A Survey of Lower Bounds for Satisfiability and Related Problems

Dieter van Melkebeek*
University of Wisconsin-Madison

September 29, 2007

Abstract

Ever since the fundamental work of Cook from 1971, satisfiability has been recognized as a central problem in computational complexity. It is widely believed to be intractable, and yet till recently even a linear-time, logarithmic-space algorithm for satisfiability was not ruled out. In 1997 Fortnow, building on earlier work by Kannan, ruled out such an algorithm. Since then there has been a significant amount of progress giving non-trivial lower bounds on the computational complexity of satisfiability.

In this article we survey the known lower bounds for the time and space complexity of satisfiability and closely related problems on deterministic, randomized, and quantum models with random access. We discuss the state-of-the-art results and present the underlying arguments in a unified framework.

1 Introduction

Satisfiability is the problem of deciding whether a given Boolean formula has at least one satisfying assignment. It is the first problem that was shown to be NP-complete, and is possibly the most commonly studied NP-complete problem, both for its theoretical properties and its applications in practice. Complexity theorists widely believe that satisfiability takes exponential time in the worst case and requires an exponent linear in the number of variables of the formula. On the other hand, we currently do not know how to rule out the existence of even a linear-time algorithm on a random-access machine. Obviously, linear time is needed since we have to look at the entire formula in the worst case. Similarly, we conjecture that the space complexity of satisfiability is linear but we have yet to establish a space lower bound better than the trivial logarithmic one.

Till the late 1990s it was even conceivable that there could be an algorithm that would take linear time and logarithmic space to decide satisfiability! Fortnow [For00], building on earlier techniques by Kannan [Kan84], developed an elegant argument to rule out such algorithms. Since then a wide body of work [LV99, FvM00, Tou01, AKR+01, vM04, FLvMV05, Wil05, DvM06, Vio07, Wil07b, vMW07, DvMW07] have strengthened and generalized the results to lead to a rich variety of lower bounds when one considers a non-trivial combination of time and space complexity. These results form the topic of this survey.

We now give some details about the evolution of the lower bounds for satisfiability. Fortnow's result is somewhat stronger than what we stated above – it shows that satisfiability cannot have

both a linear-time algorithm and a (possibly different) logarithmic-space algorithm. In fact, his argument works for time bounds that are slightly super-linear and space bounds that are close to linear, and even applies to conondeterministic algorithms.

Theorem 1.1 (Fortnow [For00]) *For every positive real ϵ, satisfiability cannot be solved by both*

(i) *a (conon-)deterministic random-access machine that runs in time $n^{1+o(1)}$ and*

(ii) *a (conon-)deterministic random-access machine that runs in polynomial time and space $n^{1-\epsilon}$.*

In terms of time-space lower bounds, Fortnow's result implies that there is no (conon-)deterministic algorithm solving satisfiability in time $n^{1+o(1)}$ and space $n^{1-\epsilon}$. Lipton and Viglas [LV99, FLvMV05] considered deterministic algorithms with smaller space bounds, namely polylogarithmic ones, and managed to establish the first time-space lower bound where the running time is a polynomial of degree larger than one. Their argument actually works for subpolynomial space bounds, i.e., for space $n^{o(1)}$. It shows that satisfiability does not have a deterministic algorithm that runs in $n^{\sqrt{2}-o(1)}$ steps and uses subpolynomial space.[1] Fortnow and Van Melkebeek [FvM00, FLvMV05] captured and improved both earlier results in one statement, pushing the exponent in the Lipton-Viglas time-space lower bound from $\sqrt{2} \approx 1.414$ to the golden ratio $\phi \approx 1.618$. Williams [Wil05, Wil07b] further improved the latter exponent to the current record of $2\cos(\pi/7) \approx 1.801$, although his argument no longer captures Fortnow's original result for deterministic machines.

Theorem 1.2 (Williams [Wil07b]) *Satisfiability cannot be solved by a deterministic random-access machine that runs in time $n^{2\cos(\pi/7)-o(1)}$ and space $n^{o(1)}$.*

The following – somewhat loaded – statement represents the state-of-the-art on lower bounds for satisfiability on deterministic machines with random access.

Theorem 1.3 (Master Theorem for deterministic algorithms) *For all reals c and d such that $(c-1)d < 1$ or $cd(d-1) - 2d + 1 < 0$, there exists a positive real e such that satisfiability cannot be solved by both*

(i) *a conondeterministic random-access machine that runs in time n^c and*

(ii) *a deterministic random-access machine that runs in time n^d and space n^e.*

Moreover, the constant e approaches 1 from below when c approaches 1 from above and d is fixed.

Note that a machine of type (ii) with $d < 1$ trivially fails to decide satisfiability as it cannot access the entire input. A machine of type (i) with $c < 1$ can access the entire input but nevertheless cannot decide satisfiability. This follows from a simple diagonalization argument which we will review in Section 3 because it forms an ingredient in the proof of Theorem 1.3. Note also that a machine of type (ii) is a special case of a machine of type (i) for $d \leq c$. Thus, the interesting values of c and d in Theorem 1.3 satisfy $d \geq c \geq 1$.

Theorem 1.3 applies when c and d satisfy a disjunction of two conditions. For values of $d > 2$ the condition $(c-1)d < 1$ is less stringent than $cd(d-1) - 2d + 1 < 0$; for $d < 2$ the situation is the other way around. See Figure 1 for a plot of the bounds involved in Theorem 1.3. We can use

---

[1]The exact meaning of this statement reads as follows: For every function $f : \mathbb{N} \to \mathbb{N}$ that is $o(1)$, there exists a function $g : \mathbb{N} \to \mathbb{N}$ that is $o(1)$ such that no algorithm for satisfiability can run in time $n^{\sqrt{2}-g(n)}$ and space $n^{f(n)}$.
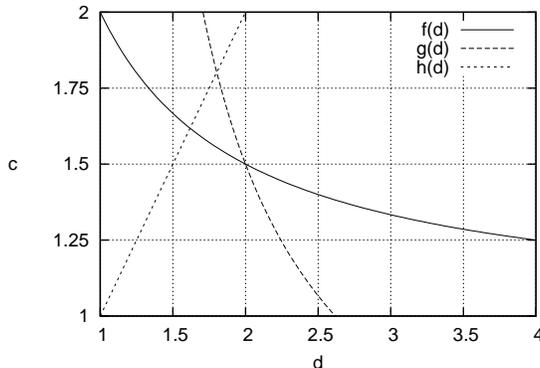
Figure 1: Bounds in the Master Theorem for deterministic algorithms: $f(d)$ solves $(c-1)d = 1$ for c, $g(d)$ solves $cd(d-1) - 2d + 1 = 0$ for c, and $h(d)$ is the identity.

