nondeterministic multi-tape Turing machine computation of time $t(n)$ can be reduced to a SAT question of size $O(t(n) \log t(n))$. Cook's result built on work by Hennie and Stearns [34] and Pippenger and Fischer [52] showing that a deterministic multi-tape Turing machine computation of time $t(n)$ can be simulated by a family $\{C_n\}$ of circuits with $O(t(n) \log t(n))$ gates. Cook's result can be extended to alternating machines making a constant number of alternations:

Lemma 1.1 *Let M be a $t(n)$ time-bounded alternating (non-random-access) multi-tape Turing machine making $a \geq 1$ alternations on inputs of length n. Then there exists a function ϕ mapping inputs x, $|x| = n$, to quantified propositional formulas of length $O(t(n) \log^2 t(n))$ such that M accepts x if and only if $\phi(x)$ is in* QBF_a. *Moreover, ϕ is computable in $O(t(n) \log^2 t(n))$ time and $O(\log t(n))$ space, and a machine that has random-access to x can compute the ith bit of $\phi(x)$ in $O(\log^{O(1)} t(n))$ time and $O(\log t(n))$ space.*

In [55] Robson showed that nondeterministic RAM computations of time $t(n)$, where the time complexity is measured using the logarithmic cost criterion, can be reduced to SAT questions of size $O(t(n) \log t(n))$. Robson's proof can be adapted to show that for each $t(n)$ time-bounded nondeterministic random-access Turing machine M there exists a function ϕ mapping inputs x, $|x| = n$, to propositional formulas of size $O(t(n) \log t(n))$ such that M accepts x if and only if $\phi(x)$ is in SAT. For the results in this thesis it will be necessary that the formulas $\phi(x)$ be easy to compute in the sense that a machine with random access to x can compute the ith bit of $\phi(x)$ using polylogarithmic time and logarithmic space. Now, the formulas in Robson's reduction consist of parts representing the computation of M together with a component for asserting that a given list L_2 is the stable sort of another list L_1. Whereas it is easy to see that the parts of Robson's formulas not concerned with the stable-sort are easy to compute in the sense identified above, it is not clear that the parts of the formulas representing the stable sort are sufficiently easy to compute.

The stable sort component of a formula in Robson's reduction has size $O(t(n) \log t(n))$. This component can be replaced by a new formula of size $O(t(n) \log^2 t(n))$ that is sufficiently easy to compute in the sense required above as follows. Consider a multi-tape Turing machine M' that accepts if and only if its input consists of two lists where the second list is a merge-sort of the first list. Such a machine can be made to run in $O(n \log n)$ time. Lemma 1.1 then implies the existance of a function ϕ' mapping inputs y, $|y| = n$, to propositional formulas of size $O(n \log n)$ such that $\phi'(y)$ is in SAT if and only if y consists of two lists where the second list is a merge-sort of the first list. Moreover, if one has random access to y, the ith bit of $\phi'(y)$ can be computed using polylogarithmic time and logarithmic space. Noting that merge-sort is a stable sort, the following lemma can be proved by replacing in Robson's reduction the part representing a stable sort with the formula obtained via ϕ':

Lemma 1.2 *Let M be a $t(n)$ time-bounded nondeterministic random-access Turing machine. Then there exists a function ϕ mapping inputs x, $|x| = n$, to propositional formulas of length $O(t(n) \log^3 t(n))$ such that M accepts x if and only if $\phi(x)$ is in* SAT. *Moreover, a machine that has random-access to x can compute the ith bit of $\phi(x)$ in $O(\log^{O(1)} t(n))$ time and $O(\log t(n))$ space.*

For completeness, an outline of Robson's proof as applied to random-access Turing machines now follows; for the precise details the reader may consult Robson's original paper [55]. Given a random-access nondeterministic Turing machine M that runs in time $t(n)$ and an input x, a formula $\phi(x)$ of size $O(t(n) \log^2 t(n))$ will be constructed such that the formula is satisfiable if and only if M accepts x.

Moreover, it will be clear that the formula will be easily computable as in the sense of the statement of Lemma 1.2. As in the proof of Cook's Theorem, the formula's variables will be used to represent an accepting computation path of machine M on input x.

First assume, without loss of generality, that for each non-index tape T, M either writes or reads to T in a given step, but never does both. The formula $\phi(x)$ will have variables and clauses for describing the internal workings of M. For instance, $\phi(x)$ will have $O(t(n)\log t(n))$ variables describing the contents of the index tapes of M, and it will have $O(t(n)\log t(n))$ variables describing the values read and stored at each step. Call these variables the STORE variables. Note that the number of variables is $O(t(n)\log t(n))$ since the machine runs for $O(t(n))$ time and the addresses have length $O(\log t(n))$. In addition, $O(t(n))$ variables are required to describe the internal state of the machine at each step and to ensure that that the transitions between steps are valid. It remains to describe, using only $O(t(n)\log^2 t(n))$ variables, that the sequence of characters asserted by the STORE variables to have been read and stored at each step is consistent.

Briefly, this is done as follows. Whenever a value is read or stored to the kth square of the jth tape at time i, a record $R(i,j,k)$ is defined to indicate the value read or stored and to specify whether a store or read was done. Note that there are $O(t(n)\log t(n))$ such records since there are $O(t(n))$ time units and the tape addresses can be specified using $O(\log t(n))$ bits. Now consider the chronological list L_1 of these records R during an accepting computation of M on input x. Note that L_1 contains $t(n)$ records of size $O(\log t(n))$. Assuming that these records are consistent with the actual contents of the non-index tapes during an accepting computation of M on input x (how this is asserted will be described subsequently), one can then use these records to construct the formula $\phi(x)$ that is true if and only if there exists an accepting path for M on input x. It remains to show how to assert that the records in L_1 do in fact represent a consistent view of the non-index tapes during an accepting computation of M on input x. This can be done by first doing a stable sort on the records in L_1 using the address as a key to obtain a list L_2. Note that in L_2, all records corresponding to the same tape square on a non-index tape are adjacent; moreover, since the sort is stable, these records are in chronological order. Therefore, to check the consistency of L_1, it suffices to check that adjacent records in L_2 are consistent. In particular, it suffices to check that a record has value 0 if its address is different from the previous record's address; or if the adjacent records correspond to the same tape square, it suffices to check that the value read in the later record corresponds to the value read or stored in the previous record. Since L_1 can be described by $O(t(n)\log t(n))$ variables, by the discussion preceding the statement of Lemma 1.2, a formula of size $O(t(n)\log^3 t(n))$ can be used to assert that L_2 is a stable sort of L_1 using address as key. Therefore using $O(t(n)\log^3 t(n))$ variables, the formula $\phi(x)$ can express that

- there exists a list L_1 of records in chronological order representing the accesses to the non-index tapes during an accepting computation of M on input x (L_1 is used by the remainder of $\phi(x)$ to assert that there exists an accepting computation of M on input x);

- L_2 is the stable sort of L_1 using address as key; and

- adjacent records in L_2 are consistent.

In fact, by appropriately breaking up the list L_1 into smaller lists of exponentially increasing size, $\phi(x)$ can express the above with only $O(t(n)\log^2 t(n))$ variables by using several stable sort formulas to sort the sublists of L_1 (see Robson's original paper [55] for details). Moreover, the resulting formula $\phi(x)$ is still sufficiently easy to compute in the sense required by the statement of Lemma 1.2. This completes the sketch of the proof of Robson's result for random-access Turing machines. For more details, the reader is referred to Robson's paper [55].

More generally, and by appropriately padding formula encodings, the following extension to the above lemma can be proved. Note also that the following lemma is the random-access Turing machine analogue of Lemma 1.1.

Lemma 1.3 *Let M be a $\Sigma_a^{t(n)}$ machine where $a \geq 1$. Then there exist a constant D and a function ϕ mapping inputs x, $|x| = n$, to quantified propositional formulas of length $Dt(n)\log^3 t(n)$ such that M accepts x if and only if $\phi(x)$ is in* QBF_a. *Moreover, a machine that has random-access to x can compute the ith bit of $\phi(x)$ in $O(\log^{O(1)} t(n))$ time and $O(\log t(n))$ space.*

There is a subtle point that must be addressed when making the extension from nondeterministic machines to Σ_a machines in Lemma 1.3. Note that the formulas ϕ constructed in Robson's result (and in Cook's original result) essentially have three type of variables: the first type of variables represent the input to the nondeterministic machine; the second type represent the nondeterministic choices made by the machine on an accepting path; and the third type represent the variables used to simulate the computation of the nondeterministic machine. Hence, the formula ϕ constructed for a nondeterministic machine M on input x is of the form $\phi_{M,x}(y,z)$ where M accepts x if and only if there exist nondeterministic choices y and settings z of the variables representing the computation of M such that $\phi_{M,x}(y,z)$ is true. Now, if the techniques used to obtain Robson's and Cook's lemmas are directly used to obtain an analogous result for, say, Σ_2 computations, one obtains that for every Σ_2 machine M, M accepts x if and only if there exists nondeterministic choices y such that for all co-nondeterministic choices z there exist settings of the variables v representing the computation of M such that $\phi_{M,x}(y,z,v)$ is true. But this is not a QBF_2 formula; indeed, it is a QBF_3 formula! The way around this problem is not to change the order of the quantifiers since the resulting formula will no longer represent the computation of the Σ_2 machine. However, the (innermost) quantifier for the variables v representing the computation can be reversed if needs be as follows: the formula is altered so that now the formula is not true if there exist settings of the variables v so that they represent an accepting computation; instead, the formula is true if for all settings of the variables v representing the computation, the computation represented is an accepting one. This argument can be generalized to Σ_a machines where $a \geq 1$.

Chapter 2

Non-uniform complexity classes

In this chapter the models of non-uniform computation that are used in this thesis are introduced and analyzed. Section 2.1 introduces the concept of a non-uniform Turing machine. The classes defined using such machines are subsequently compared with non-uniform classes defined using Karp-Lipton advice. The relative merits of using non-uniform Turing machines and Karp-Lipton advice to describe non-uniform classes is discussed.

In section 2.2 the concept of using a (simpler) circuit to describe a circuit is discussed. In particular, a novel way for accomplishing this is discussed. The relations of this new technique to other types of circuit descriptors used in the literature are briefly examined.

Finally, in section 2.3 classes of languages accepted by circuits of a specific size and specific descriptive complexity are related to classes defined using non-uniform Turing machines. This makes explicit results that were implicit in work done by Hermo and Mayordomo [36] and Hermo [35] while precisely characterizing classes such as $\mathsf{P}/O(\log n)$ using circuits.

2.1 Non-uniform Turing machines and Karp-Lipton advice

In [45] Karp and Lipton proposed an elegant way of defining non-uniform complexity classes using uniform classes and "advice" function which provide the "non-uniformity".

Definition 2.1 (Karp-Lipton) *Let $\mathcal{C}$ be a class of languages and let $\mathcal{F}$ be a collection of functions mapping $\mathbb{N}$ to $\mathbb{N}$. A language L belongs to the non-uniform class $\mathcal{C}/\mathcal{F}$ if there exist a language L' in $\mathcal{C}$ and a function $h : \mathbb{N} \to \Sigma^*$ such that $|h| \in \mathcal{F}$ and $x \in L$ if and only if $\langle x, h(|x|)\rangle \in L'$.*

The functions $\mathcal{F}$ in the above definition are sometimes called *advice functions*. In the sequel, the set $\mathcal{F}$ in a class $\mathcal{C}/\mathcal{F}$ is restricted to time-constructible functions. Moreover, it is assumed that if $f \in \mathcal{F}$, then all time-constructible functions g such that $g \leq f$ pointwise are also in $\mathcal{F}$. By abusing notation, $\mathcal{C}/a(n)$ will be used to denote the class $\mathcal{C}/\mathcal{F}$ where $\mathcal{F}$ denotes all time-constructible functions g such that $g \leq a$ pointwise. Subsequent chapters of this thesis will be primarily interested in non-uniform classes of the form $\mathcal{C}/n^{o(1)}$.and $\mathcal{C}/o(n)$.

An alternate way to define non-uniform complexity classes is to use Turing machines which are provided with an extra read-only tape whose contents depend only on the input length. This concept sounds very similar to the Karp-Lipton methodology; however, there are some subtle differences that will be discussed as soon as the formal definitions are given.

Definition 2.2 *A non-uniform Turing machine $M^{(?)}$ is a Turing machine that is provided with an extra one-way infinite read-only random-access advice tape. Fix a function $g : \mathbb{N} \to \Sigma^*$. The machine $M^{(g)}$ computes like a normal Turing machine except that on inputs of length n, and only on inputs of length n, it is provided with the string $g(n)$ at the start of its advice tape followed by an infinite sequence of blank symbols.*

Time and space complexity for non-uniform Turing machines are defined analogously as for ordinary Turing machines. A machine $M^{(?)}$ is *robust* if the time and space complexity of the machine do not depend on the advice function $g : \mathbb{N} \to \Sigma^*$ "plugged into" $M^{(?)}$. Note that this notion of robustness is not related to Schöning's notion of robustness for oracle machines [57]. Assuming that a non-uniform machine is robust rarely results in loss of generality. Hence, all non-uniform machines used in this thesis are assumed to be robust unless otherwise specified. Define the class $\mathsf{DTIME}(t(n))^{(\mathcal{F})}$ to be the class of languages accepted by $t(n)$ time-bounded deterministic non-uniform Turing machines $M^{(g)}$ where $|g|$ is bounded by some function in $\mathcal{F}$. Classes such as $\mathsf{NTIME}(t(n))^{(\mathcal{F})}$, $\mathsf{P}^{(poly)}$, etc., are defined analogously. In the sequel, the set $\mathcal{F}$ in a class $\mathcal{C}^{(\mathcal{F})}$ is restricted to time-constructible functions. Moreover, it is assumed that if $f \in \mathcal{F}$, then all time-constructible functions g such that $g \leq f$ pointwise are also in $\mathcal{F}$. By abusing notation, $\mathcal{C}^{(a)}$ will be used to denote the class $\mathcal{C}^{(\mathcal{F}}$ where $\mathcal{F}$ denotes all time-constructible functions g such that $g \leq a$ pointwise.

To avoid confusion in the sequel, it is important to note that if M is a Turing machine, the superscript function g in the notation $M^{(g)}$ is an advice function mapping integers to strings; whereas if $\mathcal{C}$ is a complexity class, the superscript function in the notation $\mathcal{C}^{(a)}$ is a bound function mapping integers to integers.

While the Karp-Lipton definition presents a convenient way for quantifying the amount of non-uniformity in a class, it does have certain drawbacks. First of all note that the definition seems to depend on the chosen pairing function. This is a particular shortcoming when the class $\mathcal{C}$ in $\mathcal{C}/\mathcal{F}$ is of the form $\mathsf{DTIME}(T)$ where $T(n) \in o(T(n+1))$. Another drawback of the Karp-Lipton definition is that it occasionally defines counter-intuitive and rather meaningless classes. For instance, every language over $\{0,1\}$ is in $\mathsf{P}/2^n$. Moreover, the notation $\mathsf{P}/2^n$ is rather misleading since languages in $\mathsf{P}/2^n$ are in fact computed by exponential-time Turing machines! Furthermore, the Karp-Lipton definition allows bizarre equalities such as $\mathsf{P}/poly = \mathsf{DTIME}(n)/poly$ containing no theoretical insight.

Defining complexity classes in terms of non-uniform Turing machines, however, avoids these pitfalls. For instance, the definition of $\mathsf{DTIME}(t(n))^{(\mathcal{F})}$ does not depend upon any encoding. Moreover, while it is not clear whether $\mathsf{P}^{(poly)} = \mathsf{DTIME}(n)^{(poly)}$, unlike with the analogous Karp-Lipton advice classes, determining the relation between these two classes is an interesting question. In fact, assuming that $\mathsf{DTIME}(t(n))^{(poly)} \subsetneq \mathsf{DTIME}(T(n))^{(poly)}$ whenever T is time-constructible and $t(n) \log t(n) \in o(T(n))$, one can easily show using the techniques in the proof of Theorem 4.1 that SAT does not have *completely non-uniform* NC^1 circuits of quasilinear size. This would be a major breakthrough. Note however, that in the special case where the set *poly* is replaced by a single function g, it is shown in Chapter 3 that $\mathsf{P}^{(g)} \neq \mathsf{DTIME}(n)^{(g)}$ and $\mathsf{P}/g \neq \mathsf{DTIME}(n)/g$.

Notwithstanding the subtle differences in defining non-uniform classes using Karp-Lipton advice functions and non-uniform Turing machines, the following tight relationship between time classes defined using either definition can be shown.

Theorem 2.1 *Let $t(n)$ and $a(n)$ be such that $t(O(n + a(n))) \subseteq O(t(n))$. Then*

$$\mathsf{DTIME}(t(n))/a(n) = \mathsf{DTIME}(t(n))^{(a)}.$$

Proof. Suppose that L is in $\mathsf{DTIME}(t(n))/a(n)$. Then there exist a $t(n)$ time-bounded deterministic Turing machine M and a function $h : \mathbb{N} \to \Sigma^*$ such that $|h| \leq a$ and $x \in L$ if and only if M accepts $\langle x, h(|x|)\rangle$. Then L is in $\mathsf{DTIME}(t(n))^{(a)}$ via a machine $M'^{(h)}$ that simulates M on $\langle x, h(|x|)\rangle$. Note that M' runs in $O(t(n))$ time since M is $O(t(n))$ time-bounded and $t(O(n + a(n))) \subseteq O(t(n))$.

Now suppose that L is in $\mathsf{DTIME}(t(n))^{(a(n))}$. Then there exist a non-uniform machine $M^{(?)}$ and a function $h : \mathbb{N} \to \Sigma^*$ such that $|h| \leq a$ is time-constructible, $M^{(h)}$ is $t(n)$ time-bounded, and $M^{(h)}$ accepts L. Now consider a machine M' that on input $\langle x, y\rangle$ simulates $M^{(h)}$ on x. Note that M' runs in $O(t(n))$ time and that $x \in L$ if and only if M' accepts $\langle x, h(|x|)\rangle$. ■

It is easy to see that this theorem generalizes to complexity classes defined in terms of more general models of computation, such as alternating machines, as well as generalizing to other types of resource-bounded classes.

Most of the non-uniform classes discussed in this thesis are of the type given in the hypotheses of the above theorem. Consequently, most of the results in this thesis are transferable to either definition. In general, however, non-uniform classes are mostly defined using the Karp-Lipton methodology in this thesis since this is the standard approach in the literature. Only in Chapter 3 were more precision is required will non-uniform Turing machines be extensively used.

2.2 Circuit descriptors

In Hermo and Mayordomo [36] and Hermo [35], Karp-Lipton advice classes with sublinear advice were characterized in terms of circuits with a particular amount of Kolmogorov complexity. This work was motivated through the effort to prove a converse to a theorem proved by Karp and Lipton in [45] showing that P/log consists of all languages with polynomial-size circuits having "easy descriptions". In this section, a new way of describing the complexity of a circuit is introduced and compared with the "easy descriptions" of Karp and Lipton. This will involve revisiting some of the ideas from the proofs in [36] and [35].

Fix a paddable encoding w of circuits by strings over $\{0, 1\}$ and consider a circuit C with n inputs such that $|w(C)| = s(n)$. Then a *descriptor* or *description* of C is a circuit C' with $\log_2 s(n)$ inputs such that C' on input i (in binary) outputs the ith bit of $w(C)$. This definition is due to Charles Rackoff [53].

Karp and Lipton [45], give a different definition for the descriptor/description of a circuit. There a description of a circuit C is defined as a circuit C' that recognizes the direct connection language of C. That is, when C' is given the input $\langle i, b, j, k\rangle$, it returns 1 if and only if the gate i in C has inputs j and k and performs the functions coded by b. To differentiate, let us refer to descriptors as defined by Rackoff as *R-descriptors* and the descriptors as defined by Karp and Lipton as *KL-descriptors*. A family $\{C_n\}$ of circuits has *R-descriptors* $\{C'_n\}$ (respectively *KL-descriptors* $\{C''_n\}$) if for all n, C'_n (C''_n) is an R-descriptor (KL-descriptor) for C_n.

The following theorem relates R-descriptors and KL-descriptors.

Theorem 2.2 *A language A has polynomial-size circuits with R-descriptors of size $O(d(n))$ if and only if A has polynomial-size circuits with KL-descriptors of size $O(d(n))$.*

The proof of the theorem, relies on an idea exploited by Hermo and Mayordomo [36] to show that the languages recognized by polynomial size circuits with logarithmic size KL-descriptors are precisely those languages in P/log. Hermo and Mayordomo take advantage of the Shannon-Lupanov effect which essentially states that the interconnections between gates in a circuit can be used to store a non-trivial amount of information. More specifically, they take advantage of the following theorem whose proof can be found in Wegener [66].

Theorem 2.3 *Given a boolean function $f : \{0,1\}^m \to \{0,1\}$ the smallest circuit computing f is less than or equal to $2^m/m + o(2^m/m)$.*

So arguing as in the proof of Theorem 2.6 in Hermo [35] and exploiting Theorem 2.3 one can show that a language A has circuits of polynomial-size with R-descriptors of size $O(d(n))$ if and only if A is in $\mathsf{P}/O(d(n)\log d(n))$ if and only if A has circuits of polynomial-size with KL-descriptors of size $O(d(n))$. Theorem 2.2 now follows.

In fact, the arguments in Hermo [35] can be used to show that if a language A has circuits of size $t(n)$ with R-descriptors of size $d(n)$ then A has circuits of size $O(t(n)^2)$ with KL-descriptors of size $O(d(n))$; and if a language A has circuits of size $t(n)$ with KL-descriptors of size $d(n)$ then A has circuits of size $O(t(n)^2)$ with R-descriptors of size $O(d(n))$. It seems likely that by carefully analyzing the techniques in Hermo's proof and employing the construction by Pippenger and Fischer [52] showing that (non-random-access) multi-tape Turing machine computations of time $t(n)$ have circuits of size $O(t(n)\log t(n))$, one could prove that if a language A has circuits of size $t(n)$ with R-descriptors of size $d(n)$ then A has circuits of size $O(t(n)\log t(n))$ with KL-descriptors of size $O(d(n))$. An interesting problem is to determine whether circuits of size $t(n)$ with KL-descriptors of size $d(n)$ recognize the same class of languages as circuits of size $t(n)$ with R-descriptors of size $d(n)$.

Observe that a language L has polynomial-size circuits with logarithmic size R-descriptors if and only if the language has polynomial-size circuits with logarithmic size KL-descriptors. Karp and Lipton refer to the latter condition as meaning that L has polynomial-size circuits with *easy descriptions*.

R-descriptors are of particular interest due to the following theorem by Rackoff [53].

Theorem 2.4 (Rackoff) *Let $t(n)$ and $T(n)$ be such that $(n\log n)(t(n)\log t(n))$ and $t(n)\log^2 t(n)$ are in $O(T(n))$, and such that $t(n)$ is $O(T(n))$-time-constructible. Then there exists a language L in $\mathsf{DTIME}(T(n))$ that cannot be computed by a family of circuits of size at most $t(n)$ with R-descriptors having encodings of size $\leq n$. In fact, for no n is there a circuit of the stated size and R-descriptor size which recognizes $L \cap \{0,1\}^n$.*

Proof. Let $L = \{x : x,\ |x| = n$, is a (possibly padded) encoding of a circuit C' which is an R-descriptor for a circuit C on n inputs such that C has size bounded by $t(n)$ and C outputs false on $x\}$. The circuit lower bound for L is obvious. So it suffices to show that L is in $\mathsf{DTIME}(T(n))$.

Note first that using standard techniques, a circuit with n gates, i.e., a circuit with a standard encoding of length $O(n \log n)$, can be evaluated in time $O(n \log^2 n)$. This fact will be used several times. Now for a deterministic machine M to check whether x is in L, it first constructs $t(n)$ which can be done in time $O(T(n))$. Then the machine decodes C'; this takes linear time. Note now that since C is supposed to have at most $t(n)$ gates, it follows that the standard encoding of C should thereby have $dt(n) \log t(n)$ bits where the constant d depends only on the encoding system used and not on the circuit being encoded. Hence, C' must be evaluated $dt(n) \log t(n)$ times to obtain the standard description of a circuit of size $t(n)$. So M evaluates C' $dt(n) \log t(n)$ times on consecutive inputs until it either runs out of inputs for C' or it obtains the description of a circuit C with at most $t(n)$ gates, whichever comes first. How long does this take? Since the circuit C' has an encoding of length n, it can be evaluated in time $O(n \log n)$ by the remarks above. So to obtain the description of C takes $O(n \log n \cdot t(n) \log t(n)) \subseteq O(T(n))$ time. Lastly, M evaluates C on input x. By the above remarks, and since C has at most $t(n)$ gates, this can be done in time $O(t(n) \log^2 t(n)) \subseteq O(T(n))$. Hence, L is in $\mathsf{DTIME}(T(n))$. ■

Similar arguments to Theorem 2.4 are used to obtain some of the results in Chapter 3. In light of Theorem 2.4, the sequel will mainly be concerned with the R-descriptors of a circuit. In particular, "descriptors" will henceforth refer exclusively to R-descriptors unless otherwise indicated. A circuit will be said to have *descriptive size* $d(n)$ if it has an R-descriptor of size $d(n)$.

It will be useful to introduce notation for versions of the standard circuit classes, such as AC^0, ACC^0, TC^0 and NC^1, which incorporate the notion of descriptive size.

Definition 2.3 *Let $\mathcal{F}$ be a class of functions from $\mathbb{N}$ to $\mathbb{N}$. Then $\mathsf{AC}^0_{\mathcal{F}}$ denotes the class of languages accepted by AC^0 circuits with descriptors of size $d(n)$ where $d \in \mathcal{F}$.*

The classes $\mathsf{ACC}^0_{\mathcal{F}}$, $\mathsf{TC}^0_{\mathcal{F}}$ and $\mathsf{NC}^1_{\mathcal{F}}$ are defined analogously. Occasionally, circuit classes of the type $\mathcal{C}^0_{\{d(n)\}}$ will be denoted by $\mathcal{C}_{d(n)}$. In the sequel, the classes $\mathsf{TC}^0_{n^{o(1)}}$ and $\mathsf{NC}^1_{n^{o(1)}}$ will be of particular importance.

The topic of defining descriptors for circuits of unbounded fan-in will be revisited and discussed in much more detail in Chapter 6 where certain extra subtle points in the definition will be considered.

2.3 Circuit descriptive size versus advice

An interesting problem is to determine the relationships of classes of languages accepted by circuits of a given size and descriptive size with classes defined by non-uniform Turing machines or using Karp-Lipton advice. Partial answers to these questions already exist. For instance, the class $\mathsf{P}/poly$ corresponds precisely with all languages accepted by circuits of polynomial size. Hermo and Mayordomo [36] showed that P/log corresponds to polynomial size circuits in $\mathsf{K}[log, poly]$, where K stands for Kolmogorov and

$$\mathsf{K}[f(n), t(n)] = \{x : \exists y, |y| \leq f(n) \text{ and } U(y) = x \text{ in at most } t(n) \text{ steps}\},$$

and U is a fixed universal Turing machine (UTM). Hermo and Mayordomo also show that the class of all languages recognized by polynomial size circuits with logarithmic size KL-descriptors corresponds precisely with $\mathsf{P}/O(\log n \log \log n)$, a result used above in the proof of Theorem 2.2.

In this section theorems relating classes of languages accepted by circuits of given size and descriptive size with classes of languages accepted by time-bounded deterministic Turing machines with advice are given. The results presented are implicit in the results of Hermo and Mayordomo [36] and Hermo [35] but are not explicitly stated in the forms given here.

Theorem 2.5 *Let L be accepted by a family $\{C_n\}$ of circuits of size $s(n)$ having descriptors of size $d(n)$. Then L is accepted by a $O(s(n)\log s(n)d(n)\log^2 d(n) + s(n)\log^2 s(n))$ time-bounded deterministic non-uniform Turing machine $M^{(g)}$ where $|g(n)| \in O(d(n)\log d(n))$. Moreover, machine M need not have random access.*

Proof. Fix a standard encoding of circuits such that a circuit with k gates has an encoding of length $O(k\log k)$. Let $g(n)$ be the encoding of the descriptor D_n for C_n. Note that since D_n has $d(n)$ gates, the length of the encoding is in $O(d(n)\log d(n))$.

On input x of length n, $M^{(g)}$ evaluates the circuit D_n encoded on its advice tape $O(s(n)\log s(n))$ times to obtain the standard encoding of C_n and then evaluates C_n on x. As noted in the prof of Theorem 2.4, using standard techniques, a circuit with n gates can be evaluated on a standard (non-random-access) multi-tape Turing machine in $O(n\log^2 n)$ time. Thus, it takes $O(s(n)\log s(n)d(n)\log^2 d(n))$ time to obtain the standard encoding for C_n. Moreover, after having obtained the encoding of C_n, this circuit, which has $s(n)$ gates, can then be evaluated in time $O(s(n)\log^2 s(n))$. The theorem now follows. ■

A partial converse to the above theorem can be obtained for Karp-Lipton advice classes where the uniform class is defined using a standard multi-tape Turing machine.

Theorem 2.6 *Suppose that $L \subseteq \{0,1\}^*$ is in $\mathsf{SEQ\text{-}T\text{-}DTIME}(t(n))/a(n)$, where $t(n) \geq n$ is time-constructible, and $\mathsf{SEQ\text{-}T\text{-}DTIME}(t(n))$ is the class of languages accepted by $O(t(n))$ time-bounded multi-tape Turing machines. Then L has circuits of size $O(t(n)\log t(n) + a(n))$ with descriptors of size $O(\log^{O(1)} t(n) + a(n))$.*

Proof. By the hypotheses, there exist a $O(t(n))$ time-bounded multi-tape Turing machine M and a function $g : \mathbb{N} \to \{0,1\}^*$ such that $|g| = a$ and x is in L if and only if M accepts $\langle x, g(|x|)\rangle$. The techniques in Pippenger and Fischer [52] can then be used to construct a circuit C_n of size $O(t(n)\log t(n))$ and with $n+a(n)$ input bits such that C_n on input $xg(n)$, where $|x| = n$, accepts if and only if x is in L. Note that the constructibility of t is needed to obtain these circuits. Note also that the circuits resulting from the Pippenger and Fischer [52] construction are very regular. Indeed, it is easy to see that given an integer i of length $\log t(n)$ in binary, the ith bit of the encoding of C_n can be constructed in polylogarithmic time. It follows that the circuits $\{C_n\}$ have descriptors of size $O(\log^{O(1)} t(n) + a(n))$, where the extra $a(n)$ gates are used to store the advice $g(n)$ for inputs of length n. ■

Note that by using the arguments used by Hermo and Mayordomo to show that $\mathsf{P}/O(\log n \log\log n)$ consists of all languages accepted by polynomial size circuits with descriptors of logarithmic size, the descriptors in the above theorem can be made to be of size $s(n)$ where,

$$s(n)\log s(n) = \log^{O(1)} t(n) + a(n).$$

It remains open to determine if the above inclusions are the tightest possible.

Chapter 3

Hierarchies of non-uniform complexity classes

In this chapter, provable hierarchies of non-uniform classes are presented. In particular, a new proof of a result in [45] as well as some new separations are given. These results generalize the standard time and space hierarchy theorems [19, 33, 58, 70].

In section 3.1 a new proof is given of a result of Karp and Lipton [45] showing that if $f < g$ i.o., then $\mathsf{P}/f \subsetneq \mathsf{P}/g$. More generally, if f and g are functions mapping $\mathbb{N}$ to $\mathbb{N}$ such that $f(n) < g(n) \leq 2^n$ i.o. and f is time-constructible, then there exist a function $h : \mathbb{N} \to \{0,1\}^*$, $|h| = g$, and a deterministic Turing machine $U^{(h)}$ running in time $g(n)$ such that for all functions $h' : \mathbb{N} \to \{0,1\}^*$, $|h'| \leq f$, no Turing machine $M^{(h')}$ accepts $L(U^{(h)})$. The new proof is given since the underlying idea, based on the argument used by Kannan in [42] to show that $\mathsf{DSPACE}(n^{\omega(1)})$ contains languages not in $\mathsf{P}/poly$, is used extensively in the following sections.

In section 3.2 new time hierarchies are shown for classes defined in terms of time-bounded non-uniform Turing machines. In particular, it is shown that if $t(n)$ and $T(n)$ are functions such that $t(n) \log t(n) \in o(T(n))$, then for any function a,

$$\mathsf{DTIME}(t(n))^{(a)} \subsetneq \mathsf{DTIME}(T(n))^{(a)}.$$

Analogous results are shown for nondeterministic classes and for non-uniform classes defined using Karp-Lipton advice functions.

In section 3.3 some new separations of uniform and non-uniform classes are given. In particular, if $t(n)$ and $T(n)$ are functions such that $T(n)$ is time-constructible and $t(n) \log t(n) \in o(T(n))$, then,

$$\mathsf{DTIME}(T(n)) \not\subseteq \mathsf{DTIME}(t(n))^{(n)}.$$

This theorem is generalized to arbitrary amounts of non-uniformity at the expense of an exponential blow-up in $T(n)$ with respect to $t(n)$. This theorem generalizes the arguments used by Kannan when showing that

In this chapter, attention will be restricted, without loss of generality, to languages over $\{0,1\}$. All Turing machines defined in this chapter are tacitly assumed to be of the multi-tape random-access type, even though the results also hold for Turing machines that access their input sequentially. In this chapter, x_i^n will denote the ith string of length n according to lexicographic order, where $x_1^n = 0^n$ and $x_{2^n}^n = 1^n$.

3.1 Hierarchies obtained by increasing advice

The following hierarchy, of which a special case was presented without proof in [45], shows that increasing the amount of advice available on each input size by even one bit is enough to increase the computing power of a Turing machine. A different proof than the one given below based on Kolmogorov complexity arguments appears in the Ph.D. thesis of Hermo [35]. The new proof presented here is of interest since the arguments, based on ideas from [42], are adapted to obtain the hierarchy results in the subsequent sections.

Theorem 3.1 *Let f and g be functions mapping $\mathbb{N}$ to $\mathbb{N}$ such that $f(n) < g(n) \leq 2^n$ i.o. Then there exists a function $h : \mathbb{N} \to \{0,1\}^*$, $|h| = g$, and a robust deterministic non-uniform Turing machine $U^{(h)}$ running in time $g(n)$ such that for all functions $h' : \mathbb{N} \to \{0,1\}^*$, $|h'| \leq f$, no Turing machine $M^{(h')}$ accepts $L(U^{(h)})$.*

Proof. Consider first the case where $f(n) < g(n)$ for all n. Enumerate all Turing machines and let M_n denote the nth machine. To prove the theorem, it suffices to define an advice function $h : \mathbb{N} \to \{0,1\}^*$, $|h(n)| = g(n)$, and a deterministic Turing machine $U^{(h)}$ running in time $g(n)$ such that for any advice function $h' : \mathbb{N} \to \{0,1\}^*$, $|h'| = f(n)$, there exists an input x of length n such that $U^{(h)}$ and $M_n^{(h')}$ differ on x.

The first step is to define h. The string $h(n)$ will be used to describe a function G_n mapping $\{x_1^n, x_2^n, \ldots, x_{f(n)+1}^n\}$ to $\{0,1\}$ and defined as follows. Let $\mathcal{F}_n$ consist of all functions

$$F_n^k : \{x_1^n, x_2^n, \ldots, x_{f(n)+1}^n\} \to \{0,1\},$$

where k ranges over all bit strings of length $f(n)$, and $F_n^k(x_j^n) = 1$ if and only if $M_n^{(h')}$ accepts x_j^n when $h'(n) = k$. Note that there are $2^{f(n)}$ functions in $\mathcal{F}_n$ whereas there are $2^{f(n)+1}$ functions mapping $\{x_1^n, x_2^n, \ldots, x_{f(n)+1}^n\}$ to $\{0,1\}$. Let G_n be any function mapping $\{x_1^n, x_2^n, \ldots, x_{f(n)+1}^n\} \to \{0,1\}$ such that G_n is not in $\mathcal{F}_n$. Note that G_n can be specified by $f(n)+1$ bits: each bit indicates G_n's behaviour on the $f(n)+1$ possible inputs. Let the first $f(n)+1$ bits of $h(n)$ be the $f(n)+1$ bits specifying G_n; the remaining $g(n) - f(n) + 1$ bits in $h(n)$ can be arbitrary.

The machine $U^{(h)}$ can now be defined. On input x_i^n, $U^{(h)}$ accepts if and only if the ith bit on the advice tape is 1; otherwise, $U^{(h)}$ rejects. Note that for inputs x_i^n, $1 \leq i \leq f(n)+1$, this implies that $U^{(h)}$ accepts if and only if $G_n(x_i^n) = 1$. It is easy to see that $U^{(?)}$ is robust.

The claim is that no Turing machine taking $f(n)$ advice can accept $L = L(U^{(h)})$. Indeed, suppose to the contrary that there exist $h' : \mathbb{N} \to \{0,1\}^*$, $|h'| = f$ and a machine $M^{(h')}$ such that $M^{(h')}$ accepts L. Suppose moreover, that $M = M_n$. By definition of G_n and $\mathcal{F}_n$, there exists an input x_j^n, $1 \leq j \leq f(n)+1$, such that $F_n^{h'(n)}(x_j^n) \neq G_n(x_j^n)$. But then, by definition of U and $F_n^{h'(n)}(x_j^n)$, it follows that $U^{(h)}$ and $M^{(h')} = M_n^{(h')}$ differ on x_i^n, a contradiction.

The general case where $f < g$ i.o. now follows by adjusting the behaviour of U so that it diagonalizes against M_n on inputs of size m where m is the nth integer on which f is less than g. ∎

In particular, Theorem 3.1 implies the corollary below which was stated without proof by Karp and Lipton [45]. Note that this corollary is just Theorem 3.1 reformulated for non-uniform classes defined using Karp-Lipton advice functions.

Corollary 3.2 *Let f and g be functions mapping $\mathbb{N}$ to $\mathbb{N}$ such that $f(n) < g(n) \leq 2^n$ i.o. and g is polynomial time-constructible. Then $\mathsf{P}/g \not\subseteq \mathsf{P}/f$.*

Proof. Let $U^{(h)}$ be the robust non-uniform deterministic machine running in time $g(n)$ whose existence is guaranteed by Theorem 3.1 such that for all functions $h' : \mathbb{N} \to \{0,1\}^*$, $|h'| \leq f$, no Turing machine $M^{(h')}$ accepts $L(U^{(h)})$. It follows that $L(U^{(h)})$ is not in P/f. So to the prove the corollary, it suffices to show that $L(U^{(h)})$ is in P/g.

To that end, consider a deterministic machine M that on input w of length n first checks that the input is of the form $\langle x, y\rangle$ where $|y| = g(|x|)$ and then simulates $U^{(h')}$ on input x where $h'(|x|) = y$. Since $U^{(?)}$ is robust and runs in time $g(n)$, and since g is polynomial time-constructible, it follows that M runs in polynomial time. Observe that $U^{(h)}$ accepts x if and only if M accepts $\langle x, h(|x|)\rangle$. Hence, $L(U^{(h)})$ is in P/g, and the corollary follows. ■

3.2 Hierarchies obtained by increasing resources

A time hierarchy can be shown for deterministic classes with advice by combining the techniques of the proof of the standard deterministic time hierarchy theorem [33] with the techniques used to prove Theorem 3.1.

Indeed, as in [33], a "tape-reduction" lemma is required. The lemma is analogous to similar results proved for standard Turing machines in [34]. Note, however, that the proof for random-access Turing machines is substantially simpler than the corresponding proof for standard Turing machines.

Lemma 3.1 *A random-access Turing machine with two work-tapes can simulate a $t(n)$ time-bounded random-access Turing machine with k work-tapes in time $O(t(n)\log t(n))$.*

Proof. Let M be a $t(n)$ time-bounded random-access Turing machine with k work-tapes and consider a random-access Turing machine N with two work-tapes that simulates M as follows. One of N's tapes is used to store the contents of the index tapes of M. The other work-tape is used to store interleaved the contents of M's work tapes. Note that since M is $t(n)$ time-bounded, the length of each of its index tapes is $O(\log t(n))$, and hence, only $O(\log t(n))$ space is needed for the first work-tape.

Now, N simulates the movements of the heads on each tape of M one at a time. Alterations are made to N's first work-tape accordingly to how M adjusts the contents of its k index tapes. If in addition M wants to read from or write to a work-tape (some moves of M may just adjust the index tapes and not look at the work-tapes), say work-tape i, N first reads off the head position of tape i off its first work-tape and then copies the value read (modulo some factor due to the interleaving of the contents of the k work-tapes of M onto the second work-tape of N) onto the index tape corresponding to its second work-tape. It then can read the value that M would read off its ith tape. Or, if M wanted to write to the ith tape, N would write to the appropriate place on its second work-tape. So it takes $O(\log T)$ time to simulate a read or write to M's ith tape. Since there are a constant number of tapes to be simulated, it thus takes $O(\log T)$ time for N to simulate one move of M. ■

The time hierarchy for deterministic classes with a fixed amount of advice can now be given.

Theorem 3.3 *Let $a(n)$ be any function such that $a(n) < 2^n$ and let $T(n)$ be time-constructible. If $t(n) \in \Omega(n)$ is such that $t(n)\log t(n) \in o(T(n))$, then*

$$\mathsf{DTIME}(t(n))^{(a)} \subsetneq \mathsf{DTIME}(T(n))^{(a)}.$$

Proof. Fix an enumeration of all Turing machines where all machines appear infinitely often, and let M_n denote the nth machine. To prove the theorem, it suffices to define an advice function $g : \mathbb{N} \to \{0,1\}^*$, $|g| = a$, and a deterministic Turing machine $U^{(g)}$ running in time $T(n)$ such that for any advice function $h : \mathbb{N} \to \{0,1\}^*$, $|h| \le a$, no deterministic Turing machine $M^{(h)}$ running in time $t(n)$ accepts $L = L(U^{(g)})$.

The first step is to define g. The string $g(n)$ will be used to describe a function G_n mapping $\{x_2^n, x_3^n, \ldots, x_{a(n)+1}^n\}$ to $\{0,1\}$ where G_n is defined as follows. Let $\mathcal{F}_n$ consist of all functions

$$F_n^k : \{x_2^n, x_3^n, \ldots, x_{a(n)+1}^n\} \to \{0,1\},$$

where k ranges over all bit strings of length $a(n)$ not equal to $0^{a(n)}$, and $F_n^k(x_j^n) = 1$ if and only if $M_n^{(h)}$ accepts x_j^n when $h(n) = k$. Note that there are $2^{a(n)} - 1$ functions in $\mathcal{F}_n$ whereas there are $2^{a(n)}$ functions mapping $\{x_2^n, x_3^n, \ldots, x_{a(n)+1}^n\}$ to $\{0,1\}$. Let G_n be the function mapping $\{x_2^n, x_3^n, \ldots, x_{a(n)+1}^n\}$ to $\{0,1\}$ not in $\mathcal{F}_n$. Note that G_n can be specified by $a(n)$ bits: each bit indicates G_n's behaviour on the $a(n)$ possible inputs. Define $g(n)$ to be the $a(n)$ bits specifying G_n.

The machine $U^{(g)}$ can now be defined. On all inputs of length n, $U^{(g)}$ halts after $T(n)$ steps. On inputs x_i^n, $i \ge 2$, $U^{(g)}$ accepts if and only if the $i-1$th bit on the advice tape is 1; otherwise, $U^{(g)}$ rejects. Note that for inputs x_i^n, $2 \le i \le a(n)+1$, this implies that $U^{(g)}$ accepts if and only if the $i-1$th bit of $g(n)$ is 1 if and only if $G_n(x_{i-1}^n) = 1$.

It remains to describe $U^{(g)}$'s behaviour on $x_1^n = 0^n$. On input 0^n, $U^{(g)}$ simulates M_n on input 0^n and advice $0^{a(n)}$. If the simulation finishes and rejects before $U^{(g)}$ halts, then $U^{(g)}$ accepts; otherwise, $U^{(g)}$ rejects.

The claim is that for all functions $h : \mathbb{N} \to \{0,1\}^*$, $|h| \le a$, no deterministic Turing machine $M^{(h)}$ running in $t(n)$ time accepts $L = L(U^{(h)})$. Indeed, suppose to the contrary that there exist a function $h : \mathbb{N} \to \{0,1\}^*$, $|h| = a$, and a deterministic Turing machine $M^{(h)}$ running in $t(n)$ time such that $M^{(h)}$ accepts L. By Lemma 3.1, $U^{(g)}$ can simulate k steps of $M^{(h)}$'s computation using $ck\log k$ steps, where c depends only on M. Let m be sufficiently large such that $M_m = M$ and $ct(m)\log t(m) \le T(m)$.

A contradiction is now obtained by showing that for any value of $h(m)$, there exists an input x of length m such that $U^{(g)}$ and $M^{(h)} = M_m^{(h)}$ decide differently on x. Suppose that $h(m) = 0^{a(m)}$. Since $ct(m)\log t(m) \le T(m)$, $U^{(g)}$ has time to finish the simulation of $M^{(h)} = M_m^{(h)}$ on input x_1^m and thus, by definition of $U^{(g)}$, $U^{(g)}$ and $M^{(h)}$ differ on x_1^m. If instead $h(m) \ne 0^{a(m)}$, then by definition of G_m and $\mathcal{F}_m$, there exists an input x_j^m, $2 \le j \le a(m)+1$, such that $F_m^{h(m)}(x_j^m) \ne G_m(x_j^m)$. But then, by definition of $F_m^{h(m)}(x_j^m)$, it follows that $U^{(g)}$ and $M^{(h)} = M_m^{(h)}$ differ on x_i^m. ■

Note that by arguing as in the proof of Corollary 3.2, the above theorem can be translated to non-uniform classes defined using Karp-Lipton advice functions. In particular, Theorem 3.3 implies that

$$\mathsf{DTIME}(t(n))/a(n) \subsetneq \mathsf{DTIME}(T(n))/a(n).$$

for all functions a, t and T such that $t(n) \in \Omega(n)$, $t(n)\log t(n) \in o(T(n))$, and a and T are time-constructible.

Obtaining a time-hierarchy for nondeterministic machines taking advice is a little harder. Nevertheless, the proof of the above theorem, in conjunction with ideas from the simple and general proof of the nondeterministic time hierarchy given by Žák [70], can be adapted to obtain a time hierarchy for nondeterministic time with advice. The proof is due to Stephen Cook.

Theorem 3.4 (Cook) *Let $a(n)$ be any function such that $a(n) < 2^n$ and let $T(n)$ be time-constructible. If $t(n) \in \Omega(n)$ is such that $t(n+1) \in o(T(n))$, then*

$$\mathsf{NTIME}(t(n))^{(a)} \subsetneq \mathsf{NTIME}(T(n))^{(a)}.$$

Proof. Fix an enumeration of all Turing machines where all machines appear infinitely often, and let M_n denote the nth machine. Let $S(n)$ be a non-decreasing unbounded integer-valued function such that:

- $S(n)$ is computable in $O(n)$ time.
- For each n, the smallest integer $A(n) \leq n$ such that $S(A(n)) = S(n)$ is computable in $O(n)$ time.
- If $S(n-1) < S(n)$ then $S(n) = S(2^{nT(n)})$.

Such functions $S(n)$ can be found for all "nice" $T(n)$.

To prove the theorem, it suffices to define an advice function $g : \mathbb{N} \to \{0,1\}^*$, $|g| = a$, and a nondeterministic Turing machine $U^{(g)}$ running in time $T(n)$ such that for any advice function $h : \mathbb{N} \to \{0,1\}^*$, $|h| \leq a$, no nondeterministic Turing machine $M^{(h)}$ running in time $t(n)$ accepts $L = L(U^{(g)})$.

The first step is to define g. The string $g(n)$ will be used to describe a function G_n mapping $\{x_2^n, x_3^n, \ldots, x_{a(n)+1}^n\}$ to $\{0,1\}$ where G_n is defined as follows. Let $\mathcal{F}_n$ consist of all functions

$$F_n^k : \{x_2^n, x_3^n, \ldots, x_{a(n)+1}^n\} \to \{0,1\},$$

where k ranges over all bit strings of length $a(n)$ not equal to $0^{a(n)}$, and $F_n^k(x_j^n) = 1$ if and only if $M_{S(n)}^{(h)}$ accepts x_j^n when $h(n) = k$. Note that there are $2^{a(n)} - 1$ functions in $\mathcal{F}_n$ whereas there are $2^{a(n)}$ functions mapping $\{x_2^n, x_3^n, \ldots, x_{a(n)+1}^n\}$ to $\{0,1\}$. Let G_n be the function mapping $\{x_2^n, x_3^n, \ldots, x_{a(n)+1}^n\}$ to $\{0,1\}$ not in $\mathcal{F}_n$. Note that G_n can be specified by $a(n)$ bits: each bit indicates G_n's behaviour on the $a(n)$ possible inputs. Define $g(n)$ to be the $a(n)$ bits specifying G_n.