the first condition to rule out larger and larger values of d for values of c that get closer and closer to 1 from above. Thus, Fortnow's result for deterministic machines is a corollary to Theorem 1.3. The second condition does not hold for large values of d for any $c \geq 1$, but yields better time lower bounds for subpolynomial-space algorithms. We can obtain time-space lower bounds from Theorem 1.3 by setting $c = d$; in that case we can omit the part from the statement involving the machine of type (i) as it is implied by the existence of a machine of type (ii). The first condition thus yields a time lower bound of $n^{d-o(1)}$ for subpolynomial space, where $d > 1$ satisfies $d(d-1) = 1$, i.e., for d equal to the golden ratio $\phi \approx 1.618$. The second condition leads to a time lower bound of $n^{d-o(1)}$ for subpolynomial space, where $d > 1$ satisfies $d^2(d-1) - 2d + 1 = 0$; the solution to the latter equation equals the above mysterious constant of $2\cos(\pi/7) \approx 1.801$, which is larger than ϕ. Thus, the Master Theorem captures Theorem 1.2 as well.

The successive improvements of recent years beg the question how far we can hope to push the time-space lower bounds for satisfiability in the near future. On the end of the spectrum with small space bounds, there is a natural bound of 2 on the exponent d for which the current techniques allow us to prove a time lower bound of n^d for algorithms solving satisfiability in logarithmic space. We will discuss this bound in Section 4.1 and its reachability in Section 9. On the end of the spectrum with small time bounds, the quest is for the largest exponent e such that we can establish a space lower bound of n^e for any algorithm solving satisfiability in linear time. The techniques presented in this survey critically rely on sublinear space bounds so we cannot hope to reach $e = 1$ or more along those lines. Note that sublinear-space algorithms for satisfiability are unable to store an assignment to the Boolean formula.

All the known lower bounds for satisfiability on deterministic random-access machines use strategies similar to one pioneered by Kannan in his early investigations of the relationship between nondeterministic and deterministic linear time [Kan84]. The arguments really give lower bounds for nondeterministic linear time; they translate to lower bounds for satisfiability by virtue of the very efficient quasi-linear reductions of nondeterministic computations to satisfiability. The same type of reductions exist to many other NP-complete problems – in fact, to the best of my knowledge, they exist for all of the standard NP-complete problems. Thus, the lower bounds for satisfiability as stated in Theorem 1.3 actually hold for all these problems. In Section 4.2 we discuss how the underlying arguments can be adapted and applied to other problems that are closely related to

3

satisfiability, such the cousins of satisfiability in higher levels of the polynomial-time hierarchy and the problem of counting the number of satisfying assignments to a given Boolean formula modulo a fixed number.

Lower bounds for satisfiability on deterministic machines relate to the P-versus-NP problem. Similarly, in the context of the NP-versus-coNP problem, one can establish lower bounds for satisfiability on conondeterministic machines, or equivalently, for tautologies on nondeterministic machines. The statement of Theorem 1.3 partially realizes such lower bounds because the machine of type (i) is conondeterministic; all that remains is to make the machine of type (ii) conondeterministic, as well. In fact, Fortnow proved his result for conondeterministic machines of type (ii). Similar to the deterministic case, Fortnow and Van Melkebeek [FvM00, FLvMV05] improved the time lower bound in this version of Fortnow's result from slightly super-linear to a polynomial of degree larger than 1. In terms of time-space lower bounds on the large-time end of the spectrum, their result yields a time lower bound of $n^{\sqrt{2}-o(1)}$ for subpolynomial space nondeterministic machines that decide tautologies. Diehl, Van Melkebeek, and Williams [DvMW07] improved the exponent in the latter result from $\sqrt{2} \approx 1.414$ to $\sqrt[3]{4} \approx 1.587$ but their proof does not yield nontrivial results at the end of the spectrum with space bounds close to linear.

Theorem 1.4 (Diehl-Van Melkebeek-Williams [DvMW07]) *Tautologies cannot be solved by a nondeterministic random-access machine that runs in time $n^{\sqrt[3]{4}-o(1)}$ and space $n^{o(1)}$.*

The following counterpart to Theorem 1.3 captures all the known lower bounds for tautologies on nondeterministic machines with random access.

Theorem 1.5 (Master Theorem for nondeterministic algorithms) *For all reals c and d such that $(c^2 - 1)d < c$ or $c^2 d < 4$, there exists a positive real e such that tautologies cannot be solved by both*

(i) a nondeterministic random-access machine that runs in time n^c and

(ii) a nondeterministic random-access machine that runs in time n^d and space n^e.

Moreover, the constant e approaches 1 from below when c approaches 1 from above and d is fixed.

Similar to the deterministic setting, the interesting values in Theorem 1.5 satisfy $d \geq c \geq 1$. The hypothesis is the disjunction of two conditions. See Figure 1 for a plot of the bounds involved. The first condition is binding for larger values of d and allows us to derive Fortnow's result in full form. The second condition is binding for smaller values of d, which includes the range in which the hypothesis holds for $c = d$. The first condition yields a time lower bound of $n^{d-o(1)}$ for subpolynomial space, where $d > 1$ satisfies $d(d^2 - 1) = d$, i.e., for $d = \sqrt{2}$. The second condition leads to such a lower bound for $d > 1$ satisfying $d^3 = 4$, yielding Theorem 1.4.

Theorems 1.3 and 1.5 can be viewed as the first two in a sequence where the machines of type (ii) can have more and more alternations. We will not pursue this sequence any further in full generality but the case of small values for c plays a role in the lower bounds for satisfiability on "somewhat-nonuniform" models, which we discuss next.

Complexity theorists do not think that nonuniformity helps in deciding satisfiability. In particular, we conjecture that satisfiability requires circuits of linear-exponential size. At the same time, we cannot rule out that satisfiability has linear-size circuits.
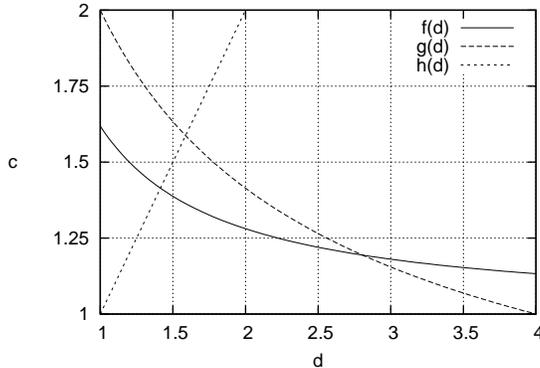
Figure 2: Bounds in the Master Theorem for nondeterministic algorithms: $f(d)$ solves $(c^2-1)d = c$ for c, $g(d)$ solves $c^2 d = 4$ for c, and $h(d)$ is the identity.