The machine $U^{(g)}$ can now be defined. On all inputs of length n, $U^{(g)}$ halts after $T(n)$ steps. On inputs x_i^n, $i \geq 2$, $U^{(g)}$ accepts if and only if the $i-1$th bit on the advice tape is 1; otherwise, $U^{(g)}$ rejects. Note that for inputs x_i^n, $2 \leq i \leq a(n)+1$, this implies that $U^{(g)}$ accepts if and only if the $i-1$th bit of $g(n)$ is 1 if and only if $G_n(x_{i-1}^n) = 1$.

It remains to describe $U^{(g)}$'s behaviour on $x_1^n = 0^n$. On input 0^n, $U^{(g)}$ first computes $S(n)$. By definition of $S(n)$, this can be done in $O(n)$ time. Next, $U^{(g)}$ computes the smallest integer m such that $S(m) = S(n)$, i.e., $U^{(g)}$ computes $A(n)$. Again, by definition of $S(n)$, this can be done in $O(n)$ time. Next, $U^{(g)}$ does the following two computations in sequence:

1. $U^{(g)}$ simulates $M_{S(n)}$ *deterministically* on input 0^m and advice $0^{a(m)}$.

2. $U^{(g)}$ simulates $M_{S(n)}$ on input 0^{n+1} and advice $0^{a(n+1)}$.

If the first simulation has finished when $U^{(g)}$ halts, then $U^{(g)}$ decides the opposite of what that simulation decided. If instead the first simulation has not finished when $U^{(g)}$ halts, then $U^{(g)}$ accepts if the second simulation has finished and accepted; otherwise, $U^{(g)}$ rejects.

The claim is that for all functions $h : \mathbb{N} \to \{0,1\}^*$, $|h| \leq a$, no nondeterministic Turing machine $M^{(h)}$ running in time $t(n)$ accepts $L = L(U^{(g)})$. Indeed, suppose to the contrary that there exist a function $h : \mathbb{N} \to \{0,1\}^*$, $|h| = a$, and a nondeterministic Turing machine $M^{(h)}$ running in time $t(n)$ accepting L. First note that in the same way that one shows that a two-tape nondeterministic Turing machine can simulate a multi-tape Turing machine with only a constant factor slowdown [14], it can be shown that a random-access two-tape nondeterministic Turing machine can simulate a random-access multi-tape Turing machine with only a constant factor slowdown. Hence, $U^{(g)}$ can simulate k steps of $M^{(h)}$'s computation using ck steps, where c depends only on M. Moreover, since $t(n+1) \in T(n)$, $U^{(g)}$ can *deterministically* simulate k steps of $M^{(h)}$'s computation using $2^{c'T(k)}$ steps, for some constant c' depending only on M.

Now pick $m \geq c'$ sufficiently large so that $m = A(m)$, $ct(m+1) \leq T(m)$, and $M_{S(m)} = M$. Note then that by definition of m and S, it follows that $S(m') = S(m'+1)$ for all m', $m \leq m' < 2^{mT(m)}$. In particular, $U^{(g)}$ simulates $M = M_{S(m)}$ on input $0^{m'}$, $m \leq m' \leq 2^{mT(m)}$. Note moreover that since M is $t(n)$ time-bounded, for all m', $m \leq m' \leq 2^{mT(m)}$, $U^{(g)}$ on input $0^{m'}$ has enough time to simulate M on $0^{m'+1}$ and advice $0^{a(m'+1)}$.

There are two cases: either $h(j) = 0^{a(j)}$ for all j, $m \leq j < 2^{mT(m)}$, or there exists m', $m \leq m' \leq 2^{mT(m)}$, such that $h(m') \neq 0^{a(m')}$. Assume first that an m', $m \leq m' \leq 2^{mT(m)}$, exists such that $h(m') \neq 0^{a(m')}$. Then by definition of $G_{m'}$ and $\mathcal{F}_{m'}$, there exists an input $x_j^{m'}$, $2 \leq j \leq a(m')+1$, such that $F_{m'}^{h(m')}(x_j^{m'}) \neq G_{m'}(x_j^{m'})$. But then, by definition of $F_{m'}^{h(m')}$, it follows that $U^{(g)}$ differs from $M^{(h)} = M_{S(m')}^{(h)} = M_{S(m')}^{(h)}$ on $x_j^{m'}$. This contradicts the assumption that $U^{(g)}$ and $M^{(h)}$ accept the same language.

So it must be that $h(j) = 0^{a(j)}$ for all j, $m \leq j < 2^{mT(m)}$. Note that by choice of m, on input $0^{2^{mT(m)}}$, $U^{(g)}$ has enough time to finish the deterministic simulation of $M = M_{S(m)}$ on input 0^m and advice $0^{a(m)}$. Let r be the smallest integer between m and $2^{mT(m)}$ such that on input 0^r, $U^{(g)}$ has enough time to finish the simulation of $M_{S(m)}$ on input 0^m and advice $0^{a(m)}$. Note then that by definition of $U^{(g)}$, for all j, $m \leq j < r$, $U^{(g)}$ accepts 0^j if and only $M_{S(j)}^{(h)} = M_{S(m)}^{(h)} = M^{(h)}$ accepts 0^{j+1}:

$$\forall j, m \leq j < r, U^{(g)} \downarrow 0^j \leftrightarrow M^{(h)} \downarrow 0^{j+1}.$$

Moreover, $U^{(g)}$ accepts 0^r if and only if $M_{S(m)}^{(h)} = M^{(h)}$ rejects 0^m:

$$U^{(g)} \downarrow 0^r \leftrightarrow M^{(h)} \uparrow 0^m.$$

But then, if $U^{(g)}$ and $M^{(h)}$ accept the same language, it follows that

$$U^{(g)} \downarrow 0^m \leftrightarrow U^{(g)} \downarrow 0^r \leftrightarrow U^{(g)} \uparrow 0^m,$$

a contradiction. ■

Again, by arguing as in the proof of Corollary 3.2, the above theorem can be translated to non-uniform classes defined using Karp-Lipton advice functions. In particular, Theorem 3.4 implies that

$$\mathsf{NTIME}(t(n))/a(n) \subsetneq \mathsf{NTIME}(T(n))/a(n).$$

for all functions a, t and T such that $t(n) \in \Omega(n)$, $t(n+1) \in o(T(n))$, and a and T are time-constructible.

In fact, the proof of Theorem 3.4 can be used to obtain hierarchies on more general models of computation, such as alternating machines which take advice. For instance, the same proof framework can used to show that if $t(n+1) \in o(T(n))$, then

$$(\Sigma_{s(n)}^{t(n)})^{(a)} \subsetneq (\Sigma_{s(n)}^{T(n)})^{(a)},$$

a result that will be used in Chapter 4.

3.3 Separations of uniform and non-uniform classes

In this section, the emphasis is on obtaining separations between uniform and non-uniform classes. The following theorem, together with its deterministic version Theorem 3.5, is used extensively in later chapters to obtain lower bounds for SAT and PERMANENT on non-uniform classes.

The proof refines the argument used by Rackoff [53] in the proof of Theorem 2.4.

Theorem 3.5 *Let $t(n)$ and $T(n)$ be such that $T(n)$ is time-constructible and $t(n) \in o(T(n))$. Then*

$$\mathsf{NTIME}(T(n)) \not\subseteq \mathsf{coNTIME}(t(n))^{(n)}.$$

Proof. Fix an enumeration of all non-uniform co-nondeterministic Turing machines taking n bits of advice on input of length n such that all machines appear infinitely often. Consider a nondeterministic Turing machine U that computes as follows: on input x, $|x| = n$, U first computes $T(n)$. Then U simulates the complement of the nth machine $M_n^{(?)}$ on input x and advice x for $T(n)$ steps. Computation branches of U that do not have enough time to finish the simulation reject after $T(n)$ steps. It is clear that $L = L(U)$ is in $\mathsf{NTIME}(T(n))$.

The claim is that L is not in $\mathsf{coNTIME}(t(n))^{(n)}$. Indeed, suppose to the contrary that there exists a co-nondeterministic Turing machine $M^{(?)}$ running in time $t(n)$ and a function h mapping $\mathbb{N}$ to $\{0,1\}^*$ such that $|h(n)| = n$ and $L = L(M^{(h)})$. First note that in the same way that one shows that a two-tape nondeterministic Turing machine can simulate a multi-tape Turing machine with only a constant factor slowdown [14], it can be shown that a random-access two-tape nondeterministic Turing machine can simulate a random-access multi-tape Turing machine with only a constant factor slowdown. Hence, U can simulate n steps of $\overline{M^{(?)}}$'s computation in cn steps where c only depends on M. Let m be a sufficiently long encoding of M such that $ct(m) \leq T(m)$. Consider U on input $h(m)$. By definition, U simulates $\overline{M_m^{(?)}} = \overline{M^{(?)}}$ on input $h(m)$ and advice $h(m)$. The simulation of $\overline{M^{(?)}}$ on $h(m)$ and advice $h(m)$ requires time $ct(m) \leq T(m)$, and thus, U has enough time to complete the simulation. But then, $h(m)$ is in L if and only if U accepts $h(m)$ if and only if $\overline{M^{(?)}}$ accepts $h(m)$ when given advice $h(m)$ if and only if $M^{(h)}$ rejects $h(m)$ if and only if $h(m)$ is not in L, a contradiction. ∎

Note that in much the same way one can prove the following analogous theorem for non-uniform classes defined using Karp-Lipton advice functions.

Theorem 3.6 *Let $t(n)$ and $T(n)$ be such that $T(n)$ is time-constructible and $t(3n+2) \in o(T(n))$. Then*

$$\mathsf{NTIME}(T(n)) \not\subseteq \mathsf{coNTIME}(t(n))/n.$$

Note that the "$3m + 2$" term that appears in the statement of the theorem is an artifact of the definition of $\langle , \rangle$ and is therefore a bit artificial. That is, it reflects a property of the pairing function rather than giving any insight about non-uniform classes (see also the comments about Karp-Lipton advice classes in section 2.1).

In the same way that the nondeterministic time hierarchy is much harder to prove as compared with the deterministic time hierarchy, it seems to be much harder to show the existence of a language in, say, $\mathsf{NTIME}(t(n)) - \mathsf{NTIME}(o(t(n-1)))^{(n)}$ for sufficiently "nice" functions t .

Nevertheless, the arguments in the proof of Theorem 3.5 are quite general and can be used to show analogous separations between other complexity classes. For instance, the arguments can be used to show that $\Sigma_{s(n)}^{n \log n}$ is not contained in $(\Sigma_{O(1)}^{n})^{(n)}$ when $s(n)$ is unbounded, a result that is used in the proof of Theorem 4.1 in Chapter 4.

In the deterministic case, the separation obtained between uniform classes and classes defined by non-uniform Turing machines taking n bits of advice on inputs of length n is not as tight. This is due to the fact that the best known simulations, to the author's knowledge, of multi-tape random-access deterministic Turing machines by random-access deterministic Turing machines with a fixed number of tapes are provided by Lemma 3.1. So in the following theorem, a logarithmic factor slowdown is incurred when a deterministic machine is used to diagonalize over all deterministic machines taking n bits of advice on inputs of length n.

Theorem 3.7 *Let $t(n)$ and $T(n)$ be such that $t(n)\log t(n) \in o(T(n))$ and $T(n)$ is time-constructible. Then*

$$\mathsf{DTIME}(T(n)) \not\subseteq \mathsf{DTIME}(t(n))^{(n)}.$$

Like Theorem 3.5, Theorem 3.7 has formulations using Karp-Lipton advice functions rather than non-uniform Turing machines. As with Theorem 3.5, note the similarity between Theorem 3.7 and Theorem 2.4.

Note that Theorem 3.7 implies that if t is superpolynomial and time constructible, there then exist languages in $\mathsf{DTIME}(t(n)) - \mathsf{P}$ that are not in $\mathsf{P}^{(log)}$ or even in $\mathsf{P}^{(n)}$. A reasonable question to then ask would be whether the intersection of $\mathsf{P}^{(log)}$ with any time complexity class larger than P is simply P. The answer is negative. Recall that a tally language is a language over a single letter alphabet. Note first that all tally languages are in $\mathsf{DTIME}(n)^{(1)}$. In particular, all tally languages are in $\mathsf{P}^{(log)}$. Now note that the standard time hierarchy theorems can be adapted to show that for each time constructible function $t(n)$, there exists a tally language L that cannot be computed by a deterministic machine in $O(\sqrt{t})$ time. It follows that $\mathsf{DTIME}(n)^{(1)} \cap (\mathsf{DTIME}(t) - \mathsf{P})$ is non-empty for all time-constructible t. So even one bit of advice can be used to speed up some computations.

So which languages are decided faster with the aid of n bits of advice on inputs of length n? To that end, it would be nice if one could show, say, that languages complete for time complexity classes larger than P are not decided faster with the aid of n bits of advice on inputs of length n. Theorem 3.7 can be used to obtain a slightly weaker result along these lines. Recall that there is a polynomial-time many-one reduction or *Karp reduction* from a language A to a language B if there exists a polynomial-time computable function f such that x is in A if and only if $f(x)$ is in B. This reduction is called *natural* if in addition the length of $|f(x)|$ only depends on the length of x. Note that all known natural NP-complete problems such as SAT are complete under natural Karp reductions. The reader is referred to [41], where to the author's knowledge the idea of a natural Karp reduction first appears, for more information. The theorem below states that all problems complete under natural Karp reductions for a time complexity class $\mathcal{C}$ properly containing P are not in $\mathsf{P}^{(n)}$. So, for instance, problems complete for $\mathsf{DTIME}(2^{O(\log^2 n)})$ under natural Karp reductions are not in $\mathsf{P}^{(log)}$ or P/log.

Theorem 3.8 *Let $\mathcal{C}$ be any deterministic time-complexity class properly containing P. Then no problems in $\mathcal{C}$ complete under natural Karp reductions are in $\mathsf{P}^{(n^{o(1)})}$.*

Proof. If $\mathcal{C}$ has no problems complete under natural Karp reductions, then there is nothing to prove. So assume that C is complete for $\mathcal{C}$ under natural Karp reductions. Now assume, in order to obtain a contradiction, that C is in $\mathsf{P}^{(n^{o(1)})}$. In particular, assume there exists a function $h : \mathbb{N} \to \Sigma^*$, $|h| \in n^{o(1)}$, and a polynomial-time non-uniform Turing machine $M^{(?)}$ such that $C = L(M^{(h)})$.

Let L be the language in $\mathcal{C}$ not in $\mathsf{P}^{(n)}$ guaranteed by Theorem 3.7. Then there exists a polynomial-time computable function f such that x is in L if and only if $f(x)$ is in C, and moreover, $|f|$ only depends on $|x|$. Let $g(n) = |f(x)|$ where x is any string of length n. Note that since f is a natural Karp reduction, g is well-defined and is bounded by a polynomial.

Now consider a non-uniform Turing machine $N^{(h\circ g)}$ that on input x of length n uses its advice $h(g(n))$ to compute like M on input $f(x)$. Note that $N^{(h\circ g)}$ runs in polynomial time and accepts L. Moreover, since $|h| \in n^{o(1)}$, it follows that $|h \circ g| \in n^{o(1)}$. But then, L is in $\mathsf{P}^{(n^{o(1)})}$, a contradiction. ∎

The techniques from the proofs of Theorems 3.1 and 3.3 can be used to obtain the following, more general separation of uniform and non-uniform classes. Note however that whereas the separations are obtained for classes taking arbitrary amounts of advice, the separations are not as tight as those obtained in Theorem 3.5.

Theorem 3.9 *Let $T(n)$ and $k(n) \geq n$ be time-constructible and let $t(n)$ be such that $t(n) \log t(n) \in o(T(n))$. Suppose in addition that $a(n) = k(n) + \lfloor \log\lfloor 2^n/k(n)\rfloor\rfloor \leq 2^n$. Then there exists a language L in $\mathsf{DTIME}(k(n)2^{k(n)}T(n))$ such that for any advice function $h : \mathbb{N} \to \{0,1\}^*$ where $|h| \leq a$, no deterministic Turing machine $M^{(h)}$ running in time $t(n)$ can decide L.*

Proof. Encode all Turing machines taking $a(n)$ advice such that each machine appears infinitely often in the encoding. The idea is to construct a $k(n)2^{k(n)}T(n)$ time-bounded deterministic Turing machine U such that for any advice function $h : \mathbb{N} \to \{0,1\}^*$, $|h| \leq a$, and each machine M taking $a(n)$ advice, there exists a sufficiently large n such that $M = M_n$ and U and $M^{(h)}$ differ on some input of length n.

To that end, for all n and all m, $0 \leq m < \lfloor 2^n/k(n)\rfloor$, define the following sets:

$$\begin{aligned} \mathcal{A}^n_m &= \{x^{a(n)}_{m2^{k(n)}+1}, x^{a(n)}_{m2^{k(n)}+2}, \ldots, x^{a(n)}_{(m+1)2^{k(n)}}\}, \\ \mathcal{I}^n_m &= \{x^n_{mk(n)+1}, x^n_{mk(n)+2}, \ldots, x^n_{(m+1)k(n)}\}. \end{aligned}$$

The machine U will be designed so that if $M = M_n$ and n is sufficiently large, then for any value of $h(n)$, if $h(n) \in \mathcal{A}^n_m$, where $0 \leq m < \lfloor 2^n/k(n)\rfloor$, then there exists an input $x \in \mathcal{I}^n_m$ such that U and

$M_n^{(h)} = M^{(h)}$ differ on x. Note that the union of the $\mathcal{A}_m^n$ over $0 \le m < \lfloor 2^n/k(n) \rfloor$ consists of all bit strings of length $a(n)$ and that the union of the $\mathcal{I}_m^n$ over $0 \le m < \lfloor 2^n/k(n) \rfloor$ is contained in the set of all bit strings of length n (i.e., in the set of all inputs of length n).

Machine U computes as follows. On all inputs of length n, U halts after $k(n)2^{k(n)}T(n)$ steps. On input x of length n, U first computes the integer m for which $x \in \mathcal{I}_m^n$. This can be done in linear time. Then for each value of $h(n)$ in $\mathcal{A}_m^n$, U simulates $M_n^{(h)}$ on each input in $\mathcal{I}_m^n$. If U halts before finishing the simulations, then U rejects. Otherwise, the results of these simulations are tabulated. Using this table, U construct a function G_m^n mapping $\mathcal{I}_m^n$ to $\{0,1\}$ defined as follows. Let $\mathcal{F}_m^n$ consist of all functions

$$F_k^{n,m} : \mathcal{I}_m^n \to \{0,1\},$$

where k ranges over all bit strings in $\mathcal{A}_m^n$, and for all $y \in \mathcal{I}_m^n$, $F_k^{n,m}(y) = 1$ if and only if $M_n^{(h)}$ accepts y when $h(n) = k$. Note that there are $2^{k(n)}$ functions in $\mathcal{F}_n$ whereas there are $2^{k(n)+1}$ functions mapping $\mathcal{I}_m^n$ to $\{0,1\}$. Note also that a function F mapping $\mathcal{I}_m^n$ to $\{0,1\}$ can be specified by a bit string of length $k(n)$: bit i is 1 if and only if F is 1 on the ith element of $\mathcal{I}_m^n$. Then G_m^n is defined to be the function mapping $\mathcal{I}_m^n$ to $\{0,1\}$ not in $\mathcal{F}_n$ with the lexicographically smallest bit string description. After U has constructed G_m^n, U accepts x_j^n if and only if $G_m^n(x_j^n) = 1$.

The claim is that for all advice functions $h : \mathbb{N} \to \{0,1\}^*$, $|h| \le a$, no deterministic Turing machine $M^{(h)}$ running in time $t(n)$ accepts $L = L(U)$. Indeed, fix an advice function $h : \mathbb{N} \to \{0,1\}^*$, $|h| = a$, and suppose to the contrary that a deterministic Turing machine $M^{(h)}$ running in time $t(n)$ accepts L. Recall that at the beginning of the previous section it was noted that a random-access Turing machine with at least two tapes can simulate a multi-tape random-access Turing machine with only a logarithmic factor slowdown. Hence, U requires $ck(n)2^{k(n)}t(n)\log t(n)$ steps to compute G_m^n, where c depends only on M. Choose j sufficiently large so that $M = M_j$ and $ck(j)2^{k(j)}t(j)\log t(j) \le ck(j)2^{k(j)}T(j)$.

Now suppose that $h(j) = x^{a(j)}_{m2^{k(j)}+\ell}$ for some integers m and ℓ. Then by definition of G_m^j, there exists an input $x \in \mathcal{I}_m^j$ such that G_m^j and $F_{h(j)}^{j,m}$ differ on x. By definition of $F_{h(j)}^{j,m}$, it follows that $M^{(h)} = M_j^{(h)}$ accepts x if and only if $G_m^j(x) = 0$. But, on input x, U has enough time to calculate $G_m^j(x)$ and thus, U will accept x if and only if $G_m^j(x) = 1$. Hence, U and $M^{(h)}$ differ on x, contradicting the assumption that $M^{(h)}$ and U decided the same language. ■

Theorem 3.9 implies the upper bound given below on the power of polynomial size circuits.

Corollary 3.10 $\mathsf{DTIME}(2^{n^{\omega(1)}}) \not\subseteq \mathsf{P}/poly$. *In particular,* $\mathsf{DTIME}(2^{n^{\log^* n}}) \not\subseteq \mathsf{P}/poly$.

Proof. Let $T(n)$ and $k(n)$ be in $n^{\omega(1)}$ and apply Theorem 3.9. ■

Kannan [42] used essentially the same arguments to obtain the stronger result that $\mathsf{DSPACE}(t(n))$ is not contained in $\mathsf{P}/poly$ whenever t is superpolynomial.

3.4 Discussion

Note that the arguments used in all the theorems in this chapter can, of course, be used to prove analogous results for space-bounded machines and classes.

It is likely that the diagonalization results proved in this chapter can be substantially improved, although new methods seem to be needed. For instance, it is expected that NP contains sets that do not have polynomial circuits, a result much stronger than the upper bound for $\mathsf{P}/poly$ provided by Corollary 3.10.

It would be interesting to determine in what ways Theorems 3.3 and 3.4 can be strengthened. For instance, can the amount of advice required by the diagonalizing machines in these two theorems be substantially reduced? Another question that would be of interest to settle would be whether $\mathsf{DTIME}(t(n))^{(O(a(n)))}$ is properly contained in $\mathsf{DTIME}(T(n))^{(O(a(n)))}$ for appropriate functions $a(n)$ and $t(n) \in o(T(n))$. Settling this question in the affirmative would imply a substantial strengthening of the amount of non-uniformity in the results in Chapters 4 and 5. Of even more interest would be to determine whether $\mathsf{DTIME}(t(n))^{(poly)}$ is properly contained in $\mathsf{DTIME}(T(n))^{(poly)}$ whenever T is time-constructible and $t(n) \in o(T(n))$. As was mentioned in section 2.1, affirmative answers to this conjecture could be used together with the techniques used to prove Theorem 4.1 to show that SAT does not have non-uniform NC^1 circuits of quasilinear size.

Extending the amount of non-uniformity in Theorems 3.5 and 3.7 seems quite difficult. Indeed, obtaining a separation of the form $\mathsf{DTIME}(T(n)) \not\subseteq \mathsf{DTIME}(t(n))^{(2n)}$ where T does not grow substantially faster than $t(n)$ seems quite hard. Moreover, extending the amount of non-uniformity in Theorem 3.5 to $\omega(n)$ would imply the existence of a language in NP having no linear circuits, a result that would be a major breakthrough for complexity theory. This further demonstrates how difficult increasing the amount of non-uniformity in Theorem 3.7 is likely to be.

Likewise, substantially decreasing the amount of time needed by the diagonalizing machine in the proof of Theorem 3.9 could lead to a proof that $\mathsf{E} = \mathsf{DTIME}(2^{O(n)})$ is not contained in $\mathsf{P}/poly$. However, Impagliazzo and Wigderson [40] have shown that this implies that $\mathsf{P} = \mathsf{BPP}$, where BPP is the class of languages accepted by randomized polynomial time Turing machines making bounded error. Thus, substantially improving Theorem 3.9 seems quite difficult since it would settle one of the major open questions in computational complexity theory.

An extension that is likely to be more tractable is to prove that for functions t and T such that T is time-constructible and $t(n) \in o(T(n))$,

$$\mathsf{NTIME}(T(n)) \not\subseteq \mathsf{NTIME}(t(n))^{(n)}.$$

It seems plausible that the proof of Theorem 3.5, perhaps with ideas borrowed from Žák's proof of the nondeterministic hierarchy [70], can be adapted to prove such a result.

Chapter 4

Lower bounds for SAT. Part I

In the next two chapters the emphasis is on using the results from Chapters 2 and 3 to extend recent time-space lower bounds for SAT to non-uniform Turing machines. Note that it will be convenient to state and prove the results in these two chapters for non-uniform classes defined using Karp-Lipton advice functions rather than by employing the non-uniform Turing machine model introduced in Chapter 2. Nevertheless, using Theorem 2.1 equivalent formulations of all the results in this and the following chapter can be obtained for non-uniform classes defined using non-uniform Turing machines. The choice to use Karp-Lipton advice functions is made since the precision gained by using non-uniform Turing machines is not needed for the results in this and the following chapter. Moreover, using Karp-Lipton advice functions results in a cleaner exposition of the proofs in these two chapters.

In this chapter Fortnow's [26] time-space lower bounds for SAT are extended to non-uniform classes. In [26] Fortnow showed that for all $\epsilon > 0$, SAT cannot be computed by a random-access multi-tape deterministic Turing machine running in $n^{1+o(1)}$ time and using $n^{1-\epsilon}$ space. More generally, Fortnow showed that for all $\epsilon > 0$ and any unbounded function $r(n)$ such that $r(n) \in O(\frac{\log n}{\log \log n})$,

$$\overline{\mathrm{SAT}} \notin \mathsf{NTIME}(n^{1+\frac{1}{r(n)}}) \cap \mathsf{NTISP}(n^{o(r(n))}, n^{1-\epsilon}).$$

Fortnow's paper also presented a proof (credited to Harry Buhrman) showing that SAT cannot be computed by log-space uniform NC^1 circuits of size $n^{1+o(1)}$.

The basic tools in Fortnow's proof are an argument giving sufficient conditions for collapsing a non-constant number of levels of the polynomial hierarchy, a version of Nepomnjaščiĭ's theorem [48] as generalized by Kannan [43], and a simple diagonalization. As will be shown, these arguments readily adapt to the non-uniform arena where the amount of non-uniformity is subpolynomial.

In sections 4.1 and 4.2 non-uniform extensions of Buhrman's circuit lower bound for SAT are considered. In section 4.1 it is shown that SAT cannot both have circuits of size $n^{1+o(1)}$ and be in $\mathsf{SC}/n^{o(1)}$ or $\mathsf{NL}/n^{o(1)}$. In particular, SAT cannot have quasilinear NC^1 circuits generated by a nondeterministic logarithmic-space transducer taking subpolynomial advice. Nondeterministic transducers are defined below. In section 4.2 similar techniques are used to show that SAT does not have quasilinear NC^1 circuits of subpolynomial descriptive size.

In section 4.3 non-uniform extensions of Fortnow's time-space tradeoffs for $\overline{\mathrm{SAT}}$ are discussed. In particular, a proof is given showing that for all $\epsilon > 0$, SAT cannot be computed by a deterministic machine running in time $n^{1+o(1)}$, space $n^{1-\epsilon}$, and taking subpolynomial advice. This result follows from a more general result showing that

$$\overline{\text{SAT}} \notin \mathsf{NTIME}(n^{1+\frac{1}{r(n)}})/n^{o(1)} \cap \mathsf{NTISP}(n^{o(r(n))}, n^{1-\epsilon})/n^{o(1)}.$$

for all $\epsilon > 0$ and all unbounded functions $r(n)$ such that $r(n) \in O(\frac{\log n}{\log \log n})$.

4.1 A lower bound for SAT on $\mathsf{NC}^1[\mathsf{NL}/n^{o(1)}]$ circuits

The main theorem proved in this subsection is the following extension to Buhrman's circuit lower bound for SAT:

Theorem 4.1 SAT *cannot both have circuits of size* $n^{1+o(1)}$ *and be in* $\mathsf{NL}/n^{o(1)}$.

Recall that $\mathsf{NC}^1[\mathsf{NL}/n^{o(1)}]$ consists of all languages A for which there exist a family $\{C_n\}$ of NC^1 circuits accepting A, a nondeterministic logarithmic-space transducer M, and a function $h : \mathbb{N} \to \Sigma^*$ such that $|h| \in n^{o(1)}$ and M on input $\langle 1^n, h(n)\rangle$ outputs C_n. Note that a *nondeterministic transducer* M is a transducer that is allowed to make nondeterministic moves and such that it only produces output on an accepting computation path. In addition, a nondeterministic transducer always has at least one accepting computation path on each input, and moreover, the output on each accepting path must be identical. Note also that $\mathsf{NC}^1[\mathsf{NL}/n^{o(1)}]$ properly contains log-space uniform NC^1.

Theorem 4.1 implies the following lower bound for SAT on $\mathsf{NC}^1[\mathsf{NL}/n^{o(1)}]$ circuits:

Corollary 4.2 SAT *cannot be computed by* $\mathsf{NC}^1[\mathsf{NL}/n^{o(1)}]$ *circuits of size* $n^{1+o(1)}$.

Proof. If SAT has $\mathsf{NC}^1[\mathsf{NL}/n^{o(1)}]$ circuits of size $n^{1+o(1)}$ then SAT is in $\mathsf{NL}/n^{o(1)}$ contradicting Theorem 4.1. ■

The proof of Theorem 4.1 is similar to the proof given in [26] for Buhrman's original result. As with Buhrman's result, the argument relies on two lemmas. The first, Lemma 4.1, appeared in Fortnow [26] where it was credited to Buhrman. The second, Lemma 4.2, is an easy non-uniform generalization of a version of Nepomnjaščiĭ's Theorem [48] as it appeared in Kannan [43] and later in Fortnow [26].

Lemma 4.1 (Buhrman) *If* SAT *has circuits of size* $n^{1+o(1)}$ *then there exists an unbounded function* $s(n) \in O(\frac{\log n}{\log \log n})$ *such that* $\text{QBF}_{s(n)}$ *is in* Σ_2^p.

Proof. The proof is essentially a word-for-word translation of the proof of Lemma 4.1 in [26] and is included for completeness.

Assume SAT has circuits of size $n^{1+o(1)}$. In [45] Karp and Lipton showed that if SAT has polynomial-size circuits, then $\Sigma_2^p = \Pi_2^p$. As Fortnow notes in [26], Karp and Lipton's proof can be adapted to show that if SAT has circuits of size $n^{1+o(1)}$, then $\text{QBF}_2 \in \Pi_2^{n^{1+o(1)}}$. Hence, $\text{QBF}_2 \in \Pi_2^{n^{1+o(1)}}$. This fact can then be used to convert a formula with $k-1$ quantifier alternations to an equivalent but slightly longer formula with $k-2$ alternations. Recursively applying this reduction to a $\text{QBF}_{s(n)}$ formula for an appropriately slow growing function $s(n)$ then gives an equivalent QBF_2 formula of polynomial size with respect to the original formula, thereby proving the lemma.

For simplicity, assume that $\mathrm{QBF}_2 \in \Pi_2^{n \log n}$. The general case $\mathrm{QBF}_2 \in \Pi_2^{n^{1+o(1)}}$ is similar. Let ψ_k, $|\psi_k| = m_k$, be a formula with $k-1$ alternations beginning with $\exists$. Only the case where k is even is considered since the case where k is odd is similar. So ψ_k is of the form

$$\psi_k = \exists \bar{x}_1 \forall \bar{x}_2 \ldots \forall \bar{x}_{k-2} \exists \bar{x}_{k-1} \forall \bar{x}_k \phi(\bar{x}_1, \ldots, \bar{x}_k).$$

Consider the formula

$$\tau(\bar{x}_1, \ldots, \bar{x}_{k-2}) = \exists \bar{x}_{k-1} \forall \bar{x}_k \phi(\bar{x}_1, \ldots, \bar{x}_k).$$

Since $\mathrm{QBF}_2 \in \Pi_2^{n \log n}$ and $|\tau| \leq |\psi_k| = m_k$, the truth of τ can be computed by a Π_2 machine running in time $m_k \log m_k$. Lemma 1.3 then implies the existence of a formula

$$\forall \bar{y}_1 \exists \bar{y}_2 \sigma(\bar{x}_1, \ldots, \bar{x}_{k-2}, \bar{y}_1 . \bar{y}_2),$$

of length bounded by $m_k \log^4 m_k$ equivalent to τ. Hence, ψ_k is equivalent to the formula

$$\exists \bar{x}_1 \forall \bar{x}_2 \ldots \forall \bar{x}_{k-2} \forall \bar{y}_1 \exists \bar{y}_2 \sigma(\bar{x}_1, \ldots, \bar{x}_{k-2}, \bar{y}_1, \bar{y}_2),$$

a formula of length $m_k \log^4 m_k$ with $k-2$ alternations.

The above reduction is now applied repeatedly until a QBF_2 formula equivalent to the original $\mathrm{QBF}_{s(n)}$ formula is obtained. If a formula with $k-1$ quantifier alternations has length m_k, it is not hard to see that the equivalent QBF_2 formula produced using the above reduction has length bounded by $m_k \log^{O(k)} m_k$. So choosing $s(n) \in O(\frac{\log n}{\log \log n})$ ensures that given a $\mathrm{QBF}_{s(n)}$ formula ϕ, the equivalent QBF_2 formula obtained by the above reduction has polynomial length with respect to ϕ. ■

The following lemma showing that non-uniform nondeterministic computations can be simulated in linear time by non-uniform alternating machines is also needed for the proof of Theorem 4.1.

Lemma 4.2 *Let $\epsilon > 0$. Then for any function $\alpha(n)$, $1 \leq \alpha(n) \leq n^{o(1)}$, and any class $\mathcal{F}$ of functions mapping $\mathbb{N}$ to $\mathbb{N}$,*

$$\mathsf{NTISP}(n^{O(\alpha(n))}, n^{1-\epsilon})/\mathcal{F} \subseteq \Sigma^n_{O(\alpha(n))}/\mathcal{F}.$$

Lemma 4.2 follows from Nepomnjaščiĭ's Theorem in the form in which it appeared in [43] and [26]:

Lemma 4.3 (Nepomnjaščiĭ's Theorem) *Let $\epsilon > 0$. Then for any function $\alpha(n)$, $1 \leq \alpha(n) \leq n^{o(1)}$,*

$$\mathsf{NTISP}(n^{O(\alpha(n))}, n^{1-\epsilon}) \subseteq \Sigma^n_{O(\alpha(n))}.$$

Proof. The proof is essentially a word-for-word translation of Fortnow's proof whose proof is in turn based on Kannan's proof.

Fix ϵ and c and let A be in $\mathsf{NTISP}(n^{c\alpha(n)}, n^{1-\epsilon})$. Then there exists a nondeterministic Turing machine M accepting A which on input x of length n takes $n^{c\alpha(n)}$ time and uses $n^{1-\epsilon}$ space. Without loss of generality, assume that before accepting, M erases its work tapes and moves all its heads to the left-most position. Let A_f^x denote this final configuration on input x.

Now fix an input x, $|x| = n$. Note then that if A_0^x is the initial configuration of M on x, then M accepts x if and only if there exists a valid computation of M from A_0^x to A_f^x of length $n^{c\alpha(n)}$. So it suffices to construct a $\Sigma^n_{O(\alpha(n))}$ algorithm that checks whether there exists a computation path of M from A^x_0 to A^x_f of length $n^{c\alpha(n)}$.

Before giving the algorithm, note that since M uses $n^{1-\epsilon}$ space, each configuration of M on x has length $O(n^{1-\epsilon})$. Hence, if A and B are two configurations of M on input x, checking whether $A \vdash_{M,x} B$ can be done in deterministic time $O(n^{1-\epsilon})$.

The procedure CHECK(A, B, t) given below returns true if there exists a computation path of M on input x from A to B of length t. Let $q(n) = n^\beta$ and define CHECK(A, B, t) as follows:

1. If $t \leq q(n)(q(n) + 1)$, existentially guess a valid computation path of M on input x from A to B of length t. Accept if such a path is found; reject otherwise.

2. Else (i.e., if $t > q(n)(q(n)+1)$), existentially guess $q(n)$ configurations $A_1, \dots, A_{q(n)}$ of M on input x. Let $A_0 = A$.

3. Universally guess an $i < q(n)$.

4. Let $m = \lceil t/(q(n) + 1) \rceil$. If $0 \leq i < q(n)$, then output CHECK(A_i, A_{i+1}, m); otherwise, output CHECK$(A_{q(n)}, B, t - mq(n))$.

Let M' be an alternating machine that computes CHECK$(A^x_0, A^x_f, n^{c\alpha(n)})$ and thus, simulates M. To prove the lemma, it suffices to show that M' requires $O(\alpha(n))$ alternations and runs in linear time. It can be verified that computing CHECK$(A^x_0, A^x_f, n^{c\alpha(n)})$ requires $O(\frac{c}{\beta}\alpha(n))$ alternations (see Fortnow [26]). Moreover, the time taken for each call of CHECK, not including time spent in recursive calls, is in $O(n^{2\beta+1-\epsilon})$. Choosing $\beta < \epsilon/2$ then ensures that M' runs in linear time. ∎

The proof of Lemma 4.3 now follows easily:

Proof of Lemma 4.2. Suppose that $A \in \mathsf{NTISP}(n^{c\alpha(n)}, n^{1-\epsilon})/t(n)$ for some constant c and some $t(n) \in \mathcal{F}$. Then there exists a nondeterministic Turing machine M running in time $n^{c\alpha(n)}$ and space $n^{1-\epsilon}$ and a function $h : \mathbb{N} \to \Sigma^*$ such that $|h(n)| = t(n)$ and x is in A if and only if M accepts $\langle x, h(|x|)\rangle$. Lemma 4.3 then implies the existence of a $\Sigma^n_{O(\alpha(n))}$ machine accepting $L(M)$. The lemma follows. ∎

The proof of Theorem 4.1 is by contradiction. The first hypothesis, in conjunction with Lemma 4.1, is used to show that for some unbounded function $s(n) \in O(\frac{\log n}{\log\log n})$, $\Sigma^{n\log n}_{s(n)}$ is contained in Σ^p_2. The second hypothesis is then used to show that Σ^p_2 is contained in $\mathsf{NL}/n^{o(1)}$. Lemma 4.2 then implies that $\Sigma^{n\log n}_{s(n)}$ is contained in $\Sigma^n_{O(1)}/n^{o(1)}$. Diagonalizing as in the proof of Theorem 3.5 then gives the desired contradiction.

Proof of Theorem 4.1. Assume SAT has circuits of size $n^{1+o(1)}$. Lemma 4.1 implies the existence of an unbounded function $s(n) \in O(\frac{\log n}{\log\log n})$ such that,

$$\Sigma^{n\log n}_{s(n)} \subseteq \Sigma^p_2. \tag{4.1}$$

Now assume in addition that SAT $\in \mathsf{NL}/n^{o(1)}$ to derive a contradiction. The first step is to show that this assumption implies that $\Sigma^p_2 \subseteq \mathsf{NL}/n^{o(1)}$. To that end, note that any language in $\mathsf{NP}/n^{o(1)}$ can be reduced to SAT via a log-space transducer taking subpolynomial advice. So if SAT $\in \mathsf{NL}/n^{o(1)}$, it follows that $\mathsf{NP}/n^{o(1)} = \mathsf{NL}/n^{o(1)}$. In particular, $\mathsf{NP}/n^{o(1)} = \mathsf{coNP}/n^{o(1)}$.

Now fix $A \in \Sigma_2^p$. There exist a polynomial f and a relation $R \in \mathsf{coNP}$ such that $x \in A$ if and only if there exists y, $|y| \le f(|x|)$, such that $(x,y) \in R$ (assume that $|(x,y)| = |x| + |y|$). Moreover, by padding y (y can be thought of as the list of non-deterministic choices made in the "existential stage" of the Σ_2^p machine accepting A, and so, it can be padded with unused extra choices), assume without loss of generality, that $|y| = f(|x|)$. Since $\mathsf{coNP}/n^{o(1)} = \mathsf{NP}/n^{o(1)}$, it follows that $R \in \mathsf{NP}/n^{o(1)}$. So there exist a polynomial time nondeterministic machine M and a function $h : \mathbb{N} \to \Sigma^*$ such that $|h|$ is subpolynomial and $(x,y) \in R$ if and only if M accepts $\langle (x,y), h(|(x,y)|)\rangle$. Consider a nondeterministic machine M' that on input $\langle x, w\rangle$, checks that $|w| = |h(|x| + f(|x|))|$, guesses a string y of length $f(|x|)$, and then simulates M on $\langle (x,y), w\rangle$. Note that M' runs in polynomial time and that $x \in A$ if and only if M' accepts $\langle x, h(|x| + f(|x|))\rangle$. Moreover, since $|h|$ is subpolynomial, $|h(n + f(n))|$ is also subpolynomial and thus, $A \in \mathsf{NP}/n^{o(1)} = \mathsf{NL}/n^{o(1)}$. So $\Sigma_2^p \subseteq \mathsf{NL}/n^{o(1)}$.

Equation (4.1) and Lemma 4.2 then imply that

$$\Sigma_{s(n)}^{n \log n} \subseteq \Sigma_2^p \subseteq \mathsf{NL}/n^{o(1)} \subseteq \Sigma_{O(1)}^{n}/n^{o(1)}.$$

A contradiction is now obtained by finding a language in $\Sigma_{s(n)}^{n \log n} - \Sigma_{O(1)}^{n}/n$ using the arguments in the proof of Theorem 3.5. ■

Recall that SC consists of the class of all languages accepted by deterministic Turing machines running simultaneously in polynomial time and polylogarithmic space. Alternatively, SC can be thought of as the class of all languages accepted by circuit families of polynomial size and polylogarithmic width. The proof of Theorem 4.1 can be adapted to to give the following result:

Theorem 4.3 SAT *cannot both have circuits of size* $n^{1+o(1)}$ *and be in* $\mathsf{SC}/n^{o(1)}$.

In particular, SAT cannot be computed by $\mathsf{SC}/n^{o(1)}$-uniform quasilinear-size polylogarithmic-width circuits.

The proof of Theorem 4.1 cannot be used to extend to the nondeterministic version of $\mathsf{SC}/n^{o(1)}$. The problem is that, to the author's knowledge, no class known to be closed under complementation is contained between $\mathsf{NTISP}(n^{O(1)}, \log^{O(1)} n)/n^{o(1)}$ and $\mathsf{NP}/n^{o(1)}$. Thus, assuming that SAT is in $\mathsf{NTISP}(n^{O(1)}, \log^{O(1)} n)/n^{o(1)}$ does not imply that $\mathsf{NP}/n^{o(1)}$ is closed under complement. This is a problem since showing that $\mathsf{NP}/n^{o(1)} = \mathsf{coNP}/n^{o(1)}$ is a key step in the proof of Theorem 4.1 above. Nevertheless, van Melkebeek [63] has shown that Theorems 4.1 and 4.3 can be strengthened by using some of the arguments presented in section 4.3 below. These extensions are briefly discussed in the discussion section at the end of this chapter.

4.2 A lower bound for SAT on $\mathsf{NC}^1_{n^{o(1)}}$ circuits

In this section lower bounds on the size of log-depth circuits of "small" descriptive size computing SAT are given. Recall that $\mathsf{NC}^1_{n^{o(1)}}$ circuits are NC^1 circuits that have descriptors of subpolynomial size. The following extension of Buhrman's result for circuits with subpolynomial descriptors is proved:

Theorem 4.4 SAT *cannot both have circuits of size* $n^{1+o(1)}$ *and have* $\mathsf{NC}^1_{n^{o(1)}}$ *circuits.*

The theorem implies, in particular, the following lower bound for SAT:

Corollary 4.5 SAT *does not have* $NC^1_{n^{o(1)}}$ *circuits of size* $n^{1+o(1)}$.

Note the similarity to Corollary 4.2. Here the $\mathsf{NC}^1[\mathsf{NL}/n^{o(1)}]$ circuits from Corollary 4.2 are replaced by NC^1 circuits with subpolynomial descriptive size.

Proof of Theorem 4.4. The proof is similar to the proof of Theorem 4.1.

As in the proof of Theorem 4.1, begin by assuming that SAT has circuits of size $n^{1+o(1)}$ in order to show, using Lemma 4.1, that

$$\Sigma^{n \log n}_{s(n)} \subseteq \Sigma^p_2. \tag{4.2}$$

for some unbounded $s(n) \in O(\frac{\log n}{\log \log n})$

Now assume in addition that SAT is in $\mathsf{NC}^1_{n^{o(1)}}$ in order to derive a contradiction. The first step is to show that this assumption implies that $\Sigma^p_2 \subseteq \mathsf{NC}^1_{n^{o(1)}}$. To that end, recall that any language in NP can be reduced to SAT via a log-space uniform NC^1 reduction. Since log-space uniform NC^1 is contained in $\mathsf{NC}^1_{n^{o(1)}}$, this implies that any language in $\mathsf{NP}/n^{o(1)}$ can be reduced to SAT via an $\mathsf{NC}^1_{n^{o(1)}}$ reduction. So since SAT is in $\mathsf{NC}^1_{n^{o(1)}}$, it follows that $\mathsf{NP}/n^{o(1)} = \mathsf{NC}^1_{n^{o(1)}}$, and hence, that $\mathsf{NP}/n^{o(1)}$ is closed under complement. Arguing now as in the proof of Theorem 4.1 then gives that $\Sigma^p_2 \subseteq \mathsf{NC}^1_{n^{o(1)}}$. Equation (4.2) then implies that

$$\Sigma^{n \log n}_{s(n)} \subseteq \Sigma^p_2 \subseteq \mathsf{NC}^1_{n^{o(1)}} \tag{4.3}$$

Now note that any language L recognized by an $\mathsf{NC}^1_{n^{o(1)}}$ circuit family $\{C_n\}$ can be recognized by a deterministic Turing machine M_L running in polynomial time and subpolynomial space, using subpolynomial advice: on input of size n, M_L has as advice the descriptor circuit for C_n. On input x, $|x| = n$, M_L evaluates C_n on x, constructing each gate of C_n as needed with the aid of the descriptor circuit for C_n. Since the family $\{C_n\}$ has logarithmic depth, only logarithmic space is required to evaluate C_n. Moreover, since the descriptors have subpolynomial size, only subpolynomial time and (reusable) subpolynomial space is needed for each evaluation of the descriptor circuit. So, since every subpolynomial function is bounded by $\sqrt{n}$, it follows that $\mathsf{NC}^1_{n^{o(1)}} \subseteq \mathsf{DTISP}(n^{O(1)}, \sqrt{n})/n^{o(1)}$. By Lemma 4.2 and Equation (4.3) it then follows that

$$\Sigma^{n \log n}_{s(n)} \subseteq \Sigma^p_2 \subseteq \mathsf{NC}^1_{n^{o(1)}} \subseteq \mathsf{DTISP}(n^{O(1)}, \sqrt{n})/n^{o(1)} \subseteq \Sigma^n_{O(1)}/n^{o(1)}.$$

As in the proof of Theorem 4.1, a contradiction is now obtained by finding a language in $\Sigma^{n \log n}_{s(n)} - \Sigma^n_{O(1)}/n$ by using the arguments in the proof of Theorem 3.5. ∎

An obvious question to ask is which of Corollary 4.2 and Corollary 4.5 presents the tightest lower bound? Of course, this question can only be answered by determining the relation between the classes $\mathsf{NC}^1[\mathsf{NL}/n^{o(1)}]$ and $\mathsf{NC}^1_{n^{o(1)}}$. The author's conjecture is that neither class includes the other.

4.3 Extending Fortnow's lower bound

In this section, Fortnow's time-space tradeoff for $\overline{\text{SAT}}$ is extended to classes with subpolynomial advice.

Theorem 4.6 *Let $r(n)$ be any unbounded function such that $r(n) \in O(\frac{\log n}{\log\log n})$. Then for all $\epsilon > 0$,*

$$\overline{\mathrm{SAT}} \notin \mathsf{NTIME}(n^{1+\frac{1}{r(n)}})/n^{o(1)} \cap \mathsf{NTISP}(n^{o(r(n))}, n^{1-\epsilon})/n^{o(1)}.$$

More concretely, Theorem 4.6 gives the following time-space tradeoff for $\overline{\mathrm{SAT}}$:

Theorem 4.7 *For any $\epsilon > 0$,*

$$\overline{\mathrm{SAT}} \notin \mathsf{NTISP}(n^{1+o(1)}, n^{1-\epsilon})/n^{o(1)}.$$

An alternate proof of the latter theorem is given in the next chapter.