Time-space lower bounds for deterministic machines straightforwardly translate into size-width lower bounds for sufficiently uniform circuits, and into depth-logarithm-of-the-size lower bounds for sufficiently uniform branching programs. Lower bounds for (co)nondeterministic machines similarly imply lower bounds for very uniform (co)nondeterministic circuits. Logarithmic-time uniformity trivially suffices for all of the above results to carry over without any changes in the parameters. We currently do not know of any interesting lower bounds for fully nonuniform general circuits. However, modulo some deterioration of the parameters, we can relax or even eliminate the uniformity conditions in some parts of Theorems 1.3 and 1.5. This leads to lower bounds with relatively weak uniformity conditions in a few models of interest.

Fortnow showed how to apply his technique to logspace-uniform NC^1-circuits [For00]. Allender et al. [AKR+01] extended this result to logspace-uniform SAC^1-circuits and their negations. Van Melkebeek [vM04] derived all these circuit results as instantiations of a general theorem, and showed directly that in each case $\text{NTS}(n^{O(1)}, n^{1-\epsilon})$-uniformity for a positive constant ϵ suffices, where $\text{NTS}(t,s)$ refers to nondeterministic computations that run in time t and space s. We can further relax the uniformity condition from nondeterministic to alternating computations of the same type with a constant number of alternations, i.e., to $\Sigma_k\text{TS}(n^{O(1)}, n^{1-\epsilon})$-uniformity for arbitrary constant k. See Section 2.2 for the precise definitions of the complexity classes and uniformity conditions involved.

We discuss "somewhat-nonuniform" versions of Theorems 1.3 and 1.5 in Section 6. Here we suffice with the corresponding statement for alternating machines when c ranges over values close to 1, since this setting allows us to capture all the above results.

Theorem 1.6 (Somewhat-Nonuniform Algorithms) *For every nonnegative integer k, every real d, and every positive real ϵ, there exists a real $c > 1$ such that satisfiability cannot both*

(i) have $\Sigma_k\text{TS}(n^d, n^{1-\epsilon})$-uniform conondeterministic circuits of size n^c and

(ii) be in $\Sigma_k\text{TS}(n^d, n^{1-\epsilon})$.

For certain types of circuits, part (i) implies a uniform algorithm for satisfiability that is efficient enough so that we do not need to state (ii). In particular, we obtain the following corollary to the proof of Theorem 1.6.

5

Corollary 1.7 *For every nonnegative integer k and positive real ϵ, satisfiability cannot be solved by $\Sigma_k \mathrm{TS}(n^{O(1)}, n^{1-\epsilon})$-uniform families of any of the following types: circuits of size $n^{1+o(1)}$ and width $n^{1-\epsilon}$, SAC^1-circuits of size $n^{1+o(1)}$, or negations of such circuits.*

Recall that SAC^1-circuits are circuits of logarithmic depth with bounded fan-in ANDs, unbounded fan-in ORs, and negations only on the inputs. NC^1-circuits of size $n^{1+o(1)}$ are a special type of SAC^1-circuits of size $n^{1+o(1)}$. Negations of SAC^1-circuits are equivalent to circuits of logarithmic depth with bounded fan-in ORs, unbounded fan-in ANDs, and negations only on the inputs.

There is another direction in which we can extend the lower bounds to a nonuniform setting. Tourlakis [Tou01] observed that the arguments of Fortnow and of Lipton-Viglas carry through when the machines involved receive subpolynomial advice. The same holds for almost all the results stated in this survey. We refer to Section 3.1 and the end of Section 6 for more details.

Other models of computation that capture important capabilities of current or future computing devices include randomized and quantum machines. To date we know of no nontrivial lower bounds for satisfiability on such models with two-sided error but we do have interesting results for problems that are somewhat harder than satisfiability.

In the setting of randomized computations with two-sided error, the simplest problem for which we can prove nontrivial lower bounds is $\Sigma_2\mathrm{SAT}$, the language consisting of all valid Σ_2-formulas. $\Sigma_2\mathrm{SAT}$ constitutes the equivalent of satisfiability in the second level of the polynomial-time hierarchy.

At first glance, it might seem that results from space-bounded derandomization let us derive time-space lower bounds for randomized algorithms as immediate corollaries to time-space lower bounds for deterministic algorithms. In particular, assuming we have a randomized algorithm that solves satisfiability in logarithmic space and time n^d, Nisan's deterministic simulation [Nis94] yields a deterministic algorithm for satisfiability that runs in polynomial time and polylogarithmic space. However, even for $d = 1$, the degree of the polynomial is far too large for this simulation to yield a contradiction with known time-space lower bounds for deterministic algorithms.

At the technical level, the arguments for satisfiability in the deterministic setting do not carry over to the setting of randomized algorithms with two-sided error. The difficulty is related to the fact that we know efficient simulations of randomized computations with two-sided error in the second level of the polynomial-time hierarchy but not in the first level. Roughly speaking, this is why we have results for $\Sigma_2\mathrm{SAT}$ but not for satisfiability itself. Diehl and Van Melkebeek [DvM06] proved the first lower bound for $\Sigma_2\mathrm{SAT}$ in the randomized setting and still hold the record, namely an almost-quadratic time lower bound for subpolynomial space.

Theorem 1.8 (Diehl-Van Melkebeek [DvM06]) *For every real $d < 2$ there exists a positive real e such that $\Sigma_2\mathrm{SAT}$ cannot be solved by a randomized random-access machine with two-sided error that runs in time n^d and space n^e. Moreover, e approaches $1/2$ from below as d approaches 1 from above.*

Note a few other differences with the deterministic setting. The format of Theorem 1.8 is weaker than that of Theorem 1.3, which entails machines of type (i) and (ii). In the randomized setting, we do not know how to take advantage of the existence of an algorithm for $\Sigma_2\mathrm{SAT}$ that runs in time n^c for small c but unrestricted space to derive better time-space lower bounds for $\Sigma_2\mathrm{SAT}$. The parameters of Theorem 1.8 are also weaker than those of the corresponding result for $\Sigma_2\mathrm{SAT}$ in the deterministic setting, where the bound on d is larger than 2 and e converges to 1 when d goes to 1. See Section 4.2 for the exact bounds for $\Sigma_2\mathrm{SAT}$ in the deterministic setting.

Theorem 1.8 also applies to Π_2SAT, the complement of Σ_2SAT, as randomized computations with two-sided error can be trivially complemented. For the equivalents of satisfiability, tautologies, Σ_2SAT, Π_2SAT, etc. in higher levels of the polynomial-time hierarchy, stronger results can be shown, including results in the model where the randomized machines have two-way sequential access to the random-bit tape. Theorem 1.8 refers to the more natural but weaker coin flip model of space-bounded randomized computation, which can be viewed as equipping a deterministic machine with one-way access to a random bit tape. We refer to Section 7.2 for more details.

In the setting of one-sided error (with errors only allowed on the membership side), we do have lower bounds for the first level of the polynomial-time hierarchy, namely for tautologies. Such results trivially follow from Theorem 1.5 since randomized machines with one-sided error are special cases of nondeterministic machines. For example, we can conclude from Theorem 1.5 that tautologies cannot have both a randomized algorithm with one-sided error that runs in time $n^{1+o(1)}$ and a randomized algorithm with one-sided error that runs in polynomial time and subpolynomial space. Diehl and Van Melkebeek [DvM06] observed that the (then) known lower bound proofs for satisfiability on deterministic machines can be extended to lower bound proofs for tautologies on randomized machines with one-sided error without any loss in parameters. Their argument holds for all proofs to date, including Theorem 1.3. In particular, we know that tautologies cannot be solved by a randomized algorithm with one-sided error that runs in time $n^{2\cos(\pi/7)-o(1)}$ and subpolynomial space.

In the quantum setting, the simplest problem for which we currently know nontrivial lower bounds is MajMajSAT. MajSAT, short for majority-satisfiability, denotes the problem of deciding whether the majority of the assignments to a given Boolean formula satisfy the formula. Similarly, an instance of MajMajSAT asks whether a given Boolean formula depending on two sets of variables y and z has the property that for at least half of the assignments to y, at least half of the assignments to z satisfy the formula.

Allender et al. [AKR+01] showed a lower bound for MajMajSAT on randomized machines with unbounded error. The parameters are similar to those in Fortnow's time-space lower bound for satisfiability. In particular, they prove that MajMajSAT does not have a randomized algorithm with unbounded error that runs in time $n^{1+o(1)}$ and space $n^{1-\epsilon}$. Van Melkebeek and Watson [vMW07], building on earlier work by Adleman et al. [ADH97], showed how to simulate quantum computations with bounded error on randomized machines with unbounded error in a time- and space-efficient way. As a result, they can translate the lower bound of Allender et al. to the quantum setting.

Theorem 1.9 (Van Melkebeek-Watson [vMW07], using Allender et al. [AKR+01]) *For every real d and positive real ϵ there exists a real $c > 1$ such that at least one of the following fails:*

(i) MajMajSAT *has a quantum algorithm with two-sided error that runs in time n^c and*

(ii) MajSAT *has a quantum algorithm with two-sided error that runs in time n^d and space $n^{1-\epsilon}$.*

Corollary 1.10 *For every positive real ϵ there exists a real $d > 1$ such that* MajMajSAT *does not have a quantum algorithm with two-sided error that runs in time n^d and space $n^{1-\epsilon}$.*

There is a – very simple – reduction from satisfiability to MajSAT but presumably not the other way around since MajSAT is hard for the entire polynomial-time hierarchy [Tod91]. The same statement holds for MajMajSAT and Σ_2SAT instead of MajSAT and satisfiability, respectively. The

7

reason why we have quantum lower bounds for MajMajSAT but not for Σ_kSAT for any integer k bears some similarity to why we have randomized lower bounds for Σ_2SAT but not for satisfiability. MajSAT tightly captures randomized computations with unbounded error in the same was as Σ_kSAT captures Σ_k-computations. We can efficiently simulate randomized computations with two-sided error on Σ_2-machines but we do not know how to do so on nondeterministic machines. Similarly, we can efficiently simulate quantum computations with bounded error on randomized machines with unbounded error but we do not know how to do that on Σ_k-machines. This analogy actually suggests that we ought to get quantum lower bounds for MajSAT rather than only for MajMajSAT. We discuss that prospect in Section 9.

1.1 Scope

This paper surveys the known robust lower bounds for the time and space complexity of satisfiability and closely related problems on general-purpose models of computation. The bounds depend on the fundamental capabilities of the model (deterministic, randomized, quantum, etc.) but are robust, up to polylogarithmic factors, with respect to the details of the model specification. For each of the basic models, we focus on the simplest problem for which we can establish nontrivial lower bounds. Except for the randomized and quantum models, that problem is satisfiability (or tautologies).

We do not cover lower bounds on restricted models of computation. The latter includes general-purpose models without random access, such as one-tape Turing machines with sequential access, off-line Turing machines (which have random access to the input and sequential access to a single work tape), and multi-tape Turing machines with sequential access via one or multiple tape heads. In those models, techniques from communication complexity can be used to derive lower bounds for simple problems like deciding palindromes or computing generalized inner products. Time-space lower bounds for such problems immediately imply time-space lower bounds for satisfiability by virtue of the very efficient reductions to satisfiability. However, in contrast to the results we cover, these arguments do not rely on the inherent difficulty of satisfiability. They rather exploit an artifact of the model of computation, e.g., that a one-tape Turing machine with sequential access deciding palindromes has to waste a lot of time in moving its tape head between both ends of the tape. Note that on random-access machines palindromes and generalized inner products can be computed simultaneously in quasi-linear time and logarithmic space. We point out that some of the techniques in this survey lead to improved results on some restricted models of computation, too, but we do not discuss them.

Except in Corollary 1.7, we also do not consider restricted circuit models. In several of those models lower bounds have been established for problems computable in polynomial time. Such results imply lower bounds for satisfiability on the same model provided the problems reduce to satisfiability in a simple way. As we will see in Section 2.3, problems in nondeterministic quasi-linear time are precisely those that have this property in a strong sense – they translate to satisfiability in quasi-linear time and do so in an oblivious way. All of the classical lower bounds on restricted circuit models involve problems in nondeterministic quasi-linear time and therefore also hold for satisfiability up to polylogarithmic factors. These results include the exponential lower bounds for the size of constant-depth circuits (for parity and its cousins), the quadratic lower bound for branching program size (for a version of the element distinctness problem, whose complement lies in nondeterministic quasi-linear time), and the cubic lower bound for formula size (for Andreev's addressing function). See [BS90] for a survey that is still up to date in terms of the strengths of the bounds except for the formula size lower bound [Has98]. We point out that some of the more

recent work in circuit complexity does not seem to have implications for satisfiability. In particular, the non-uniform time-space lower bounds by Ajtai [Ajt99] and their improvements by Beame et al. [BSSV03] do not yield time-space lower bounds for satisfiability. These authors consider a problem in P based on a binary quadratic form, and showed that any branching program for it that uses only $n^{1-\epsilon}$ space for some positive constant ϵ takes time

$$\Omega(n \cdot \sqrt{\log n / \log \log n}). \tag{1}$$

An extremely efficient reduction of the problem they considered to satisfiability is needed in order to obtain nontrivial lower bounds for satisfiability, since the bound (1) is only slightly super-linear. The best known reductions (see Section 2.3.1) do not suffice. Moreover, their problem does not appear to be in nondeterministic quasi-linear time.