The arguments used in the proof of Theorem 4.6 are essentially the same as those used by Fortnow to prove his original result (Theorem 3.2 in [26]). The essential difference lies in the introduction of a diagonal argument similar to the one used in the proofs of Theorems 3.7, 4.1 and 4.4 in order to proceed to the non-uniform case.

As in Fortnow's proof, a version of Nepomnjaščiĭ's Theorem is required. The non-uniform version Lemma 4.2 suffices. Moreover, a version of Fortnow's lemma giving sufficient conditions for collapsing a non-constant number of levels of the polynomial hierarchy is required:

Lemma 4.4 *Let $r(n)$ be an unbounded function such that $r(n) \in O(\frac{\log n}{\log\log n})$. Suppose furthermore that $\overline{\mathrm{SAT}} \in \mathsf{NTIME}(n^{1+\frac{1}{r(n)}})/n^{o(1)}$. Then for any $\gamma > 0$, there exists a constant ℓ such that*

$$\Sigma^{n \log n}_{\frac{r(n)}{\ell}} \subseteq \mathsf{coNTIME}(n^{1+\gamma})/n^{o(1)}.$$

Before giving the proof for Lemma 4.4, it is insightful to recall the proof of its uniform equivalent in Fortnow [26]:

Lemma 4.5 (Fortnow) *Let $r(n)$ be an unbounded function such that $r(n) = O(\frac{\log n}{\log\log n})$. Suppose furthermore that $\overline{\mathrm{SAT}} \in \mathsf{NTIME}(n^{1+\frac{1}{r(n)}})$. Then for any $\gamma > 0$, there exists a constant ℓ such that*

$$\Sigma^{n \log n}_{\frac{r(n)}{\ell}} \subseteq \mathsf{coNTIME}(n^{1+\gamma}).$$

The argument behind Fortnow's lemma is as follows: fix $\gamma > 0$ and for simplicity, consider the case where $r(n) = \frac{\log n}{\log\log n}$ so that $n^{1+\frac{1}{r(n)}} = n \log n$. The goal is to show that if $\overline{\mathrm{SAT}} \in \mathsf{NTIME}(n \log n)$, then for some ℓ,

$$\Sigma^{n \log n}_{\frac{\log n}{\ell \log\log n}} \subseteq \mathsf{coNTIME}(n^{1+\gamma}).$$

Fortnow observes that if $A \in \Sigma^{n \log n}_{\frac{\log n}{\ell \log\log n}}$, then $\bar{A} \in \Sigma^{n \log n}_{\frac{\log n}{\ell' \log\log n}}$ for any $\ell' < \ell$. So it suffices to show

$$\Sigma^{n \log n}_{\frac{\log n}{\ell \log\log n}} \subseteq \mathsf{NTIME}(n^{1+\gamma}).$$

The idea is to use the fact that $\overline{\mathrm{SAT}} \in \mathsf{NTIME}(n \log n)$ to convert a propositional formula with $k-1$ quantifier alternations into a formula with $k-2$ alternations without substantially increasing the size of the formula. Hence, repeatedly applying the reduction gives a polynomial-size formula with no alternations equivalent to the original formula.

This is done as follows. Let M be a nondeterministic machine computing $\overline{\mathrm{SAT}}$ in $O(n \log n)$ time. Lemma 1.3 then implies that there exists a constant a and a function ϕ mapping inputs x, $|x| = n$, of M

to propositional formulas of length $at(n)\log^4 t(n)$ such that M accepts x if and only if $\phi(x)$ is in SAT. Moreover, ϕ is computable in $O(t(n)\log^4 t(n))$ time. Now let ψ_k be a formula of length m_k with $k-1$ alternations beginning with $\exists$. It suffices to consider the case where k is even as the case where k is odd is similar. So ψ_k is of the form

$$\psi_k = \exists\bar{x}_1\forall\bar{x}_2\ldots\exists\bar{x}_{k-1}\forall\bar{x}_k\phi(\bar{x}_1,\ldots,\bar{x}_k),$$

Consider the formula

$$\tau(\bar{x}_1,\ldots,\bar{x}_{k-1}) = \forall\bar{x}_k\phi(\bar{x}_1,\ldots,\bar{x}_k).$$

Note that the truth of $\tau(\bar{x}_1,\ldots,\bar{x}_{k-1})$ is a $\overline{\text{SAT}}$ question of length bounded by m_k. So, the truth of $\tau(\bar{x}_1,\ldots,\bar{x}_{k-1})$ is equivalent to the truth of

$$\phi(\tau(\bar{x}_1,\ldots,\bar{x}_{k-1})) = \exists\bar{y}\sigma(\bar{x}_1,\ldots,\bar{x}_{k-1},\bar{y}),$$

where the size of σ is bounded by $am_k\log^4 m_k$. So ψ_k is equivalent to

$$\psi_{k-1} = \exists\bar{x}_1\forall\bar{x}_2\ldots\exists\bar{x}_{k-1}\exists\bar{y}\sigma(\bar{x}_1,\ldots,\bar{x}_{k-1},\bar{y}),$$

a formula with $k-2$ alternations and length m_{k-1} bounded by $am_k\log^4 m_k$. Pad the encoding of ψ_{k-1} to ensure that $m_{k-1} = am_k\log^4 m_k$. In particular, this ensures when reducing a formula ψ, the lengths of the intermediate formulas (i.e., the m_i) depend only on the length of ψ. This will be important in the proof of the non-uniform version Lemma 4.4.

Let $s(n) = \frac{\log n}{\ell\log\log n}$. By induction, Fortnow shows that for $\ell > \frac{16}{\gamma}$, a formula of length $n\log^4 n$ with $s(n)$ alternations can be reduced by the above method to a satisfiability question of length $m_1 = n^{1+\frac{\gamma}{2}}$.

Now, if $A \in \Sigma_{s(n)}^{n\log n}$, Lemma 1.3 says that asking if $x \in A$ is equivalent to asking if an $s(n)-1$ alternation formula $\psi_{s(n)}$ of length $n\log^4 n$ is true. But by the above the latter is equivalent to asking whether a formula with no alternations of length $n^{1+\frac{\gamma}{2}}$ is in SAT, which is certainly in $\mathsf{NTIME}(n^{1+\gamma})$.

A sketch of the proof of Lemma 4.4 can now be given. Note first that as opposed to Fortnow's lemma, the assumption is now that $\overline{\text{SAT}} \in \mathsf{NTIME}(n\log n)/n^{o(1)}$. So instead of there being a nondeterministic machine solving $\overline{\text{SAT}}$ in $O(n\log n)$ time, there exist a nondeterministic machine M running in time $O(n\log n)$ and a function $h:\mathbb{N}\to\Sigma^*$ such that $|h|$ is subpolynomial and $x\in\overline{\text{SAT}}$ if and only if M accepts $\langle x, h(|x|)\rangle$. Hence, for a machine M' to do the reductions required in the proof of Fortnow's lemma, it requires on inputs of size n to have the subpolynomial advice required by M to solve $\overline{\text{SAT}}$ on instances of size $m_{s(n)}, m_{s(n)-1},\ldots,m_2$, and m_1, where $s(n)$ is as defined above, $m_{s(n)} = n\log^4 n$ and $m_{k-1} = am_k\log^4 m_k$, $1 < k \le s(n)$. Since $m_1 \in n^{O(1)}$ and $s(n)\in O(\log n)$, the advice required is the product of two subpolynomial functions. But the product of two subpolynomial functions is also subpolynomial, and hence, the advice required by M' is subpolynomial. Lemma 4.4 now follows.

Proof of Theorem 4.7. Fix $r(n)$ and ϵ, and let ℓ be the constant guaranteed by Lemma 4.4 for $\gamma = \epsilon/4$. Let $\alpha(n)$ be any function such that $\alpha(n)\in o(r(n))$. Construct a language L in $\Sigma_{\frac{r(n)}{\ell}}^{n\log n} - (\Sigma_{O(\alpha(n))}^{n}/n^{o(1)})$ by arguing as in the proof of Theorem 3.5. Now assume that

$$\overline{\text{SAT}} \in \mathsf{NTIME}(n^{1+\frac{1}{r(n)}})/n^{o(1)} \cap \mathsf{NTISP}(n^{\alpha(n)}, n^{1-\epsilon})/n^{o(1)}.$$

A contradiction is then derived by showing that $L \in \Sigma_{O(\alpha(n))}^{n}/n^{o(1)}$.

Since $\overline{\text{SAT}} \in \mathsf{NTIME}(n^{1+\frac{1}{r(n)}})$, Lemma 4.4 implies that $\overline{L}$ is in $\mathsf{NTIME}(n^{1+\gamma})/n^{o(1)}$. In particular, there exist a nondeterministic Turing machine M_1 running in time $n^{1+\gamma}$ and a function $h : \mathbb{N} \to \Sigma^*$ such that $|h|$ is subpolynomial and x is in L if and only if M_1 accepts $\langle x, h(|x|)\rangle$. Note that $n^{1+\gamma} \log^{O(1)} n^{1+\gamma} \in O(n^{1+2\gamma})$. By Lemma 1.3, there exists a constant b such that for each input x, $|x| = n$, there exists a propositional formula $\phi(x)$ of length $bn^{1+2\gamma}$ such that x is in $L(M_1)$ if and only if $\phi(x)$ is satisfiable. Moreover, there exists a k, independent of n, such that the ith bit of $\phi(x)$ can be computed in $O(\log^k n)$ time and $O(\log n)$ space provided that there is random-access to x.

Since $\overline{\text{SAT}} \in \mathsf{NTISP}(n^{\alpha(n)}, n^{1-\epsilon})/n^{o(1)}$, there exist a non-deterministic machine M_2 running in $n^{\alpha(n)}$ time and using $n^{1-\epsilon}$ space and a function $h' : \mathbb{N} \to \Sigma^*$ such that $|h'|$ is subpolynomial and x is in $\overline{\text{SAT}}$ if and only if M_2 accepts $\langle x, h(|x|)\rangle$. This fact together with the properties enjoyed by ϕ will be used to construct a nondeterministic machine that accepts $\overline{\text{SAT}}$ in $n^{1+o(1)}$ time and $n^{1-\epsilon}$ space.

To that end, consider a random-access nondeterministic machine M_3 that computes as follows on input $\langle x, h'(b|x|^{1+2\gamma})\rangle$: M_3 computes like M_2 on input $\langle \phi(x), h'(b|x|^{1+2\gamma})\rangle$. However, M_3 does not actually write down $\phi(x)$; instead, it computes each bit of $\phi(x)$ as required by the simulation of M_2. As discussed above, it takes $O(\log^k n)$ time to compute a bit of $\phi(x)$. Hence, the simulation of M_2 by M_3 takes time $t(n) < (n^{1+2\gamma})^{\alpha(n^{1+2\gamma})} \log^k n$ and requires $s(n) < (n^{1+2\gamma})^{1-\epsilon}$ space. Note that since $\alpha(n) \in o(\log n)$, it follows that $t(n) \in n^{O(\alpha(n))}$. Moreover, since $\gamma = \epsilon/4$, it follows that $s(n) \in O(n^{1-\delta})$ for some $\delta > 0$. Finally, $|h'(bn^{1+2\gamma})|$ is subpolynomial, and so,

$$L(M_1) \in \mathsf{NTISP}(n^{O(\alpha(n))}, n^{1-\delta})/n^{o(1)}.$$

It is easy to see that this implies that in fact $L \in \mathsf{NTISP}(n^{O(\alpha(n))}, n^{1-\delta})/n^{o(1)}$.

Lemma 4.2 then implies that $L \in \Sigma^n_{O(\alpha(n))}/n^{o(1)}$, a contradiction. ∎

The nondeterministic time-space lower bounds for $\overline{\text{SAT}}$ presented in this section are generalized in the next chapter using techniques due to Lipton and Viglas [46].

4.4 Discussion

As mentioned at the end of section 4.1, the circuit lower bounds presented in that section can be improved using the arguments employed by Fortnow and in section 4.3 to obtain lower bounds for $\overline{\text{SAT}}$ on nondeterministic machines. Indeed, the following theorem due to van Melkebeek [63] can be proved.

Theorem 4.8 (van Melkebeek) *For any $\epsilon > 0$, SAT cannot both have $\mathsf{NTISP}(n^{O(1)}, n^{1-\epsilon})$-uniform circuits of size $n^{1+o(1)}$ and be in $\mathsf{NTISP}(n^{O(1)}, n^{1-\epsilon})$.*

The theorem implies that SAT does not have $\mathsf{NTISP}(n^{O(1)}, n^{1-\epsilon})$-uniform circuits of size $n^{1+o(1)}$ and width $n^{1-\epsilon}$, nor does SAT have $\mathsf{NTISP}(n^{O(1)}, n^{1-\epsilon})$-uniform circuits of size $n^{1+o(1)}$ and logarithmic depth. In addition, SAT does not have $\mathsf{NTISP}(n^{O(1)}, n^{1-\epsilon})$-uniform branching programs of $n^{1+o(1)}$ size.

In combination with the non-uniform arguments from this chapter, van Melkebeek's proof can then be adapted to obtain the following theorem:

Theorem 4.9 *For any $\epsilon > 0$, SAT cannot both have $(\mathsf{NTISP}(n^{O(1)}, n^{1-\epsilon})/n^{o(1)})$-uniform circuits of size $n^{1+o(1)}$ and be in $\mathsf{NTISP}(n^{O(1)}, n^{1-\epsilon})/n^{o(1)}$.*

Chapter 5

Lower bounds for SAT. Part II

In the previous chapter, non-uniform extensions of Fortnow's time-space tradeoffs for SAT [26] were presented. In this chapter, the related work of Lipton and Viglas [46] is generalized and extended to classes with subpolynomial advice.

Lipton and Viglas proved the existence of a language computable in linear nondeterministic time not computable by a random-access deterministic Turing machine running in $n^{\sqrt{2}-\epsilon}$ time and $n^{o(1)}$ space for any $\epsilon > 0$. A corollary of this result is that for all $\epsilon > 0$, SAT, infinitely often, cannot be computed by a random-access deterministic Turing machine in $n^{\sqrt{2}-\epsilon}$ time and $n^{o(1)}$ space.

The proofs of Lipton and Viglas are based on surprisingly simple and standard arguments. As with Fortnow's work, Lipton and Viglas built on proof techniques first used by Kannan [43] to show that there exists an integer k such that for all integers $j \neq k$, $\mathsf{NTIME}(n^j)$ properly contains $\mathsf{DTISP}(n^j, o(n^{j/k}))$. More specifically, beginning from the assumption that linear nondeterministic time computations can be done deterministic n^α time and $n^{o(1)}$, Lipton and Viglas showed that for some polynomial $t(n)$, nondeterministic time $t(n)$ is contained in nondeterministic time $t^{\frac{\alpha^2}{2}+o(1)}$. The nondeterministic time hierarchy then implies a lower bound for α, thereby yielding their separation. This basic framework is extended in this chapter.

This chapter is structured as follows. In section 5.1 the techniques of Lipton and Viglas, in conjunction with some ideas from Kannan [43], are used to show that for any positive $\epsilon \leq 1$ and any $\delta > 0$, SAT cannot be computed by a random-access multi-tape deterministic Turing machine using $n^{\frac{1}{2}(\sqrt{\epsilon^2+8}-\epsilon)-\delta}$ time, n^ϵ space and having access to subpolynomial advice. In addition, a proof showing that

$$\mathsf{DTISP}(n^r, n^{1-\epsilon}) \subsetneq \mathsf{NTIME}(n^r),$$

for all rational $r \geq 1$ and all $\epsilon > 0$ is given. The above two facts follow from a general theorem showing that for all rational $r \geq 1$ and all ϵ such that $0 < \epsilon \leq 1/r$, if

$$\mathsf{NTIME}(n^r) \subseteq \mathsf{DTISP}(n^{\alpha r}, n^{\epsilon r})/n^{o(1)},$$

then $\alpha \geq \frac{1}{2}(\sqrt{\epsilon^2 + 4 + 4/r} - \epsilon)$.

In section 5.2, the techniques from section 5.1 are generalized to obtain time-space tradeoffs for $\overline{\text{SAT}}$ and co-nondeterministic time on nondeterministic machines taking subpolynomial advice. In particular, an alternate proof of Theorem 4.7 from Chapter 4 is given.

Finally, in section 5.3 lower bounds on the time-space product for deterministic machines taking subpolynomial advice and accepting SAT are given. Specifically, it is shown that if M is a deterministic Turing machine deciding SAT in $t(n)$ time and $s(n)$ space with the aid of subpolynomial advice, then $t(n)s(n) \in \Omega(n^{(\sqrt{17}+1)/4})$. In particular, $t(n)s(n) \in \omega(n^{1.28})$. This improves a similar result by Fortnow [26].

It is worth stating again, as it was at the beginning of the previous chapter, that even though the results in this chapter are stated for non-uniform classes defined using Karp-Lipton advice functions, equivalent results can be obtained with the same techniques for non-uniform classes defined using non-uniform Turing machines.

5.1 Nondeterministic time vs. deterministic time-space

The main result proved in this section is the following theorem:

Theorem 5.1 *For all rational numbers r and ϵ such that $r \geq 1$ and $0 < \epsilon \leq 1/r$, if*

$$\mathsf{NTIME}(n^r) \subseteq \mathsf{DTISP}(n^{\alpha r}, n^{\epsilon r})/n^{o(1)},$$

for some rational number $\alpha \geq 1$, then $r\alpha^2 + \epsilon r\alpha - r - 1 \geq 0$. In particular,

$$\alpha \geq \frac{1}{2}\left(\sqrt{\epsilon^2 + 4 + 4/r} - \epsilon\right).$$

A word should be said about the constants appearing in the statement of the above theorem. The easy way to explain these constants is to remark that they simply result from optimizing the simulations done in the proof of Theorem 5.1. This answer is, of course, not very satisfying. Unfortunately, it is not entirely understood why these proof techniques yield these particular lower bounds; indeed, it would be of interest to develop some more intuition as to why these particular constants are appearing. These remarks also apply to the nondeterministic time-space tradeoffs given for co-nondeterministic computations in the next section.

In conjunction with Lemma 1.3, Theorem 5.1 can be used to obtain time-space lower bounds for SAT on non-uniform machines.

Corollary 5.2 *For all ϵ, $0 < \epsilon < 1$, if* SAT $\in \mathsf{DTISP}(n^\alpha, n^\epsilon)/n^{o(1)}$, *then $\alpha \geq \frac{1}{2}\left(\sqrt{\epsilon^2 + 8} - \epsilon\right)$.*

Proof. Fix ϵ and suppose that SAT $\in \mathsf{DTISP}(n^\alpha, n^\epsilon)/n^{o(1)}$. Then there exist a random-access deterministic machine M running in time $O(n^\alpha)$ and space $O(n^\epsilon)$ and a function h mapping $\mathbb{N}$ to Σ^* such that $|h|$ is subpolynomial and x is in SAT if and only if M accepts $\langle x, h(|x|)\rangle$.

Let L be in $\mathsf{NTIME}(n)$. By Lemma 1.3, there exists a constant b such that for each input x, $|x| = n$, there exists a propositional formula $\phi(x)$ of length $bn\log^2 n$ such that x is in L if and only if $\phi(x)$ is satisfiable. Moreover, the ith bit of $\phi(x)$ can be computed in $O(\log^{O(1)} n)$ time and $O(\log n)$ space provided there is random-access to x.

Now consider a random-access deterministic machine M' that on input $\langle x, h(b|x|\log^2 |x|)\rangle$ computes as follows: M' computes like M on input $\langle \phi(x), h(b|x|\log^2 |x|)\rangle$. However, M' does not actually write $\phi(x)$; instead, it computes each bit of $\phi(x)$ as required by the simulation of M. By the above, it takes $O(\log^{O(1)} n)$ time to compute a bit of $\phi(x)$. Hence, the simulation of M by M' thereby takes time $n^\alpha \log^{O(1)} n$ and requires $n^\epsilon \log^{O(1)} n$ space. Note moreover, that $x \in L$ if and only if M' accepts $\langle x, h(b|x| \log^2 |x|)\rangle$. Hence, $L \in \mathsf{DTISP}(n^\alpha \log^{O(1)} n, n^\epsilon \log^{O(1)} n)/n^{o(1)}$. Since $L \in \mathsf{NTIME}(n)$ is arbitrary, it follows that $\mathsf{NTIME}(n) \subseteq \mathsf{DTISP}(n^\alpha \log^{O(1)} n, n^\epsilon \log^{O(1)} n)/n^{o(1)}$. Theorem 5.1 then implies the desired lower bound for α. ■

In particular, Corollary 5.2 implies the following lower bounds for SAT:

$$\begin{aligned}
&\text{SAT} \notin \mathsf{DTISP}(n^{1+o(1)}, n^{1-\epsilon})/n^{o(1)}, \epsilon > 0,\\
&\text{SAT} \notin \mathsf{DTISP}(n^{1.186}, \sqrt{n})/n^{o(1)},\\
&\text{SAT} \notin \mathsf{DTISP}(n^{\sqrt{2}-\epsilon}, n^{o(1)})/n^{o(1)}, \epsilon > 0.
\end{aligned}$$

It seems likely that separating $\mathsf{DTIME}(n^k)$ and $\mathsf{NTIME}(n^k)$ should be less difficult than separating P and NP. Indeed, using different techniques than those used in this thesis, Paul, Pippenger, Szemerédi and Trotter [51] were able to separate $\mathsf{DTIME}(n)$ and $\mathsf{NTIME}(n)$ for multi-tape Turing machines. Theorem 5.1 implies the following separation in the general case:

Theorem 5.3 *For all rational numbers $r \geq 1$ and all $\epsilon > 0$,*

$$\mathsf{DTISP}(n^r, n^{1-\epsilon}) \subsetneq \mathsf{NTIME}(n^r).$$

Note that in the case where $r = 1$, Theorem 5.3 only says that $\mathsf{DTISP}(n, n^{1-\epsilon})$ is properly contained in $\mathsf{NTIME}(n)$ and thus appears weaker than the Paul, Pippenger, Szemerédi and Trotter result; however, Theorem 5.3 holds for random-access multi-tape Turing machines, whereas the separation from [51] only holds for multi-tape Turing machines. Note the similarity between Theorem 5.3 and Kannan's result [43] showing the existence of a positive integer ℓ such that for all positive integers $k \neq \ell$, $\mathsf{DTISP}(n^k, o(n^{k/\ell}))$ is properly contained in $\mathsf{NTIME}(n^k)$.

Proof of Theorem 5.1

The arguments used to prove Theorem 5.1 are very closely based on the arguments used by Kannan [43] and Lipton and Viglas [46]. As in those two papers, a translation lemma is needed.

Lemma 5.1 *For all rational numbers $r \geq 1$, if $\mathsf{coNTIME}(n^r) \subseteq \mathsf{NTISP}(n^{\alpha r}, n^{\beta r})/n^{o(1)}$ for some $\alpha \geq 1$ and $\beta > 0$, then for any rational number $q > r$, $\mathsf{coNTIME}(n^q) \subseteq \mathsf{NTISP}(n^{\alpha q} \log n, n^{\beta q})/n^{o(1)}$.*

Proof. Fix r, q, α and β, and suppose that $\mathsf{coNTIME}(n^r) \subseteq \mathsf{NTISP}(n^{\alpha r}, n^{\beta r})/n^{o(1)}$. Let $L \subseteq \Sigma^*$ be in $\mathsf{coNTIME}(n^q)$. Consider the language

$$L' = \{x\#^{f(|x|)-|x|} : x \in L\},$$

where $\#$ is not a symbol in Σ and $f(n) = n^{q/r}$. Note that L' is in $\mathsf{coNTIME}(n^r)$ and hence in $\mathsf{NTISP}(n^{\alpha r}, n^{\beta r})/n^{o(1)}$. In particular, there exist a Turing machine M with j work tapes that runs in $n^{\alpha r}$ time and $n^{\beta r}$ space and a function h mapping $\mathbb{N}$ to Σ^* such that $g(n) = |h(n)|$ is constructible and subpolynomial, and $x \in L$ if and only if M accepts $\langle x, h(|x|)\rangle$.

Note that $|h \circ f| = g \circ f$ is subpolynomial. So to prove the lemma it suffices to exhibit a Turing machine M' running in time $O(n^{\alpha q} \log n)$ and space $O(n^{\beta q})$ such that x is in L if and only if $\langle x, h(f(|x|)) \rangle$ is accepted by M'. To that end, consider a Turing machine M' with $j+2$ work tapes that on input $\langle x, y \rangle$ of length n, computes as follows:

1. Compute $|x|$, $|y|$, $f(|x|)$ and $g(f(|x|))$, and write $f(|x|)$ onto the first tape. Check that $|y| = g(f(|x|))$. Since $g \circ f$ is sublinear, the latter implies that $|x| \in \Theta(n)$.