1.2 Organization

Section 2 contains preliminaries. Although the details of the model of computation do not matter, we describe a specific model for concreteness. We also specify our notation for complexity classes and exhibit complete problems which capture those classes very tightly such that time-space lower bounds for those problems and for linear time on the corresponding models are equivalent up to polylogarithmic factors. Whereas in this section we have stated all results in terms of the complete problems, in the rest of the paper we will think in terms of linear time on the corresponding models.

We present the known results in a unified way by distilling out what they have in common. Section 3 introduces the proof techniques and the tools involved in proving many of the lower bounds. It turns out that all the proofs have a very similar high-level structure, which can be characterized as indirect diagonalization. We describe how it works, what the ingredients are, and illustrate how they can be combined.

We then develop the results for the various models within this unifying framework: deterministic algorithms in Section 4, nondeterministic algorithms in Section 5, somewhat-nonuniform algorithms in Section 6, randomized algorithms in Section 7, and quantum algorithms in Section 8. We mainly focus on space bounds of the form $n^{1-\epsilon}$ and on subpolynomial space bounds as they allow us to present the underlying ideas without getting bogged down in notation and messy calculations. Sections 4 through 8 are largely independent of each other, although some familiarity with the beginning of Section 4 can help to better appreciate Sections 5, 6, and 7.

Finally, in Section 9 we propose some directions for further research.

2 Preliminaries

This section describes the machine models we use and introduces our notation for complexity classes. It establishes natural complete problems which capture the complexity of some of the models in a very tight way, e.g., satisfiability in the case of nondeterministic computations.

2.1 Machine Models

The basic models of computation we deal with are deterministic, nondeterministic, alternating, randomized, and quantum machines. Up to polylogarithmic factors, our results are robust with respect to the details of each of the basic models. Our arguments work for all variants we know;

for some variants extra polylogarithmic factors arise in the analysis due to simulations of machines within the model.

For concreteness, we describe below the particular deterministic model we have in mind. The nondeterministic, alternating, and randomized models are obtained from it in the standard way by allowing the transition function to become a relation and, for alternating machines, associating an existential/universal character to the states. An equivalent way of viewing our model of randomized computation is as deterministic machines that have one-way sequential access to an additional tape that is filled with random bits before the start of the computation. The model for quantum computing needs some more discussion because of the issue of intermediate measurements. We postpone that exposition to Section 8.2.

As the basic deterministic model we use random-access Turing machines with an arbitrary number of tapes. The machine can have two types of tapes: sequential-access tapes and random-access tapes (also referred to as indexed tapes). Each random-access tape T has an associated sequential-access tape I, which we call its index tape. The machine can move the head on T in one step to the position indexed by the contents of I. The contents of I is erased in such a step. The input tape is read-only; the output tape is write-only with sequential one-way access. The input and output tapes do not count towards the space usage of the machine. Non-index tapes contribute the largest position ever read (indexed) to the space usage. The latter convention seems to capture the memory requirements for random-access machines better than the alternative where we count the number of distinct cells accessed. It does imply that the space can be exponential in the running time. However, by using an appropriate data structure to store the contents of the tape cells accessed, we can prevent the space from being larger than the running time without blowing up the running time by more than a polylogarithmic factor.

A *configuration* of a machine M consists of the internal state of M, the contents of the work and index tapes, and the head positions on the index tapes. We use the notation $C \vdash_{M,x}^t C'$ to denote that machine M on input x can go from configuration C to configuration C' in t steps. The *computation tableau* of M on input x is a representation of the entire computation. It consists of a table in which successive rows describe the successive configurations of M on input x, starting from the initial configuration of M. If M runs in time t and space s, each configuration has size $O(s)$ and the computation tableau contains at most t rows.

2.2   Complexity Classes

Based on our machine models, we define the languages they decide in the standard way. We use the same notation for classes of machines and for the corresponding class of languages. More often than not, both interpretations work; if not, the correct interpretation should be clear from the context. Our acronyms may be a bit shorter than the usual ones and we introduce some additional ones. We use the following letters to denote a machine type X: D for deterministic, N for nondeterministic, Σ_k for alternating with at most $k-1$ alternations and starting in an existential state, Π_k for alternating with at most $k-1$ alternations and starting in a universal state, P for randomized with unbounded error, BP for randomized with two-sided error, R for randomized with one-sided error, and BQ for quantum with two-sided error. Bounded error always means that the error probability is bounded by 1/3. All of the above machine types have a natural complementary type. We use coX to denote the complementary type of X.

For functions $t, s : \mathbb{N} \to \mathbb{N}$, we denote by $\text{XTS}(t, s)$ the class of machines of type X that run in time $O(t(n))$ and space $O(s(n))$ on inputs of length n. $\text{XT}(t)$ denotes the same without the

space bound. We also define a shorthand for computations where the amount of space is negligible compared to the time. We formalize the latter as the space bound s being subpolynomial in the time bound t, i.e., $s = t^{o(1)}$. We substitute a lower-case "s" for the capital "S" in the notation to hint at that:

$$\mathrm{XTs}(t) = \mathrm{XTS}(t, t^{o(1)}).$$

Note that if t is polynomial then $t^{o(1)}$ is subpolynomial.

For alternating computations, we introduce some notation that allows us to make the number of guess bits during the initial phases explicit.

Definition 2.1 *Starting from an alternating class $\mathcal{C}$ with time bound t and space bound s, we inductively define new classes $\exists^g \mathcal{C}$ and $\forall^g \mathcal{C}$ for any function $g : \mathbb{N} \to \mathbb{N}$. $\exists^g \mathcal{C}$ consists of all languages decided by alternating machines that act as follows on an input x of length n: existentially guess a string y of length $O(g(n))$ and then run a machine M from $\mathcal{C}$ on input $\langle x, y \rangle$ for $O(t(n))$ steps and using $O(s(n))$ space. The class $\forall^g \mathcal{C}$ is obtained in an analogous way; the guess of y now happens in a universal rather than existential mode.*

Let us point out a few subtleties in Definition 2.1. First, although the machine M runs on input $\langle x, y \rangle$, we measure its complexity in terms of the length of the original input x. For example, machines of type $\exists^{n^2} \forall^{\log n} \mathrm{DTs}(n)$ have a final deterministic phase that runs in time linear rather than quadratic in $|x|$. The convention of expressing the resources in terms of the original input length turns out to be more convenient for the arguments in this survey. The second subtlety arises when we consider space-bounded classes $\mathcal{C}$. Computations corresponding to $\exists^g \mathcal{C}$ and $\forall^g \mathcal{C}$ explicitly write down their guess bits y and then run a space-bounded machine on the combined input consisting of the original input x and the guess bits y. Thus, the space-bounded machine effectively has two-way access to the guess bits y. For example, although machines corresponding to $\exists^n \mathrm{DTs}(n)$ and to $\mathrm{NTs}(n)$ both use only a subpolynomial amount of space to verify their guesses, they do not necessarily have the same computational power. This is because the former machines have two-way access to the guess bits, which are written down on a separate tape that does not count toward its space bound, whereas the latter machines only have one-way access to these bits and do not have enough space to write them down on their work tape.

For randomized machines, we default to the standard coin flip model, in which a deterministic machine has one-way read-only access to a tape that is initialized with random bits. If the machine wishes to reread random bits, it must copy them down on a worktape at the cost of space, as opposed to the more powerful model which has two-way access to the random tape. Except where stated otherwise, results about randomized machines refer to the former machine model.

By a function we always mean a function from the set $\mathbb{N}$ of nonnegative integers to itself. Time and space bounds are functions that are at least logarithmic. Note that we do consider sublinear time bounds. We will not worry about constructibility issues. Those arise when we perform time and/or space bounded simulations, such as in padding arguments or hierarchy results. We tacitly assume that the bounds we are working with satisfy such requirements. The bounds we use in the statements of the results are typically polynomials, i.e., functions of the form n^d for some positive real d. Polynomials with rational d are sufficiently smooth and meet all the constructibility conditions we need. For completeness, we sketch an example of a padding argument.

Proposition 2.2 *If*

$$\mathrm{NT}(n) \subseteq \mathrm{DTS}(n^d, n^e)$$

for some reals d and e, then for every bound $t(n) \geq n + 1$

$$\text{NT}(t) \subseteq \text{DTS}(t^d + t, t^e + \log t).$$

Proof Sketch: Let L be a language that is accepted by a nondeterministic machine M that runs in time $O(t)$. Consider the padded language $L' = \{x10^{t(|x|)-|x|-1} \mid x \in L\}$. The language L' is accepted by a nondeterministic machine that acts as follows on an input y of length N. First, the machine verifies that y is of the form $y = x10^k$ for some string x and integer k, determines the length n of x, stores n in binary, and verifies that $t(n) = N$. The constructibility of t allows us to verify the latter condition in time linear in N. Second, we run M on input x, which takes time $t(n)$. Overall, the resulting nondeterministic machine for L' runs in time $O(N)$. By our hypothesis, there also exists a deterministic machine M' that accepts L' and runs in time $O(N^d)$ and space $O(N^e)$.

We then construct the following deterministic machine accepting L. On input x of length n, we use the constructibility of t to compute $N = t(n)$ and write n and N down in binary. This step takes time $O(t(n))$ and space $O(\log t(n))$. Next, we simulate a run of M' on input $y = x10^{t(n)-n-1}$. We do so without storing y explicitly, using the input x, the values of n and N in memory, and comparing indices on the fly. The second step takes time $O((t(n))^d)$ and space $O((t(n))^e)$, resulting in overall requirements of $O((t(n))^d + t(n))$ for time and $O((t(n))^e + \log t(n))$ for space. $\quad\Box$

We measure the size of *circuits* by the bit-length of their description and assume a description that allows evaluation of the circuit in quasi-linear time, i.e., in time $O(n \cdot (\log n)^{O(1)}$. For circuits with bounded fan-in, the size is roughly equal to the number of gates and to the number of connections. For circuits with unbounded fan-in, the size is roughly equal to the latter but not necessarily to the former. For a function t we use $\text{SIZE}(t)$ to denote the class of languages that can be decided by a family of circuits $(C_n)_{n\in\mathbb{N}}$ such that the size of C_n is $O(t(n))$. We define $\text{NSIZE}(t)$ similarly using nondeterministic circuits. We call a family $(C_n)_n$ of circuits $\mathcal{C}$-uniform if all of the following problems lie in $\mathcal{C}$ as a function of the size of the circuit: given an input x, labels g_1 and g_2 of nodes of the circuit $C_{|x|}$, and an index i, decide the type of g_1 (type of gate, input with value 0, or input with value 1), and decide whether g_1 it the ith gate that directly feeds into g_2. For classes $\mathcal{C}$ of the form $\Sigma_k\text{TS}(t,s)$ for positive integers k, being $\mathcal{C}$-uniform is equivalent up to polylogarithmic factors to the requirement that the following problem lies in $\mathcal{C}$ as a function of the size of the circuit: given an input x, an index i, and a bit b, decide whether the ith bit of the description of $C_{|x|}$ equals b.

2.3 Complete Problems

The following are standard completeness results under deterministic polynomial-time mapping reductions, also known as Karp reductions. Satisfiability consists of all Boolean formulas that have at least one satisfying assignment. Satisfiability is complete for NP. Tautologies are Boolean formulas that are true under all possible assignments. Tautologies is complete for coNP. For any positive integer k, $\Sigma_k\text{SAT}$ denotes the language consisting of all true Σ_k-formulas. $\Sigma_1\text{SAT}$ is equivalent to satisfiability. $\Sigma_k\text{SAT}$ is complete for $\Sigma_k^p = \cup_{d\geq 1}\Sigma_k\text{T}(n^d)$. Similarly, $\Pi_k\text{SAT}$ denotes all true Π_k-formulas. $\Pi_1\text{SAT}$ is equivalent to tautologies, and $\Pi_k\text{SAT}$ is complete for $\Pi_k^p = \cup_{d\geq 1}\Pi_k\text{T}(n^d)$. MajSAT consists of all Boolean formulas that have a majority of satisfying assignments. The problem is complete for $\text{PP} = \cup_{d\geq 1}\text{PT}(n^d)$.