2. Simulate M on input $z = \langle x \#^{f(|x|)-|x|}, y \rangle$ on the last j tapes and accept iff M accepts. Note that $|z| \in \Theta(f(|x|)) = \Theta(f(n))$. Machine M' does not actually write down the input z. Instead, it uses its second tape to write down in binary the position of M's input head. The second tape is updated accordingly after each simulated move of M. In particular, the second tape initially has the number 0 written on it. Whenever M's input head must be moved left (right) during the simulation, the number on the second tape is decremented (incremented) by 1. If the value i on the second tape is greater than n, then the symbol under M's (simulated) input head must be #; otherwise, the symbol under M's (simulated) input head is the ith symbol on M''s input tape.

Since g is time-constructible, the first step can be done in linear time and logarithmic space with the aid of Proposition 1.2. Note that it takes $O(\log n)$ for M' to simulate a step of M since M' has to update the index on the second tape after each simulated step. Hence, the simulation of M by M' takes time $O(f(n)^{\alpha r} \log n)$, which is $O(n^{\alpha q} \log n)$, and space $O(f(n)^{\beta r})$, which is $O(n^{\beta q})$.

Hence, M' runs in time $O(n^{\alpha q} \log n)$ and space $O(n^{\beta q})$. Moreover, as desired, x is in L iff M' accepts $\langle x, h(f(|x|)) \rangle$. ■

Note that by arguing more carefully, the extra log factor appearing in the conclusion of the above lemma can be eliminated; however, this extra precision is not required to prove Theorem 5.1, and thus the tighter result is not included here.

Proof of Theorem 5.1. Fix $r \geq 1$ and $\epsilon \leq 1/r$ and assume that

$$\mathsf{NTIME}(n^r) \subseteq \mathsf{DTISP}(n^{\alpha r}, n^{\epsilon r})/n^{o(1)}. \tag{5.1}$$

This assumption will be used to show that for some rational c any machine running in co-nondeterministic time n^c can be simulated by a nondeterministic machine running in time $n^{c\alpha r \frac{\alpha+\epsilon}{r+1} + o(1)}$ and taking subpolynomial advice. This fact coupled with Theorem 3.5 from Chapter 3 then gives the desired lower bound for α.

Let

$$c = \max(1/(\alpha - \beta + \epsilon), r),$$

where

$$\beta = \frac{r}{r+1}(\alpha + \epsilon).$$

Note that $\beta = r(\alpha - \beta + \epsilon)$. Let $L \in \mathsf{coNTIME}(n^c)$ be arbitrary. Now, since L is in $\mathsf{coNTIME}(n^c)$, Equation (5.1) and Lemma 5.1 imply the existence of a deterministic Turing machine M running in time $O(n^{c\alpha} \log n)$ and space $O(n^{c\epsilon})$, and a function $h : \mathbb{N} \to \Sigma^*$ such that $|h|$ is subpolynomial and $x \in L$ if and only if M accepts $\langle x, h(|x|) \rangle$. In addition assume, without loss of generality, that there exists a constant a such that M takes exactly $an^{c\alpha}\log n$ steps on each path for all inputs of length n. Moreover, assume without loss of generality, that there exists a constant b such that all configurations of M on input of length n can be described by $bn^{c\epsilon}$ bits.

Note that the computation of M can be simulated by a Σ_2 machine M' that existentially guesses the configurations $A_0, A_1, \ldots, A_{n^{c(\alpha-\beta)}}$ of M every $n^{c\beta}\log n$ steps, universally guesses an $i < n^{c(\alpha-\beta)}$, and then checks deterministically whether $A_i \vdash_M A_{i+1}$. By Equation (5.1) and Lemma 5.1, the "universal" stage in the Σ_2 machine M' can be replaced by a deterministic computation taking subpolynomial advice, incurring a small time loss, to yield a nondeterministic machine M'' that takes subpolynomial advice and accepts $L(M)$. A nondeterministic machine taking subpolynomial advice that accepts L and runs in the same time as M'' can then be constructed from M''. So the goal is to construct M'' so that it runs as fast as possible.

To that end, begin by constructing a co-nondeterministic machine M_1 that given a string $A_0, \ldots, A_\ell$ of configurations of M, universally guesses an $i < \ell$ and checks that $A_i \vdash_M^* A_{i+1}$. In particular, consider a machine M_1 that on input w of length n computes as follows:

1. Check that w is of the form $x\#A_0\#A_1\#\cdots\#A_\ell$. Let $m = |x|$ and compute $bm^{c\epsilon}$, $m^{c(\alpha-\beta)}$ and $am^{c\beta}\log m$. Check that $\ell = m^{c(\alpha-\beta)}$ and that each of the A_i has length $bm^{c\epsilon}$.

2. Universally guess an $i < \ell$.

3. Check that $A_i \vdash_M^{am^{c\beta}\log m} A_{i+1}$

The first step can be done in linear time with the aid of Proposition 1.2. The second step also takes linear time. Finally, the third step takes time $O(m^{c\beta}\log m)$ which is $O(m^{cr(\alpha-\beta+\epsilon)}\log m)$. Note that $n \in \Theta(m + m^{c(\alpha-\beta+\epsilon)})$. But $c \geq 1/(\alpha-\beta+\epsilon)$, and so $n \in \Theta(m^{c(\alpha-\beta+\epsilon)})$. Hence, the third step takes time $O(n^r \log n)$, and M_1 runs in time $O(n^r \log n)$. Equation (5.1) and Lemma 5.1 then imply the existence of a nondeterministic machine M_2 running in time $O(n^{\alpha r + o(1)})$ and a function $h' : \mathbb{N} \to \Sigma^*$ such that $|h'|$ is subpolynomial and $x \in L(M_1)$ if and only if M_2 accepts $\langle x, h'(|x|)\rangle$.

The nondeterministic machine M'' discussed above that accepts $L(M)$ with the aid of subpolynomial advice can now be defined. Since M'' will have to simulate M_2, it will require advice. It turns out that the advice M'' requires on input of length n is $h' \circ f(n)$, where

$$f(n) = n + n^{c\epsilon} + bn^{c(\alpha-\beta+\epsilon)}.$$

Note that since f is a polynomial, $|h' \circ f|$ is subpolynomial. On input v of length n, M'' computes as follows:

1. Check that v is of the form $\langle x, y\rangle$. Let $m = |x|$. Check that $|y| = |h'(f(|x|))|$ and compute $bm^{c\epsilon}$ and $am^{c(\alpha-\beta)}$.

2. Existentially guess $am^{c(\alpha-\beta)}$ configurations $A_0, A_1, \ldots, A_{am^{c(\alpha-\beta)}}$ each of length $bm^{c\epsilon}$ of M on input x where A_0 is the initial configuration of M on x and $A_{am^{c(\alpha-\beta)}}$ is an accepting configuration.

3. Let $z = x\#A_1\#A_2\#\ldots\#A_{am^{c(\alpha-\beta)}}$ and simulate M_2 on $w = \langle z, y\rangle$. Note that $|y| = |h'(|z|)|$.

Note that since $|h' \circ f|$ is sublinear, $n \in \Theta(m)$. By Proposition 1.2, the first step can be done in linear time. The second step takes time $O(m^{c(\alpha-\beta+\epsilon)})$ which is $O(n^{c(\alpha-\beta+\epsilon)})$. Since M_2 runs in $O(n^{\alpha r+o(1)})$ time and $w \in \Theta(z)$, the third step takes time $O(|z|^{\alpha r+o(1)})$ which is $O(n^{c\alpha r(\alpha-\beta+\epsilon)+o(1)})$. Hence, M'' runs in time $O(n^{c\alpha r(\alpha-\beta+\epsilon)+o(1)})$. By definition of β, the latter is $O(n^{c\alpha r\frac{\alpha+\epsilon}{r+1}+o(1)})$.

Note that M accepts w if and only if M'' accepts $\langle w, h'(f(|w|))\rangle$. Since $|h' \circ f|$ is subpolynomial, it follows that $L(M) \in \mathsf{NTIME}(n^{c\alpha r\frac{\alpha+\epsilon}{r+1}+o(1)})/n^{o(1)}$. But, x is in L if and only if M accepts $\langle x, h(|x|)\rangle$. It is easy to see that this implies that L is in $\mathsf{NTIME}(n^{c\alpha r\frac{\alpha+\epsilon}{r+1}+o(1)})/n^{o(1)}$.

Since $L \in \mathsf{coNTIME}(n^c)$ is arbitrary, it follows that

$$\mathsf{coNTIME}(n^c) \subseteq \mathsf{NTIME}(n^{c\alpha r\frac{\alpha+\epsilon}{r+1}+o(1)})/n^{o(1)}.$$

Theorem 3.5 then implies that $\alpha r\frac{\alpha+\epsilon}{r+1} \geq 1$. ■

The statement of Theorem 5.1 was originally weaker; Dieter van Melkebeek [63] showed how to alter the original arguments to prove the current version.

5.2 Co-nondeterministic time vs. nondeterministic time-space

The techniques used to prove Theorem 5.1 can be generalized to obtain similar, but slightly weaker, time-space tradeoffs for co-nondeterministic time on nondeterministic machines taking subpolynomial advice.

Theorem 5.4 *Let α, r and ϵ be rational numbers such that $\alpha \geq 1$, $r \geq 1$ and $0 < \epsilon \leq \alpha/(r+\alpha-1)$. Suppose furthermore that*

$$\mathsf{coNTIME}(n^r) \subseteq \mathsf{NTISP}(n^{\alpha r}, n^{\epsilon r})/n^{o(1)}.$$

Then $r\alpha^3 + \epsilon r\alpha^2 - \alpha - r \geq 0$.

When $r = 1$ the following separation is obtained:

Corollary 5.5 *Let α and ϵ be rational numbers such that $\alpha \geq 1$ and $0 < \epsilon \leq 1$. Suppose furthermore that $\mathsf{coNTIME}(n) \subseteq \mathsf{NTISP}(n^\alpha, n^\epsilon)/n^{o(1)}$. Then $\alpha^3 + \epsilon\alpha^2 - \alpha - 1 \geq 0$. In particular,*

$$\alpha \geq \frac{1}{6}\left(\sqrt[3]{\gamma} + \frac{12+4\epsilon^2}{\sqrt[3]{\gamma}} - 2\epsilon\right),$$

where $\gamma = -8\epsilon^3 - 36\epsilon + 108 + 12\sqrt{69 - 54\epsilon - 3\epsilon^2 - 12\epsilon^3}$.

As noted above, the lower bound obtained in Theorem 5.4 for co-nondeterministic time on nondeterministic machines taking subpolynomial advice is not as strong as the lower bound obtained in Theorem 5.1 for nondeterministic time on deterministic machines taking subpolynomial advice. This merits some brief comments. Recall that in the proof of Theorem 5.1 a co-nondeterministic machine is efficiently simulated by first converting it into a deterministic machine using small amounts of space and then by simulating the deterministic machine efficiently using a Σ_2 machine. The hypotheses for Theorem 5.4, however, do not allow converting a co-nondeterministic machine into a deterministic machine using small amounts of space; instead, the machine can only be converted into a *nondeterministic* machine M using small amounts of space. So to efficiently simulate M, a Σ_3 machine is used: first existentially guess a sequence of configurations for M, then universally pick a pair of these configurations and existentially guess a path between these adjacent configurations. Observe that the key difference is that in the deterministic case, the last step in this simulation can be done deterministically thereby eliminating an alternation in the simulation. It is this extra alternation that ultimately results in the weaker lower bound for co-nondeterministic time on nondeterministic machines taking advice.

Lower bounds for $\overline{\text{SAT}}$ can be obtained from Corollary 5.5 using Lemma 1.3 and arguing as in the proof of Corollary 5.2.

Corollary 5.6 *For all ϵ, $0 < \epsilon < 1$, if $\overline{\text{SAT}} \in \mathsf{NTISP}(n^{\alpha}, n^{\epsilon})/n^{o(1)}$, then $\alpha^3 + \epsilon\alpha^2 - \alpha - 1 \geq 0$.*

In particular, $\overline{\text{SAT}}$ cannot be computed by a nondeterministic Turing machine using $n^{1.324}$ time and subpolynomial space and advice, nor by a nondeterministic Turing machine using $n^{1.142}$ time, $\sqrt{n}$ space and subpolynomial advice. Neither of these results follows from Fortnow's [26] lower bounds for $\overline{\text{SAT}}$. Note also that Corollary 5.6 generalizes Theorem 4.7, the non-uniform extension of Fortnow's [26] time-space tradeoff for $\overline{\text{SAT}}$ from Chapter 4.

Proof of Theorem 5.4

In addition to Lemma 5.1, the following technical lemma is required for the proof of Theorem 5.4:

Lemma 5.2 *If $\mathsf{coNTIME}(n^a) \subseteq \mathsf{NTIME}(n^b)/n^{o(1)}$ for some constants a and b, then*

$$\mathsf{coNTIME}(n^a)/n^{o(1)} \subseteq \mathsf{NTIME}(n^b)/n^{o(1)}.$$

Proof. Assume the hypotheses and let $L \in \mathsf{coNTIME}(n^a)/n^{o(1)}$. Then there exist a nondeterministic machine M running in time n^a and a function h mapping $\mathbb{N}$ to Σ^* such that $|h|$ is constructible and subpolynomial, and $x \in L$ if and only if M accepts $\langle x, h(|x|)\rangle$. Then by the hypotheses there exist a deterministic Turing machine M' running in time n^b and a function h' mapping $\mathbb{N}$ to Σ^* such that $|h'|$ is constructible and subpolynomial, and M accepts y if and only if M' accepts $\langle y, h'(|y|)\rangle$.

Define $g(n)$ to be the length of $\langle x, y\rangle$ when $|x| = n$ and $h(|n|)$. Since $|\langle x, y\rangle|$ only depends on $|x|$ and $|y|$, g is well-defined. Now consider the function

$$h''(n) = \langle h(n), h'(g(n))\rangle.$$

Note that since $|\langle x, y\rangle| \leq 4(|x| + |y|)$, and $|h|$ and $|h'|$ are subpolynomial, it follows that $|h''(n)|$ is also subpolynomial.

Consider a deterministic machine M'' that on input $w = \langle x, h''(|x|)\rangle$, $|w| = n$, computes as follows:

1. Check that the input has the right form and write $h(|x|)$ and $h'(g(|x|))$.

2. Simulate M' on input $z = \langle y, h'(|y|)\rangle = \langle y, h'(g(|x|))\rangle$ where $y = \langle x, h(|x|)\rangle$.

The first step takes linear time. The second step takes time $|z|^b$. Now since $|h''|$ is sublinear, it follows that $|x| \in \Theta(n)$. Moreover, since $|h|$ and $|h'|$ are subpolynomial, $|z| \in \Theta(|x|)$. Hence $|z| \in \Theta(n)$. So the second step takes time $O(n^b)$ and thus, M'' runs in time $O(n^b)$.

Now, x is in L if and only if M accepts $\langle x, h(|x|)\rangle$ if and only if M' accepts $\langle y, h'(|y|)\rangle$, $y = \langle x, h(|x|)\rangle$, if and only if M'' accepts $\langle x, h''(|x|)\rangle$. Hence $L \in \mathsf{NTIME}(n^b)/n^{o(1)}$. Since $L \in \mathsf{coNTIME}(n^a)/n^{o(1)}$ is arbitrary, the lemma follows. ■

Proof of Theorem 5.4. Fix $r \geq 1$ and $\epsilon \leq \alpha/(r+\alpha-1)$ and assume that

$$\mathsf{coNTIME}(n^r) \subseteq \mathsf{NTISP}(n^{\alpha r}, n^{\epsilon r})/n^{o(1)}. \tag{5.2}$$

This assumption is used to show that for some rational c any machine running in co-nondeterministic time n^c can be simulated by a nondeterministic machine running in time $n^{c\alpha^2 r\frac{\alpha+\epsilon}{\alpha+r}+o(1)}$ and taking subpolynomial advice. Theorem 3.5 then gives the desired lower bound for α.

Let

$$c = \max(1/(\alpha - \beta + \epsilon), 1/\epsilon, r),$$

where $\beta = r\frac{\alpha+\epsilon}{\alpha+r}$. Note that $\alpha\beta = r(\alpha - \beta + \epsilon)$. Moreover, since $\epsilon \leq \alpha/(r+\alpha-1)$, it follows that $\beta \geq \epsilon r$.

Let $L \in \mathsf{coNTIME}(n^c)$ be arbitrary. Now, since L is in $\mathsf{coNTIME}(n^c)$, Equation (5.2) and Lemma 5.1 imply the existence of a nondeterministic Turing machine M running in time $O(n^{c\alpha}\log n)$ and space $O(n^{c\epsilon})$, and a function $h : \mathbb{N} \to \Sigma^*$ such that $|h|$ is subpolynomial and $x \in L$ if and only if M accepts $\langle x, h(|x|)\rangle$. In addition assume, without loss of generality, that there exists a constant a such that M takes exactly $an^{c\alpha}\log n$ steps on each path for all inputs of length n. Moreover, assume without loss of generality, that there exists a constant b such that all configurations of M on input of length n can be described by $bn^{c\epsilon}$ bits.

Note that the computation of M can be simulated by a Σ_3 machine M' that existentially guesses the configurations $A_0, A_1, \ldots, A_{n^{c(\alpha-\beta)}}$ of M every $n^{c\beta}\log n$ steps, universally guesses an $i < n^{c(\alpha-\beta)}$, and then guesses a valid computation path of M of length $n^{c\beta}\log n$ from A_i to A_{i+1}. By Equation (5.2) and Lemma 5.1, the second "existential" stage in the Σ_3 machine M' can be replaced by a co-nondeterministic computation taking subpolynomial advice, incurring a small time loss, to yield a Σ_2 machine M'' that takes subpolynomial advice in its "universal" stage and accepts $L(M)$. Moreover, by Equation (5.2) and Lemmas 5.1 and 5.2, the "universal" stage in M'' can be replaced by a nondeterministic computation taking subpolynomial advice, again incurring a small time loss, to yield a nondeterministic machine M''' that takes subpolynomial advice and accepts $L(M)$. A nondeterministic machine taking subpolynomial advice that accepts L and runs in the same time as M''' can then be constructed from M'''. So the goal is to construct M''' so that it runs as fast as possible.

To that end, begin by constructing a nondeterministic machine M_1 that given two configurations A and B of M, existentially guesses a valid computation path from A to B. In particular, consider a machine M_1 that on input w of length n computes as follows:

1. Check that w is of the form $x\#A\#B$. Let $m = |x|$ and compute $bm^{c\epsilon}$ and $m^{c\beta}\log m$. Check that both A and B are valid configurations of M each of length $bm^{c\epsilon}$.

2. Existentially guess a valid computation path of M with input x from configuration A to B of length $m^{c\beta}\log m$.

The first step can be done in linear time with the aid of Proposition 1.2. The second step can be done in time $O(m^{c\epsilon} + m^{c\beta}\log m)$. Note that since $c \geq 1/\epsilon$, it follows that $n = |w| \in \Theta(m^{c\epsilon})$. Hence, M_1

runs in time $O(n + n^{\beta/\epsilon} \log n)$. Moreover, since $\beta/\epsilon \geq r \geq 1$, the latter is simply $O(n^{\beta/\epsilon} \log n)$. Note now that since $\beta/\epsilon \geq r$, Equation (5.2) and Lemma 5.1 imply that there exist a co-nondeterministic Turing machine M_2 running in time $O(n^{\frac{\alpha\beta}{\epsilon}+o(1)})$ and a function h' mapping $\mathbb{N}$ to Σ^* such that $|h'|$ is subpolynomial and M_1 accepts w if and only if M_2 accepts $\langle x, h'(|w|)\rangle$.

Now construct a co-nondeterministic machine M_3 that given a string $A_0, \ldots, A_\ell$ of configurations of M, universally guesses an $i < \ell$ and checks that $A_i \vdash_M^* A_{i+1}$ by simulating M_2. Note that M_3 requires advice for the simulation of M_2. As will become apparent below, the required advice on input of length n is $h' \circ f(n)$, where

$$f(n) = \frac{2 + n + bn^{c\epsilon}}{n + (b+1)n^{c\epsilon} + bn^{c(\alpha-\beta+\epsilon)}}.$$

Note that f is polynomially bounded, and thus, $|h' \circ f|$ is subpolynomial. The machine M_3 can now be defined. On input w of length n, M_3 computes as follows:

1. Check that w is of the form $\langle x\#A_0\#A_1\# \cdots \#A_\ell, y\rangle$. Let $m = |x|$ and compute $bm^{c\epsilon}$ and $m^{c(\alpha-\beta)}$. Check that $\ell = m^{c(\alpha-\beta)}$ and that each of the A_i has length $bm^{c\epsilon}$. Check that $|y| = |h'(2+m+bm^{c\epsilon})|$.

2. Universally guess an $i < \ell$.

3. Simulate M_2 on input $\langle x\#A_i\#A_{i+1}, y\rangle$.

The first step can be done in linear time with the aid of Proposition 1.2. The second step also takes linear time. In the third step, M_2 is simulated on input of length $\Theta(m + m^{c\epsilon})$ which is $\Theta(m^{c\epsilon})$ since $c \geq 1/\epsilon$. Since M_2 runs in $O(n^{\frac{\alpha\beta}{\epsilon}+o(1)})$ time, the third step thereby takes $O(m^{c\alpha\beta+o(1)})$ time. Note that since $\alpha\beta = r(\alpha - \beta - \epsilon)$, the latter is $O(m^{cr(\alpha-\beta-\epsilon)+o(1)})$. So M_3 runs in time $O(n + m^{cr(\alpha-\beta-\epsilon)+o(1)})$. Note now that $n \in \Theta(m + m^{c(\alpha-\beta+\epsilon)})$, since $|h'|$ is subpolynomial. In fact, since $c \geq 1/(\alpha - \beta + \epsilon)$, $n \in \Theta(m^{c(\alpha-\beta+\epsilon)})$. Hence, M_3 runs in time $O(n^{r+o(1)})$. Now consider the language L' consisting of those x for which M_3 accepts $\langle x, h'(f(|x|))\rangle$. Then, $L' \in \mathsf{coNTIME}(n^{r+o(1)})/n^{o(1)}$ since $|h' \circ f|$ is subpolynomial. Equation (5.2) and Lemmas 5.1 and 5.2 then imply the existence of a nondeterministic machine M_4 running in time $O(n^{\alpha r+o(1)})$ and a function $h'' : \mathbb{N} \to \Sigma^*$ such that $|h''|$ is subpolynomial and $x \in L'$ if and only if M_4 accepts $\langle x, h''(|x|)\rangle$.

The nondeterministic machine M''' discussed above that accepts $L(M)$ with the aid of subpolynomial advice can now be defined. It turns out that the advice M''' requires on input of length n is $h'' \circ f'(n)$, where

$$f'(n) = n + n^{c\epsilon} + bn^{c(\alpha-\beta+\epsilon)}.$$

Note that since f is a polynomial, $|h'' \circ f'|$ is subpolynomial. On input v of length n, M''' computes as follows:

1. Check that v is of the form $\langle x, y\rangle$. Let $m = |x|$. Check that $|y| = |h''(f'(|x|))|$ and compute $bm^{c\epsilon}$ and $am^{c(\alpha-\beta)}$.

2. Existentially guess $am^{c(\alpha-\beta)}$ configurations $A_0, \ldots, A_{am^{c(\alpha-\beta)}}$ each of length $bm^{c\epsilon}$ of M on input x where A_0 is the initial configuration of M on x and $A_{am^{c(\alpha-\beta)}}$ is an accepting configuration.

3. Let $z = x\#A_1\#A_2\# \ldots \#A_{am^{c(\alpha-\beta)}}$ and simulate M_4 on $w = \langle z, y\rangle$. Note that $|y| = |h''(|z|)|$.

Note that since $|h'' \circ f'|$ is sublinear, $n \in \Theta(m)$. By Proposition 1.2, the first step can be done in linear time. The second step takes time $O(m^{c(\alpha-\beta+\epsilon)})$ which is $O(n^{c(\alpha-\beta+\epsilon)})$. Since M_4 runs in $O(n^{\alpha r+o(1)})$ time and $w \in \Theta(z)$, the third step takes time $O(|z|^{\alpha r+o(1)})$ which is $O(n^{c\alpha r(\alpha-\beta+\epsilon)+o(1)})$. Hence, M''' runs in time $O(n^{c\alpha r(\alpha-\beta+\epsilon)+o(1)})$. By definition of β, the latter is $O(n^{c\alpha^2 r\frac{\alpha+\epsilon}{\alpha+r}+o(1)})$.

Note that M accepts w if and only if M''' accepts $\langle w, h''(f'(|w|))\rangle$. Since $|h'' \circ f'|$ is subpolynomial, it follows that $L(M) \in \mathsf{NTIME}(n^{c\alpha^2 r\frac{\alpha+\epsilon}{\alpha+r}+o(1)})/n^{o(1)}$. But, x is in L if and only if M accepts $\langle x, h(|x|)\rangle$. It is easy to see that this implies that L is in $\mathsf{NTIME}(n^{c\alpha^2 r\frac{\alpha+\epsilon}{\alpha+r}+o(1)})/n^{o(1)}$.

Since $L \in \mathsf{coNTIME}(n^c)$ is arbitrary, it follows that

$$\mathsf{coNTIME}(n^c) \subseteq \mathsf{NTIME}(n^{c\alpha^2 r\frac{\alpha+\epsilon}{\alpha+r}+o(1)})/n^{o(1)}.$$

Theorem 3.5 then implies that $\alpha^2 r\frac{\alpha+\epsilon}{\alpha+r} \geq 1$. ∎

Dieter van Melkebeek [63] has observed that by using a Σ_{2k+1} machine to efficiently simulate M in the above proof, time-space tradeoffs as strong as those obtained in Theorem 5.1 for nondeterministic time on deterministic machines taking subpolynomial advice can be obtained for co-nondeterministic time on nondeterministic machines taking subpolynomial advice.

5.3 Time-space products for SAT

Fortnow [26] showed that if M is a deterministic Turing machine accepting SAT in time $t(n)$ and space $s(n)$, then for some $\epsilon > 0$, $t(n)s(n) \in \Omega(n^{1+\epsilon})$. Corollary 5.2 can be used to give an improved time-space product for SAT. The proof is similar to the one used by Fortnow to prove his time-space product.

Theorem 5.7 *Let M be a deterministic Turing machine deciding* SAT *using $t(n)$ time, $s(n)$ space, and subpolynomial advice. Then $t(n)s(n) \in \Omega(n^{(\sqrt{17}+1)/4})$. In particular, $t(n)s(n) \in \omega(n^{1.28})$.*

Proof. Let $0 < \epsilon < 1$, and suppose $s(n) \in \omega(n^\epsilon)$. Then $t(n)s(n) \in \omega(n^{1+\epsilon})$ since M requires linear time to read its input. If instead $s(n) \in O(n^\epsilon)$, then $t(n)s(n) \in \Omega(n^{\frac{1}{2}(\sqrt{\epsilon^2+8}-\epsilon)})$ by Corollary 5.2. Hence, $t(n)s(n) \in \Omega(\min(n^{1+\epsilon}, n^{\frac{1}{2}(\sqrt{\epsilon^2+8}-\epsilon)}))$.

Now note that as ϵ goes from 0 to 1, $n^{1+\epsilon}$ increases while $n^{\frac{1}{2}(\sqrt{\epsilon^2+8}-\epsilon)}$ decreases. Hence, the largest product is obtained when the exponents of the latter two functions are equal, that is, when $2+2\epsilon = \sqrt{\epsilon^2+8}-\epsilon$. Solving for ϵ gives the positive root $\epsilon = (\sqrt{17}-3)/4$, whence $t(n)s(n) \in \Omega(n^{(\sqrt{17}+1)/4})$. ∎

5.4 Discussion

With the aid of a new idea, but still using the same proof framework exploited here, Fortnow and van Melkebeek [27] have been able to obtain stronger uniform time-space tradeoffs for SAT than those obtained in sections 5.1 and 5.2. For instance, they are able to prove the following result:

Theorem 5.8 (Fortnow and van Melkebeek) *For any constant $\alpha \geq 1$,*

$$\mathsf{NTIME}(n) \not\subseteq \mathsf{DTISP}(n^\alpha, o(n^{\frac{1}{2}(\frac{\alpha+2}{\alpha}-\alpha)})).$$

In particular, SAT cannot be solved by a deterministic machine using $n^{1.521}$ time and subpolynomial space. In addition, for appropriate sublinear space bounds $s(n)$, Fortnow and van Melkebeek are able to show that $\mathsf{NTISP}(n, s(n))$ computations cannot be simulated by deterministic machines running in time n^a and subpolynomial space for any a less than the golden ratio. Indeed, Fortnow and van Melkebeek obtain several time-space tradeoffs for nondeterministic time-space essentially by noting that their simulations, much like those in the proofs of the theorems in sections 5.1 and 5.2, do not require as much space as the time they use. However, whereas the approach in this chapter has been to obtain tradeoffs for classes of the type $\mathsf{NTIME}(n^r)$ for all rational $r \geq 1$, Fortnow and van Melkebeek only state tradeoffs for classes defined by linear-time bounded machines. Nevertheless, it is likely that their methods can be used to obtain slightly improved tradeoffs for $\mathsf{NTIME}(n^r)$ over those obtained in this chapter.

All the results presented in [27] admit the subpolynomial advice arguments presented in this chapter. Note that the ability to add more non-uniformity to the results in this chapter or to the results in [27] is mainly hindered by the inability to strengthen Theorem 3.5 to classes taking more advice without sacrificing the tightness of the separation.

Note that it it is consistent with current knowledge that SAT can be computed in linear time by an ordinary deterministic multi-tape Turing machine if no restrictions are placed on the space used.

Chapter 6

Non-uniform lower bounds for the Permanent

While the emphasis in the last two chapters was in adapting the diagonalization techniques from Chapter 3 to obtain lower bounds for Satisfiability on non-uniform machines, in this chapter these diagonalization techniques are used to obtain a lower bound for computing the Permanent on bounded-depth threshold circuits with small descriptors.

In [4] Allender proved the following lower bound on computing the Permanent. Let t be a constructible function such that for all k, $t^{(k)}(n) = o(2^n)$. Then the Permanent cannot be computed by bounded-depth Dlogtime-uniform threshold circuits of size $t(n)$. In this chapter, this lower bound is extended to bounded-depth threshold circuits with subpolynomial descriptors.

The chapter outline is as follows. Section 6.1 defines and discusses all the classes and computation models that are used in this chapter. In particular, threshold Turing machines are introduced in section 6.1. Section 6.2 revisits the topic of circuit descriptive size from Chapter 2 in order to give a definition of circuit descriptive size for bounded-depth circuits and in order to provide some relations between uniform bounded-depth threshold circuits and bounded-depth threshold circuits with small descriptive complexity. In particular, this section shows that the class of languages accepted by Dlogtime-uniform threshold circuits of sub-subexponential size is contained in the class of languages accepted by threshold circuits of sub-subexponential size with descriptors of polylogarithmic size. Section 6.3 gives a version of the diagonalization result Theorem 3.5 for threshold Turing machines and threshold circuits with small descriptors. In section 6.4 this result in conjunction with Allender's arguments from [4] are then subsequently used to extend Allender's result by proving a lower bound for the Permanent on threshold circuits with subpolynomial descriptors.

6.1 Preliminaries

In this chapter, extensive reference is made to the so-called *counting hierarchy*. Recall that the class PP consists of all sets A such that there exists a polynomial-time nondeterministic Turing machine M such that x is in A if and only if the number of accepting paths of M on input x exceeds the number of rejecting rejecting paths. The counting hierarchy CH is then defined [65] as the oracle hierarchy with base PP. That is,

$$\mathsf{CH} = \mathsf{PP} \cup \mathsf{PP}^{\mathsf{PP}} \cup \mathsf{PP}^{\mathsf{PP}^{\mathsf{PP}}} \cup \cdots$$

Since NP and coNP are both contained in PP [29], it follows that the standard polynomial hierarchy is contained in the counting hierarchy. Classes related to PP that are of interest here are $\mathsf{C_{=}P}$ and the function class $\#\mathsf{P}$. Recall that a function f is in $\#\mathsf{P}$ if there exists a polynomial-time nondeterministic Turing machine M such that the number of accepting paths for M on input x equals $f(x)$. Note that $\mathsf{P}^{\mathsf{PP}} = \mathsf{P}^{\#\mathsf{P}}$. The class $\mathsf{C_{=}P}$ consists of all languages A such that there exists a polynomial-time nondeterministic Turing machine M such that x is in A if and only if M on input x has the same number of accepting and rejecting paths. Note that $\mathsf{coNP} \subseteq \mathsf{C_{=}P}$ and $\mathsf{PP} \subseteq \mathsf{NP}^{\mathsf{C_{=}P}}$ [61], and thus, the oracle hierarchy with base $\mathsf{C_{=}P}$ is equal to the counting hierarchy. It is not hard to show that $\mathsf{C_{=}P} \subseteq \mathsf{PP}$. Hence, lower bounds for $\mathsf{C_{=}P}$ are lower bounds for PP. Consequently, lower bounds for $\mathsf{C_{=}P}$ are (potentially) stronger than lower bounds for PP.

Recall that the Permanent of an $n \times n$ matrix $A = (a_{ij})$ is equal to,

$$\sum_{\sigma \in S_n} a_{1\sigma(1)} \cdots a_{n\sigma(n)}.$$

Valiant [62] showed that computing the Permanent of a matrix is hard for $\#\mathsf{P}$. A decision version of the Permanent is hard for $\mathsf{C_{=}P}$ under very weak reductions. In particular, Zankó [71] showed that the language BITPERM= $\{(x, i) :$ the ith bit of the Permanent of (x) is $1\}$ is hard for $\mathsf{C_{=}P}$ under AC^0 reductions. This fact will be used to obtain a lower bound for BITPERM and thus for computing the Permanent.

A variant of the standard alternating Turing machine [16] that has "majority" configurations instead of the standard existential and universal configurations can be used to characterize the counting hierarchy. Such machines are called *threshold Turing machines (TTM)* and were first defined by Parberry and Schnitger in [50]. The exact model that is used in this chapter differs in some details to the machines described in [50]. The formal definition adopted here is based on the modified alternating Turing machines with MODm configurations defined by Allender and Gore [5]. Each configuration of a TTM has at most two successor configurations. Moreover, instead of existential and universal states, threshold Turing machines have "majority" configurations. A TTM is always provided with the length of its input. Note that unlike the machines used in the rest of this thesis, these machines do not have random-access to their non-input tapes. Nevertheless, these machines do have random-access to their input in a restricted sense: each computation path has access to exactly one of the input bits when in a halting configuration; input bits cannot be read when not in a halting configuration.

A unique computation tree $C_{M,x}$ is associated with each input x to a TTM M. The nodes in $C_{M,x}$ are configurations of M on input x. The root of $C_{M,x}$ is the initial configuration of M on input x. A configuration τ in the tree is a child of another configuration σ if τ can directly follow from σ according to the transition function of M. The machine M accepts x if the computation tree $C_{M,x}$ for M on input x is accepting. It remains to define what it means for a computation tree to be accepting. Some more definitions further describing the computation of a TTM are needed before this can be done.

A threshold Turing machine M has three types of configurations: *majority* configurations, *halting* configurations and *auxiliary* configurations. Configurations in the computation tree $C_{M,x}$ of M on an input x are grouped into *majority trees*. A majority tree T_σ with root a configuration σ is one of two things, depending on whether σ is a majority or halting configuration (the root of a majority tree cannot be an auxiliary configuration). If σ is a halting configuration, then T_σ is simply σ. If instead σ is a majority configuration, then T_σ is the tree of configurations in $C_{M,x}$ consisting of σ and all descendants τ of σ such that τ is an auxiliary configuration and on the path between σ and τ in $C_{M,x}$ only auxiliary configurations occur. Note then that the leaves of T_σ, when σ is a majority configuration, have as children other majority trees. A majority tree T_σ is *accepting* if (a) σ is an accepting halting configuration, or (b) σ is a majority configuration and a majority of the majority trees attached to the leaves of T_σ are accepting; otherwise the majority tree is *rejecting*. The initial configuration in the computation tree of M on x is either a halting or majority computation. The computation tree $C_{M,x}$ for M on x is then *accepting* if and only if the majority tree associated with the initial configuration for M on input x is accepting.

Observe that existential and universal branching can be simulated by majority branching. Resources used by a TTM, such as time and space, are measured in the same way as with standard alternating machines [16].

Note that unlike for universal and existential configurations, it is *not* known whether successive majority configurations can be combined. This fact motivates the following definition of "alternation" for threshold Turing machines.

Definition 6.1 *There is an* alternation *between two configurations σ_1 and σ_2 of a TTM if and only if σ_2 can follow from σ_1 in one step, σ_1 is a leaf of a majority-tree and σ_2 is the root of a majority tree.*

Since the counting hierarchy is the oracle hierarchy with base PP, the following relation holds between the counting hierarchy and threshold Turing machines computing in polynomial time:

Theorem 6.1 *A set L is in the counting hierarchy if and only if there exists a TTM M accepting L such that M makes a constant number of alternations and runs in polynomial time.*

This characterization of the counting hierarchy will be used extensively in this chapter.

6.2 Descriptors for unbounded fan-in circuits

In this section the concept of circuit descriptors introduced in section 2.2 is extended to unbounded fan-in bounded-depth circuits. The emphasis is primarily on circuits with only majority gates even though the results can be generalized to circuits containing other types of gates. Relations between Dlogtime-uniform unbounded fan-in bounded-depth threshold circuits and unbounded fan-in bounded-depth threshold circuits with descriptors are explored.

Before defining circuit descriptors for unbounded fan-in circuits, the notion of uniformity for bounded-depth unbounded fan-in circuits is briefly discussed. Recall that a circuit family is "uniform" if it is "sufficiently easy" to construct the nth circuit of the family given 1^n. The circuits in Allender's result [4] are *Dlogtime-uniform*. Essentially, a circuit family is Dlogtime-uniform if its *direct connection language* can be easily decided. Recall that the direct connection language for a family $\{C_n\}$ of circuits consists of all triples $\langle n, g, h\rangle$ such that the output of gate g is an input to gate h in C_n. Note that $\langle n, g, h\rangle$ has logarithmic length with respect to the number of gates in C_n.

Definition 6.2 *A circuit family is* Dlogtime-uniform *if its direct connection language is in* $\mathsf{DTIME}(n)$. *A circuit family is* Dpolylogtime-uniform *if its direct connection language is in* P.

Observe that an encoding $\mu(C)$ of an unbounded fan-in circuit C must have a means for specifying the parents of each gate in C. Also note that if a circuit has n gates, then each gate has at most $n-1$ parents. With these observations in mind, consider the following encoding system w for unbounded fan-in circuits. A threshold circuit C with n gates is represented by a string $w(C)$ of n equal-length duples. Each duple (a, b) corresponds to a unique gate A in C such that the first element a of the duple is a constant length field describing the type of gate A is (i.e., MAJORITY, input, negated input, or constant) and the second element b of the duple is a string of n bits such that the ith bit of b is 1 if and only if the gate corresponding to the ith duple in $w(C)$ is a parent of A. The first duple in $w(C)$ always corresponds to the output gate of C. Observe that $|w(C)| \in O(n^2)$. The motivation behind the design of w is to make it sufficiently "easy" to locate in the duples corresponding to the parents of a gate A provided that the duple corresponding to A is known.

As in the fan-in 2 case, an unbounded fan-in circuit C has *descriptive size* d relative an encoding μ if there exists a *descriptor* circuit C' with d gates and $\log|\mu(C)|$ input bits such that C' on input m (in binary) outputs the mth bit of $\mu(C)$. Note that all references in the sequel to descriptive size for unbounded fan-in circuits will be with respect to the encoding system w described above.

The properties enjoyed by the encoding system w result in the following relation between uniform unbounded fan-in bounded-depth threshold circuits and unbounded fan-in bounded-depth threshold circuits of small descriptive size.

Definition 6.3 *A function t is* sub-subexponential *if for all k,*

$$t(t(n)^k) = 2^{o(1)}.$$

Theorem 6.2 *Let $T = 2^t$ be a constructible sub-subexponential function and let $\{C_n\}$ be a family of depth $d(n)$ threshold circuits of size $\leq T(n)$ whose direct connection language is in* P. *Then there exists a family $\{D_n\}$ of depth $d(n)$ threshold circuits of sub-subexponential size with descriptors of size polynomial in $t(n)$ recognizing the same language as $\{C_n\}$.*

Proof. Let L be the language recognized by $\{C_n\}$. By the techniques in [5] it can be shown that there exists a threshold Turing machine N accepting L such that N runs in time $t(n)^k$, for some integer k, and moreover, the following are true (Allender and Gore prove in [5] an analogous result for bounded-depth ACC^0 circuits and alternating Turing machines with MODm configurations):

1. Every path in every computation tree of N on every input has $d(n) - 1$ alternations.

2. Given a configuration σ of N on an input of length n, the number of primary descendants of σ is computable from σ in time $t(n)^{O(1)}$.

3. Given a configuration σ of N and a number i not more than the number of primary descendants of σ, the ith primary descendent of σ can be computed in time $t(n)^{O(1)}$ from the encoding of σ.

Fix an input length n. Since threshold Turing machines only have access to their input during a halting configuration, the computation trees for N on all inputs of length n are essentially identical in the sense that all computation trees have the same topology and, disregarding the input tape, all corresponding configurations in the trees, with the exception of halting configurations, are identical. So denote by T_N^n the unique computation tree of N on inputs of length n. A circuit D_n will be constructed from T_N^n as follows. Fix a binary encoding E for the configurations in T_N^n. Since N is $O(t(n)^k)$ time-bounded, assume without loss of generality that $|E(\sigma)| = O(t(n)^k)$ for all σ in T_N^n. To each configuration σ in T_N^n such that σ is the root of a majority tree, there corresponds in D_n a threshold gate labeled $E(\sigma)$. To each halting configuration σ in T_N^n that accesses the ith bit of the input there corresponds an input gate in D_n reading the ith bit of the input. The initial configuration of T_N^n corresponds to the output gate of D_n. These are precisely the gates in D_n. The inputs to a gate $E(\sigma)$ in D_n are all gates $E(\gamma)$ such that

1. γ is a descendent of σ in T_N^n;

2. there is at most one alternation between σ and γ; and

3. γ is a root of a majority tree or is a halting configuration.

Finally, specify that the output gate in D_n (which corresponds to the initial configuration in T_N^n) is labeled 0 instead of $E(\alpha)$, where α is the root of T_N^n. In the encoding $w(D_n)$ of D_n, the ith duple corresponds to the gate labeled i by the above scheme. Note that this means that there are duples corresponding to "phantom" gates in D_n.

It is clear that D_n accepts $L \cap \{0,1\}^n$. Moreover, D_n contains $2^{t(n)^k}$ gates. Since $T = 2^t$ is sub-subexponential, it follows that $2^{t(n)^k}$ is also sub-subexponential. Also, by definition of D_n and since every path in every alternation tree of N on every input has $d(n) - 1$ alternations, it follows that the family $\{D_n\}$ has depth $d(n)$. So to prove the theorem, it remains only to show that there exists ℓ such that for all n the bits of $w(D_n)$ can be computed by a circuit of size $t(n)^\ell$.

Consider the following circuit C' for computing the bits of $w(D_n)$. On input i, C' computes the duple (say the jth duple) in which the ith bit of $w(D_n)$ belongs. Since the number m of duples in $w(D_n)$ can be expressed by a binary number of length $t(n)^k$, j can be computed in time $t(n)^k$ and hence, by a circuit (which has m hard-coded) of size $t(n)^{O(1)}$. Then C' proceeds according to whether the ith bit of $w(D_n)$ lies in the a or b fields of the jth duple. Recall that j is an encoding of a configuration in T_N^n. Moreover, by the properties enjoyed by N, the appropriate information for fields a and b can be computed from j in $t(n)^{O(1)}$ time and hence, by a circuit of size $t(n)^{O(1)}$.

The case where j corresponds to a "phantom gate" remains to be discussed. Without loss of generality, assume that if there is an alternation between a configuration σ of a threshold Turing machine and its immediate predecessor, then the state of the machine when in configuration σ will encode the fact that an alternation took place between σ and its immediate predecessor. It is easy to see that with this extra assumption on TTM configurations, detecting if j is a "phantom gate" can be done in $t(n)^{O(1)}$ time. ∎

6.3 Diagonalization

The following diagonalization separates the counting hierarchy from those sets accepted by unbounded fan-in bounded-depth threshold circuits of linear descriptive size. Note the relationship with Theorem 2.4 from Chapter 2.

Theorem 6.3 *Fix an integer $k \geq 0$ and let $T(n)$, $n \leq T(n) \leq 2^n$, be constructible. Then there exists a set L in the counting hierarchy such that for all n, $L \cap \Sigma^n$ cannot be computed by a depth-k threshold circuit of size $\leq T(n)$ and descriptive size n.*

Proof. Let $L = \{x : x$ is a description of a depth-k threshold circuit C of size $\leq T(n)$ with n inputs which rejects x, where $|x| = n\}$. It is clear that $L \cap \Sigma^n$ cannot be computed by a depth-k threshold circuit of size $\leq T(n)$ and descriptive size n. So it suffices to show that L can be computed by a polynomial time bounded Threshold Turing Machine M making a constant number of alternations.

Suppose M that computes using algorithm 6.1 on input x of length n, where n is written on a work tape, and the existential and universal branchings are simulated by threshold branching.

1. Guess the input x' and check that x' encodes a circuit C' with $O(\log T(n))$ inputs. Let r be the number of inputs in C' and let C be the circuit described by C'.

2. Call **Eval**(0). That is, evaluate the gate in C corresponding to the first duple in $w(C)$. Recall that this duple corresponds to the output gate of C.

Eval(i)

1. Let (a, b) be the duple in $w(C)$ starting at bit i, let A denote the gate corresponding to the duple, and let m be the length of the first field of each duple.

2. Evaluate C' on consecutive inputs starting at i to determine a.

3. If A is a MAJORITY gate then switch to a MAJORITY state and branch out into a tree with $T(n)$ leaves where each leaf corresponds to a gate in C. At the jth leaf, if the jth bit of b is 0, then branch once, accepting on one path and rejecting on the other. Otherwise, call **Eval**$(j(m + 2^r))$ (i.e., call **Eval** with the index of the jth duple).

4. If A is a constant gate then universally branch into a tree with n leaves. Accept at the ith leaf if and only if A is the constant gate 0 and the ith bit of x' (the guessed input) equals the ith bit of the actual input.

5. If A is an input gate then universally branch into a tree with n leaves. Suppose that A is the jth input. Accept at the ith leaf, $i \neq j$ if and only if the ith bit of x' (the guessed input) equals the ith bit of the actual input. Accept at the jth leaf if and only if jth bit of x' (the guessed input) equals the jth bit of the actual input and the jth input bit is 0.

Algorithm 6.1: Computing L

Clearly M accepts L and M makes $O(k)$ alternations. Moreover, M only accesses its input at halting configurations. Finally, noting that C' is simulated on each branch of computation $O(\log T(n)) \subseteq O(n)$ times and that each simulation of C' can be done in time $O(n \log n)$, it follows that M runs in polynomial time. But then L is in the counting hierarchy. ∎

6.4 The lower bound

The following non-uniform extension of Allender's lower bound for computing the Permanent of a matrix on bounded-depth threshold circuits is proved in this section.

Theorem 6.4 *Let t be a constructible function such that for all k, $t^{(k)}(n) = o(2^n)$. Then the Permanent cannot be computed by bounded-depth threshold circuits of size $t(n)$ and subpolynomial descriptive size.*

As in Allender's result, the proof of Theorem 6.4 takes advantage of the fact that the "decision version" BITPERM of the Permanent is hard for $\mathsf{C_=P}$ under AC^0 reductions [71]. In particular, in order to prove Theorem 6.4 it suffices to show the more general result that sets hard for $\mathsf{C_=P}$ cannot be computed by bounded-depth threshold circuits of size $t(n)$ and subpolynomial descriptive size, where t is as defined above.

The following notation will be useful in the sequel. Let $\mathsf{TC}^0_{s(n)}(t(n))$ denote the class of sets accepted by bounded-depth unbounded fan-in threshold circuits of size $t(n)$ with descriptors of size $s(n)$, and let $\mathsf{TC}^0_{n^{o(1)}}(t(n))$ denote the class of sets accepted by bounded-depth unbounded fan-in threshold circuits of size $t(n)$ with descriptors of subpolynomial size. Recall from Chapter 2 that $\mathsf{TC}^0_{s(n)} = \mathsf{TC}^0_{s(n)}(n^{O(1)})$ and $\mathsf{TC}^0_{n^{o(1)}} = \mathsf{TC}^0_{n^{o(1)}}(n^{O(1)})$.