For our purposes, we need stronger notions of completeness than the standard ones. The NP-completeness of satisfiability implies that time lower bounds for satisfiability and for NP are equivalent up to polynomial factors. In fact, on each of the standard models of computation, time-space lower bounds for satisfiability and for nondeterministic linear time are equivalent up to polylogarithmic factors. This follows because satisfiability is complete for nondeterministic quasi-linear time, i.e., time $n \cdot (\log n)^{O(1)}$, under very simple reductions. More generally, Σ_kSAT is complete for the Σ_k-level of the quasi-linear-time hierarchy under such reductions. A similar statement holds for MajSAT and quasi-linear time on randomized machines with unbounded error. We argue the hardness, as that is the only part we need for our results.

Lemma 2.3 *For any positive integer k, every language in $\Sigma_k \mathrm{T}(n)$ Karp-reduces to Σ_kSAT in deterministic time $O(n \cdot (\log n)^{O(1)})$. Moreover, given an input x of length n and an index i, the ith bit of the reduction is computable by a deterministic machine that runs in time $(\log n)^{O(1)}$ and space $O(\log n)$. The same holds if we replace $\Sigma_k \mathrm{T}(n)$ and Σ_kSAT by $\mathrm{PT}(n)$ and MajSAT, respectively.*

Several proofs of Lemma 2.3 exist [FLvMV05, Tou01]. They all build on earlier work [PF79, Coo88, Rob91] and, one way or the other, rely on the Hennie-Stearns oblivious efficient simulation of multi-tape Turing machines with sequential access by 2-tape machines with sequential access [HS66]. In Section 2.3.1 we present a simple proof that avoids the latter component. In some sense, the need for oblivious simulations is reduced from a general computation to the task of sorting, for which we employ elementary efficient sorting networks, which are inherently oblivious. The latter can be built using a divide-and-conquer strategy, which we deem considerably simpler than the Hennie-Stearns construction. Our proof is also robust with respect to the model of computation.

Combined with the fact that Σ_kSAT belongs to $\Sigma_k \mathrm{T}(n \cdot (\log n)^{O(1)})$, Lemma 2.3 implies that time-space lower bounds for Σ_kSAT and for $\Sigma_k \mathrm{T}(n)$ are equivalent up to polylogarithmic factors. The same holds, mutatis mutandis, for MajSAT and $\mathrm{PT}(n)$. In particular, we obtain the following for time and space bounds in the polynomial range.

Corollary 2.4 *Let X denote any of the machine types from Section 2.2 or their complements. Let k be a positive integer, and let d and e be reals. If*

$$\Sigma_k \mathrm{T}(n) \nsubseteq \mathrm{XTS}(n^d, n^e),$$

then for any reals $d' < d$ and $e' < e$

$$\Sigma_k \mathrm{SAT} \notin \mathrm{XTS}(n^{d'}, n^{e'}).$$

The same holds if we replace $\Sigma_k \mathrm{T}(n)$ and Σ_kSAT by $\mathrm{PT}(n)$ and MajSAT, respectively.

The simple reductions to satisfiability that underly Lemma 2.3 for $k = 1$ also exist to all of the standard natural NP-complete problems. In fact, to the best of my knowledge, all known natural NP-complete problems in nondeterministic quasi-linear time share the latter property. Consequently, the case $k = 1$ of Lemma 2.3 and of Corollary 2.4 holds if we replace satisfiability by any of these problems.

From now on, our goal will be to obtain time and space lower bounds for nondeterministic linear time, $\Sigma_k \mathrm{T}(n)$ with $k > 1$, and randomized linear time with unbounded error. For example, we will prove results of the form $\mathrm{NT}(n) \nsubseteq \mathrm{DTS}(n^d, n^e)$ for some constants $d \geq 1$ and $e > 0$. Results for satisfiability, other NP-complete problems, for Σ_kSAT with $k > 1$, or for MajSAT then follow via Corollary 2.4.

2.3.1   Proof of Lemma 2.3

We first give the proof for $k = 1$, i.e., we argue the hardness of satisfiability for nondeterministic linear time under very efficient Karp-reductions. The generalization for larger values of k will follow easily.

We start the proof with a technical claim. In principle, a linear-time nondeterministic machine M can access locations on non-index tapes that have addresses of linear length. We claim that without loss of generality, we can assume that these addresses are at most of logarithmic length. The reason is that we can construct a nondeterministic Turing machine M' that simulates M with only a constant factor overhead in time and satisfies the above restriction. For each non-index tape T of M, M' uses an additional non-index tape T' on which M' stores a list of all (address,value) pairs of cells of T which M accesses and that have an address value of more than logarithmic length. During the simulation of M, M' uses T in the same way as M does to store the contents of the cells of T with small addresses; it uses T' for the remaining cells of T accessed by M. M' can keep track of the (address,value) pairs on tape T' in an efficient way by using an appropriate data structure, e.g., sorted doubly linked lists of all pairs corresponding to addresses of a given length, for all address lengths used. Note that the list of (address,value) pairs is at most linear in size so the index values M' uses on T' are at most logarithmic. By using the power of nondeterminism to guess the right tape locations, M' can easily retrieve a pair, insert one, and perform the necessary updates with a polylogarithmic factor overhead in time. Thus, M' simulates M with a polylogarithmic factor overhead in time and only accesses cells on its tapes with addresses of at most logarithmic length.

With each computation step of M', we can associate a block consisting of a logarithmic number of Boolean variables that represent the following information about that step: the transition of the finite control of M', and the contents of the index tapes, the tape head positions of all tapes that are not indexed, and the contents of all tape cells accessed at the beginning of that step. We can verify that a sequence of such blocks represents a valid accepting computation of M' on a given input x by checking: (i) that the initial block corresponds to a valid transition out of an initial configuration of M', (ii) that all pairs of successive computation steps are consistent in terms of the internal state of M', the contents of the index tapes, and the tape head positions of all tapes that are not indexed, (iii) that the accesses to the indexed non-input tapes are consistent, (iv) that the accesses to the input tape are consistent with the input x, and (v) that the final step leads to acceptance. By the standard proofs of the NP-completeness of satisfiability, conditions (i), (v), and each of the linear number of constituent conditions of (ii) can be expressed by clauses of polylogarithmic size using the above variables and additional auxiliary variables. Each bit of those clauses can be computed in polylogarithmic time and logarithmic space. All that remains is to show that the same can be done for conditions (iii) and (iv).

We check the consistency of the accesses to the indexed non-input tapes for each tape separately. Suppose that, for a given tape T, we have the blocks sorted in a stable way on the value of the corresponding index tape in that block. Then we can perform the consistency check for tape T by looking at all pairs of consecutive blocks and verifying that, if they accessed the same cell of T, the contents of that cell in the second block is as dictated by the transition encoded in the first block, and if they accessed different cells, then the contents of the cell in the second block is blank. These conditions can be expressed in the same way as (ii) above.