Theorem 6.4 is proved with the aid of the following two lemmas whose statements and proofs are based on similar results in [4].

Lemma 6.1 *Let t be a time-constructible function such that for all ℓ, $t^{(\ell)}(n) = o(2^n)$. Suppose that BITPERM is in $\mathsf{TC}^0_{n^{o(1)}}(t(n))$. Then for each set B in the counting hierarchy, there exists k such that B has bounded-depth threshold circuits of size $t^{(k)}(n)$ with descriptors of subpolynomial size.*

Proof. Define the notation CH_i inductively as follows. Let $\mathsf{CH}_1 = \mathsf{C_=P}$ and for, $i > 1$ let $\mathsf{CH}_i = \mathsf{C_=P}^{\mathsf{CH}_{i-1}}$. Note that the counting hierarchy is equal to the union of all the CH_i.

Now suppose that BITPERM is in $\mathsf{TC}^0_{n^{o(1)}}(t(n))$. It suffices to prove that for each i all sets in CH_i can be computed by bounded-depth unbounded fan-in threshold circuits of size $t^{(4i)}(n)$ and subpolynomial descriptive size. The proof is by induction on i.

In the basis, $\mathsf{CH}_1 = \mathsf{C_=P}$ has threshold circuits of size $t(n^{O(1)})n^{O(1)} = O(t(t(t(n))))$ and subpolynomial descriptive size on account of the fact that BITPERM is hard for $\mathsf{C_=P}$ under *uniform* many-one AC^0 reductions. Indeed the circuits for a set in $\mathsf{C_=P}$ can be constructed using the subpolynomial descriptive size circuits for BITPERM and the associated polynomial-size uniform AC_0 reduction circuits.

So assume that CH_i has threshold circuits of size $O(t^{(4i)}(n))$ and subpolynomial descriptive size and consider a set $A \in \mathsf{CH}_{i+1}$. There exists a nondeterministic machine M and a set $D \in \mathsf{CH}_i$ such that M^D on input x has exactly as many accepting paths as rejecting paths if and only if $x \in A$. Now consider the set $\{(x, C) : M$ has exactly as many accepting paths as rejecting paths on input x when all oracle queries are answered according to the circuit $C\}$. Then this set is in $\mathsf{C_=P}$ and hence, by the basis has bounded-depth unbounded fan-in threshold circuits of size $O(t^{(3)}(n))$ and subpolynomial descriptive size. Replacing C by the appropriate circuits for D guaranteed by the induction hypothesis results in a bounded-depth unbounded fan-in threshold circuit family of size bounded by

$$t^{(3)}(n + t^{(4i)}(n)) \leq t^{(4(i+1))}(n)$$

accepting A. Moreover, it is clear that if the circuits for D have descriptors of subpolynomial size, then so do the circuits for A. ∎

Note that there exist sub-subexponential functions majorizing all functions t for which $t^{(k)}(n) = o(2^n)$ for all k. So in order to complete the proof of Theorem 6.4 it suffices to show the following:

Lemma 6.2 *Let t be a constructible sub-subexponential function. Then there exists a set A in the counting hierarchy requiring size greater than $t(n)$ to compute on bounded-depth unbounded fan-in threshold circuits with subpolynomial descriptive size.*

Proof. Consider the Circuit Value problem, the problem of deciding whether a given circuit evaluates to true on a given input. Recall that Circuit Value is in P and hence in CH. Note that any problem in $\mathsf{TC}^0_{n^{o(1)}}$ can be reduced to Circuit Value via a polynomial size projection of subpolynomial descriptive size. Now if Circuit Value cannot be done by bounded-depth threshold circuits of size $t(n)$ and subpolynomial descriptive size, then the theorem follows. So assume otherwise. Note then that there exists an integer k such that any problem in $\mathsf{TC}^0_{n^{o(1)}}$ can be computed by depth-k threshold circuits of size $t(n^{O(1)})$ and subpolynomial descriptive size.

The next step is to show that for all integers m any language that can be computed by depth-m threshold circuits of size $t(n)$ and subpolynomial descriptive size can be computed by depth-k threshold circuits of size $t(t(n)^{O(1)})$ and subpolynomial descriptive size. To that end let A be a language computed by depth-m threshold circuits of size $t(n)$ and subpolynomial descriptive size. Consider $A' = \{x10^j : j + |x| + 1 = t(|x|)$ and $x \in A\}$. The time-constructibility assumptions on t imply that A' is in $\mathsf{TC}^0_{n^{o(1)}}$, and hence, computable by a family $\{C_n\}$ of depth-k threshold circuits of size $t(n^\ell)$ (for some ℓ) and subpolynomial descriptive size. Now consider the following circuit family $\{D_n\}$. Circuit D_n consists of $C_{t(n)}$ where the first n input bits for $C_{t(n)}$ are set to the input for D_n, the $n+1$st input bit of $C_{t(n)}$ is hard-wired to 1, and the remaining input bits for $C_{t(n)}$ are hard-wired to 0. Then $\{D_n\}$ is a family of depth-k threshold circuits of size $t(t(n)^\ell)$ accepting A. Moreover, it is easy to see that the circuits $\{D_n\}$ have subpolynomial descriptive size.

Since t is sub-subexponential, it follows by Theorem 6.3 that there is a set B in the counting hierarchy that cannot be computed by depth-k threshold circuits of size $t(t(n)^{O(1)})$ and subpolynomial descriptive size. But then, by the above it follows that B cannot be computed by bounded-depth threshold circuits of size $t(n)$ and subpolynomial descriptive size. ∎

Theorem 6.4 now follows. Indeed, suppose to the contrary that the Permanent has bounded-depth unbounded fan-in threshold circuits of size $t(n)$ and subpolynomial descriptive size where $t^{(k)}(n) \in o(2^n)$ for all $k \in \mathbb{N}$. It follows that BitPerm has bounded-depth unbounded fan-in threshold circuits of size

$t(n)$ and subpolynomial descriptive size. But then, Lemma 6.1 implies that each set in the counting hierarchy has bounded-depth unbounded fan-in threshold circuits of $t^{(O(1))}(n)$ size and subpolynomial descriptive size, contradicting Lemma 6.2.

Corollary 6.5 *Let t be a constructible function such that for all k, $t^{(k)}(n) = o(2^n)$. Then the Permanent cannot be computed by bounded-depth Dpolylogtime-uniform threshold circuits of size $t(n)$.*

Proof. Follows from Theorems 6.2 and 6.4. ■

In fact, Lemma 6.2 and the proof of Lemma 6.1 imply the following more general result.

Theorem 6.6 *Let A be hard for $\mathsf{C_{=}P}$ under $\leq_{\mathrm{T}}^{\mathsf{TC}^0}$ reducibility, and let t be a time-constructible function such that for all k, $t^{(k)}(n) \in o(2^n)$. Then $A \notin \mathsf{TC}^0_{n^{o(1)}}(t(n))$.*

Note that a language B is hard for a class $\mathcal{C}$ under $\leq_{\mathrm{T}}^{\mathsf{TC}^0}$ reducibility if for all languages A in $\mathcal{C}$ there exists a TC^0 family of circuits with oracle gates for B accepting A. This type of reducibility is discussed in Allender [4]. Note that in [4] Allender proved a general lower bound showing that if for all k, $t^{(k)}(n) \in o(2^n)$, then languages hard for $\mathsf{C_{=}P}$ under $\leq_{\mathrm{T}}^{\mathsf{TC}^0}$ reducibility are not in $\mathsf{TC}^0(t(n))$. Since Theorem 6.2 implies that $\mathsf{TC}^0_{n^{o(1)}}(t(n))$ contains Dlogtime-uniform $\mathsf{TC}^0(t(n))$ for t such as in the above theorem, the above theorem generalizes Allender's general result.

6.5 Discussion

It is expected that SAT requires exponential size unbounded depth circuits. Consequently, since BIT-PERM is hard for NP, it follows that the Permanent should also require exponential size unbounded depth circuits. Thus the bounds presented in Allender [4] and in this chapter are likely far from optimal. Nevertheless, proving an exponential lower bound for the Permanent even on non-uniform ACC^0 circuits seems quite difficult.

As an intermediate goal, it would be interesting to determine whether the lower bound for the Permanent presented in this Chapter can be improved to circuits with sublinear rather than subpolynomial descriptive size using similar techniques.

Chapter 7

Future directions

This thesis studied models and limits of non-uniform computation. In Chapter 2 a new way of quantifying the descriptional complexity of circuits was introduced and analyzed. In addition, that chapter introduced and studied a non-uniform variant of the Turing machine. The relations of non-uniform classes defined using this new computing model to non-uniform classes defined using Karp-Lipton advice were explored. Subsequently, the relative merits of using each system to define non-uniform classes were discussed. It would be of interest to establish tighter relationships between these competing definitions of non-uniform computation models and classes. Moreover, better understanding the structure of classes defined by non-uniform Turing machines is also of interest.

Some of these questions relate to questions raised by the work in Chapter 3. In Chapter 3, the well-known time and space hierarchies were generalized to non-uniform classes. In addition, a general argument for diagonalizing over machines taking n bits of advice on inputs of length n was introduced and applied in later chapters to get non-uniform lower bounds for Satisfiability and the Permanent. In [26] Fortnow argued that simple diagonalization may yet be able to provide some new and interesting separations. By demonstrating that simple diagonalization can be used to obtain tight separations of partially non-uniform and uniform classes, the results in this thesis would seem to lend more ammunition to Fortnow's argument.

An extension of the work in Chapter 3 that does not seem too difficult would be to show the existence of a language in $\mathsf{NTIME}(t(n)) - \mathsf{NTIME}(o(t(n-1))^{(n)}$ for sufficiently "nice" functions t. This seems achievable using the present techniques. On the other hand, increasing the amount of non-uniformity in Theorems 3.5 or 3.7, and thereby possibly increasing the amount of non-uniformity in all the non-uniform results presented in this thesis, seems quite difficult to do without incurring an exponential time gap as in Theorem 3.9. Indeed, increasing the amount of uniformity to $n^{1+o(1)}$ in Theorem 3.7 would imply the existence of a language in P without linear (completely non-uniform) circuits. Another ambitious, and probably very difficult, goal would be to determine for what choices of functions t and T, $t \in o(T)$, the classes $\mathsf{DTIME}(T(n))^{(poly)}$ and $\mathsf{DTIME}(t(n))^{(poly)}$ differ.

In Chapters 4 and 5 the first non-trivial time-space lower bounds for Satisfiability on non-uniform machines were given. For instance, it was shown that SAT is not computable by a random-access multitape deterministic machine using $n^{1.414}$ time and subpolynomial space and advice, nor by a random-access multi-tape deterministic machine using $n^{1.186}$ time, $\sqrt{n}$ space and subpolynomial advice. New separations of uniform classes were also obtained. For instance, it was shown that for all $\epsilon > 0$ and all rational numbers $r \geq 1$, $\mathsf{DTISP}(n^r, n^{1-\epsilon}) \subsetneq \mathsf{NTIME}(n^r)$.

To put the lower bounds presented for SAT in this thesis in perspective, note that SAT is expected to require exponential deterministic time and exponential size circuits. Indeed, Fortnow and van Melkebeek [27] improve on the uniform lower bounds for SAT presented in this thesis. However, substantially improving the lower bounds presented in Chapters 4 and 5 seems quite difficult. Nevertheless, this is a worthwhile pursuit since strengthening, for instance, the uniform lower bounds presented in this thesis would resolve some long-standing open questions:

Theorem 7.1 *Suppose there exists an unbounded function $s(n)$ such that,*

$$\mathsf{NTIME}(n^k) \not\subseteq \mathsf{DTISP}(n^{ks(k)}, \log^{O(1)} n),$$

for infinitely many $k \in \mathbb{N}$. Then $\mathsf{SC} \neq \mathsf{NP}$.

Proof. Suppose to the contrary that $\mathsf{SC} = \mathsf{NP}$. Then SAT $\in \mathsf{DTISP}(n^\ell, \log^{O(1)} n)$ for some integer ℓ. Lemma 1.3 then implies that $\mathsf{NTIME}(n^j) \subseteq \mathsf{DTISP}(n^{j\ell} \log^{O(1)} n, \log^{O(1)} n)$ for all $j > \ell$. However, this leads to a contradiction since the hypotheses imply the existence of a j such that $j > \ell$, $s(j) > \ell$ and $\mathsf{NTIME}(n^j) \not\subseteq \mathsf{DTISP}(n^{js(j)}, \log^{O(1)} n)$. ■

Further strengthening the hypotheses in Theorem 7.1 yields an even more impressive separation:

Theorem 7.2 *Suppose there exists an unbounded function $s(n)$ such that,*

$$\mathsf{NTIME}(n^k) \not\subseteq \mathsf{DTIME}(n^{ks(k)})$$

for infinitely many $k \in \mathbb{N}$. Then $\mathsf{P} \neq \mathsf{NP}$.

However, it seems that substantially stronger techniques are needed to even get the separations required for the hypotheses of Theorem 7.1. On the other hand, weaker improvements, like those given in [27], will probably be significantly easier to achieve. It also seems likely that strengthening the lower bounds presented in Chapters 4 and 5 to classes with sublinear rather than subpolynomial advice will not be too difficult.

In Chapter 6 recent work by Allender [4] showing that the Permanent requires large uniform TC^0 circuits was extended to TC^0 circuits with subpolynomial descriptive size. As noted in the discussion section at the end of Chapter 6, it is likely that the Permanent requires exponential size completely non-uniform TC^0 circuits. However, progress in this direction seems quite difficult within the proof framework presented in this thesis.

Bibliography

[1] M. Ajtai. Σ_1^1-formulae on finite structures. *Annals of Pure and Applied Logic*, 24:1–48, 1983.

[2] M. Ajtai. Determinism versus non-determinism for linear time RAMs with memory. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, pages 632–641, Atlanta, Georgia, May 1–4 1999. ACM Press.

[3] M. Ajtai. A non-linear time lower bound for boolean branching programs. In *40th Annual Symposium on Foundations of Computer Science* [39], pages 459–464.

[4] E. Allender. The permanent requires large uniform threshold circuits. *Chicago Journal of Theoretical Computer Science*, 1999(7), Aug. 1999.

[5] E. Allender and V. Gore. A uniform circuit lower bound for the permanent. *SIAM Journal on Computing*, 23(5):1026–1049, Oct. 1994.

[6] J. L. Balcázar, J. Díaz, and J. Gabarró. On characterizations of the class PSPACE/ poly. *Theoretical Computer Science*, 52(3):251–267, 1987.

[7] J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I.* Springer-Verlag, Berlin, 1995.

[8] J. L. Balcázar and M. Hermo. The structure of logarithmic advice complexity classes. *Theoretical Computer Science*, 207(1):217–244, 28 Oct. 1998.

[9] J. L. Balcázar, M. Hermo, and E. Mayordomo. Characterizations of logarithmic advice complexity classes. In J. van Leeuwen, editor, *Proceedings of the IFIP 12th World Computer Congress. Volume 1: Algorithms, Software, Architecture*, pages 315–321, Amsterdam, The Netherlands, Sept. 1992. Elsevier Science Publishers.

[10] J. L. Balcázar and U. Schöning. Logarithmic advice classes. *Theoretical Computer Science*, 99(2):279–290, June 1992.

[11] G. Barnes and J. A. Edmonds. Time-space lower bounds for directed *st*-connectivity on graph automota models. *SIAM Journal on Computing*, 27(4):1190–1202, Aug. 1998.

[12] P. Beame. A general sequential time-space trade-off for finding unique elements. *SIAM Journal on Computing*, 20(2):270–277, Apr. 1991.

[13] P. Beame, M. Saks, and J. S. Thathachar. Time-space trade-offs for branching programs. In *FOCS: IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 254–263, 1998.

[14] R. V. Book, S. A. Greibach, and B. Wegbreit. Time- and tape-bounded Turing acceptors and AFLs. *Journal of Computer and System Sciences*, 4(6):606–621, 1970.

[15] H. Caussinus, P. McKenzie, D. Thérien, and H. Vollmer. Nondeterministic NC^1 computation. In *Proceedings, Eleventh Annual IEEE Conference on Computational Complexity*, pages 12–21, Philadelphia, Pennsylvania, 24–27 May 1996. IEEE Computer Society Press.

[16] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114–133, 1981.

[17] A. Cobham. The recognition problem for the set of perfect squares. In *Conference Record of the Seventh Annual Symposium on Switching and Automata Theory*, pages 78–87, IEEE, New York, 1966.

[18] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd ACM Symposium on the Theory of Computating*, pages 151–158, ACM, New York, 1971.

[19] S. A. Cook. A hierarchy for nondeterministic time complexity. *Journal of Computer and System Sciences*, 7(4):343–353, Aug. 1973.

[20] S. A. Cook. An observation of time-storage trade off. In *ACM Symposium on Theory of Computing (STOC '73)*, pages 29–33, New York, Apr. 1973. ACM Press.

[21] S. A. Cook. Deterministic CFL's are accepted simultaneously in polynomial time and log squared space. In *Conference Record of the Eleventh Annual ACM Symposium on Theory of Computing*, pages 338–345, Atlanta, Georgia, 30 Apr.–2 May 1979.

[22] S. A. Cook. Short propositional formulas represent nondeterministic computations. *Information Processing Letters*, 26(5):269–270, Jan. 1988.

[23] P. Dúriś and Z. Galil. A time-space tradeoff for language recognition. *Mathematical Systems Theory*, 17(1):3–12, Mar. 1984.

[24] J. Edmonds and C. K. Poon. A nearly optimal time-space lower bound for directed *st*-connectivity on the NNJAG model. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on the Theory of Computing*, pages 147–156, Las Vegas, Nevada, 29 May–1 June 1995.

[25] J. A. Edmonds. Time-space tradeoffs for directed *st*-connectivity on the NNJAG model. *SIAM Journal on Computing*, 27(5):1492–1513, Oct. 1998.

[26] L. Fortnow. Nondeterministic polynomial time versus nondeterministic logarithmic space: Time-space tradeoffs for satisfiability. In *Proceedings, Twelfth Annual IEEE Conference on Computational Complexity*, pages 52–60, Ulm, Germany, 24–27 June 1997. IEEE Computer Society Press.

[27] L. Fortnow and D. van Melkebeek. Time-space tradeoffs for nondeterministic computation. In *Proceedings, Fifteenth Annual IEEE Conference on Computational Complexity*, pages 2–13, Florence, Italy, 4–7 July 2000. IEEE Computer Society Press.

[28] M. Furst, J. B. Saxe, and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, Apr. 1984.

[29] J. T. Gill. Computational complexity of probabilistic Turing machines. *SIAM Journal on Computing*, 6(4):675–695, Dec. 1977.

[30] S. Gupta. Alternating time versus deterministic time: A separation. *Mathematical Systems Theory*, 29(6):661–672, Nov./Dec. 1996.

[31] Y. Gurevich and S. Shelah. Nondeterministic linear-time tasks may require substantially nonlinear deterministic time in the case of sublinear workspace. *Journal of the Association for Computing Machinery*, 37(3):674–687, July 1990.

[32] J. Håstad. *Computational Limitations for Small Depth Circuits.* MIT Press, Cambridge, MA, 1987.

[33] J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Trans. Amer. Math. Soc.*, 117:285–306, 1965.

[34] F. C. Hennie and R. E. Stearns. Two-tape simulations of multitape Turing machines. *Journal of the Association for Computing Machinery*, 13(4):533–546, Oct. 1966.

[35] M. Hermo. *Nonuniform complexity classes with sub-linear advice functions.* PhD thesis, Universidad del País Vasco, 1996.

[36] M. Hermo and E. Mayordomo. A note on polynomial-size circuits with low resource-bounded kolmogorov complexity. *Mathematical Systems Theory*, 27(4):347–356, July/Aug. 1994.

[37] J. Hopcroft, W. Paul, and L. Valiant. On time versus space. *Journal of the Association for Computing Machinery*, 24(2):332–337, Apr. 1977.

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

[39] IEEE. *40th Annual Symposium on Foundations of Computer Science*, New York City, New York, 17–19 Oct. 1999.

[40] R. Impagliazzo and A. Wigderson. $P = BPP$ if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 220–229, El Paso, Texas, 4–6 May 1997.

[41] V. Kabanets and J.-Y. Cai. Circuit minimization problem. To appear in STOC 2000.

[42] R. Kannan. Circuit-size lower bounds and non-reducibility to sparse sets. *Information and Control*, 55(1–3):40–56, Oct./Nov./Dec. 1982.

[43] R. Kannan. Towards separating nondeterminism from determinism. *Mathematical Systems Theory*, 17(1):29–45, Apr. 1984.

[44] M. Karchmer. Two time-space tradeoffs for element distinctness. *Theoretical Computer Science*, 47(3):237–246, 1986.

[45] R. M. Karp and R. J. Lipton. Some connections between nonuniform and uniform complexity classes. In *Conference Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*, pages 302–309, Los Angeles, California, 28–30 Apr. 1980.

[46] R. J. Lipton and A. Viglas. On the complexity of SAT. In *40th Annual Symposium on Foundations of Computer Science* [39], pages 459–464.

[47] A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *13th Annual Symposium on Switching and Automata Theory*, pages 125–129, The University of Maryland, 25–27 Oct. 1972. IEEE.

[48] V. Nepomnjaščiĭ. Rudimentary predicates and Turing calculations. *Soviet Mathematics-Doklady*, 11(6):1462–1465, 1970.

[49] C. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, Massachusetts, 1994.

[50] I. Parberry and G. Schnitger. Parallel computation with threshold functions. *Journal of Computer and System Sciences*, 36(3):278–302, June 1988.

[51] W. J. Paul, N. Pippenger, E. Szemerédi, and W. T. Trotter. On determinism versus non-determinism and related problems (preliminary version). In *24th Annual Symposium on Foundations of Computer Science*, pages 429–438, Tucson, Arizona, 7–9 Nov. 1983. IEEE.

[52] N. Pippenger and M. J. Fischer. Relations among complexity measures. *Journal of the Association for Computing Machinery*, 26(2):361–381, Apr. 1979.

[53] C. Rackoff. Personal communication, 1999.

[54] A. A. Razborov. Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. *MATHNASUSSR: Mathematical Notes of the Academy of Sciences of the USSR*, 41(4):333–338, 1987.

[55] J. M. Robson. An $O(T \log T)$ reduction from RAM computations to satisfiability. *Theoretical Computer Science*, 82(1):141–149, May 1991.

[56] J. E. Savage. Computational work and time on finite machines. *Journal of the ACM*, 19(4):660–674, Oct. 1972.

[57] U. Schöning. Robust algorithms: a different approach to oracles. *Theoretical Computer Science*, 40(1):57–66, Sept. 1985.

[58] J. I. Seiferas, M. J. Fischer, and A. R. Meyer. Separating nondeterministic time complexity classes. *Journal of the Association for Computing Machinery*, 25(1):146–167, Jan. 1978.

[59] R. Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In A. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 77–82, New York City, NY, May 1987. ACM Press.

[60] L. J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3:1–22, 1977.

[61] J. Torán. Complexity classes defined by counting quantifiers. *Journal of the Association for Computing Machinery*, 38(3):753–774, July 1991.

[62] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, Apr. 1979.

[63] D. van Melkebeek. Personal communication, 2000.

[64] A. Viglas. Personal communication, 1999.

[65] K. W. Wagner. The complexity of combinatorial problems with succinct input representation. *Acta Informatica*, 23:325–356, 1986.

[66] I. Wegener. *The Complexity of Boolean Functions*. B.G. Teubner, Stuttgart, 1 edition, 1987.

[67] A. Yao. Near-optimal time-space tradeoff for element distinctness. *SIAM Journal on Computing*, 23(5):966–975, Oct. 1994.

[68] A. C.-C. Yao. Separating the polynomial-time hierarchy by oracles. In *26th Annual Symposium on Foundations of Computer Science*, pages 1–10, Los Angeles, Ca., USA, Oct. 1985. IEEE Computer Society Press.

[69] C. K. Yap. Some consequences of non-uniform conditions on uniform classes. *Theoretical Computer Science*, 26(3):287–300, Oct. 1983.

[70] S. Žák. A Turing machine time hierarchy. *Theoretical Computer Science*, 26(3):327–333, Oct. 1983.

[71] V. Zankó. #P-completeness via many-one reductions. *International Journal of Foundations of Computer Science*, 2:77–82, 1991.

Index