In order to obtain the blocks in the required sorted order, we use efficiently constructible stable sorting networks of quasi-linear size, such as Batcher's networks. These are built using the merge-sort divide-and-conquer strategy, where each (so-called odd-even) merger network is constructed

using another level of divide-and-conquer. The resulting sorting network is of size $O(n \log^2 n)$ and each connection can be computed in polylogarithmic time and logarithmic space. We refer to [CLRS01, chapter 28] for more details about sorting networks. We associate a block of Boolean variables with each connection in the network and include clauses that enforce the correct operation of each of the comparator elements of the network. The latter conditions can be expressed in a similar way as condition (ii) above. The size and constructibility properties of the network guarantee that the resulting Boolean formula is of quasi-linear size and such that each bit can be computed in polylogarithmic time and logarithmic space.

The consistency of the input tape accesses with the actual input x can be checked in a similar way as condition (iii). The only difference is that before running the stable sorting for the input tape, we prepend n dummy blocks, the ith of which has the input tape head set to location i. The approach for handling condition (iii) then enforces that all input accesses are consistent with the values encoded in the dummy blocks. Since we know explicitly the variable that encodes the ith input bit in the dummy blocks, we can include simple clauses that force that variable to agree with the ith bit of x.

This finishes the proof of the case $k = 1$. For larger values of k, we use induction and exploit the fact that the formula we produce in the case $k = 1$ depends on the input length but is oblivious to the actual input x of that length. More precisely, on input length n the reduction produces a Boolean formula φ_n in variables x and y such that for every input x of length n there exists a setting of y that makes $\varphi(x, y)$ evaluate to true iff x is accepted by M. For larger values of k, the reduction first produces $k - 1$ blocks consisting of a linear number of variables $y_1, y_2, \ldots, y_{k-1}$, which correspond to the (co)nondeterministic choices made during the first $k-1$ alternating phases of the computation. Then the reduction applies the above procedure for $k = 1$ to the remaining (co)nondeterministic linear-time computation on the combined input consisting of the original input x and the variables $y_1, y_2, \ldots, y_{k-1}$ of the first $k-1$ blocks, resulting in an oblivious Boolean formula $\varphi(x, y_1, \ldots, y_k)$ for the matrix of the Σ_k-formula.

The result for MajSAT can be shown by exploiting the additional property that the translation for $k = 1$ is parsimonious, i.e., the number of settings of the variables y that satisfy $\varphi(x, y)$ equals the number of accepting computations of M on input x. We can then create a new Boolean formula $\varphi'(x, y, b) \equiv (\varphi(x, y) \wedge b = 0) \vee (\varphi''(y) \wedge b = 1)$, where φ'' is a Boolean formula that offsets the number of assignments to (y, b) that satisfy $\varphi'(x, y, b)$ in such a way that the total count is more than half iff x is accepted by M in the probabilistic sense.

3   Common Structure of the Arguments

This section describes the common fabric of the lower bounds presented in this survey. All of the arguments share the same high-level structure, which can be characterized as indirect diagonalization.

Indirect diagonalization is a technique to separate complexity classes. In the case of lower bounds for satisfiability on deterministic machines, we would like to obtain separations of the form $\mathrm{NT}(n) \not\subseteq \mathrm{DTS}(t, s)$ or $\mathrm{NT}(n) \not\subseteq \mathrm{coNT}(n^c) \cap \mathrm{DTS}(t, s)$ for some interesting values of the parameters t, s, and c. The proofs go by contradiction and have the following outline:

1. We assume that the separation does not hold, i.e., we assume the unlikely inclusion $\mathrm{NT}(n) \subseteq \mathrm{DTS}(t, s)$ or $\mathrm{NT}(n) \subseteq \mathrm{coNT}(n^c) \cap \mathrm{DTS}(t, s)$.

2. Next, using our hypothesis, we derive more and more unlikely inclusions of complexity classes.

3. Finally, we derive a contradiction with a direct diagonalization result.

The techniques we use to derive more inclusions in step 2 go in two opposing directions:

(a) speeding up deterministic space-bounded computations by introducing more alternations, and

(b) using the hypothesis to eliminate alternations at a moderate increase in running time.

The hypothesis $\text{NT}(n) \subseteq \text{DTS}(t, s)$ allows us to simulate nondeterministic computations on deterministic space-bounded machines, which brings us in the realm of (a). The hypothesis $\text{NT}(n) \subseteq \text{DTS}(t, s)$ or $\text{NT}(n) \subseteq \text{coNT}(n^c)$ makes (b) possible. By combining (a) and (b) in the right way, we can speed up nondeterministic computations in the first level of the polynomial-time hierarchy, which can be shown impossible by a simple direct diagonalization argument.

More generally, we are shooting for separations of the form $\mathcal{C}_1 \nsubseteq \mathcal{C}_2 \cap \mathcal{C}_3$. We are in the situation where there exists a hierarchy of classes built on top of $\mathcal{C}_1$ such that

(ã) $\mathcal{C}_3$ can be sped up in higher levels of the hierarchy, and

(b̃) computations in a certain level of the hierarchy can be simulated in a lower level with a slowdown which is small if we assume that $\mathcal{C}_1 \subseteq \mathcal{C}_2$.

An appropriate combination of those two transformations allows us to speed up computations within the same level of the hierarchy, which contradicts a direct diagonalization result.

In the rest of this section we first list the direct diagonalization results we use for step 3 of the indirect diagonalization paradigm. Then we describe the two techniques (a) and (b) for step 2. They are all we need to derive each of the lower bounds in this survey except the quantum ones. In the quantum setting we go through intermediate simulations in the so-called counting hierarchy rather than in the polynomial-time hierarchy. We will describe the equivalents of (ã) and (b̃) in the counting hierarchy when we get to the quantum bounds in Section 8. We end this section with a concrete instantiation to illustrate the above paradigm.

3.1 Direct Diagonalization Results

A variety of direct diagonalization results have been used in the literature to derive the lower bounds discussed in this survey. Almost all of the arguments can be reformulated in such a way that the following straightforward direct diagonalization result suffices. The result says that computations in any fixed level of the polynomial-time hierarchy cannot be sped up in general by complementation within the same level. We state it formally for polynomial time bounds as that version is robust with respect to the details of the model of computation and is all we need. We include a proof sketch for completeness.

Lemma 3.1 *Let k be a positive integer and a, b be reals such that $1 \le a < b$. Then*

$$\Sigma_k \text{T}(n^b) \nsubseteq \Pi_k \text{T}(n^a).$$

Proof Sketch: The idea is to use an efficient universal machine U for Σ_k-machines to complement the behavior of every Π_k-machine N that runs in time $O(n^a)$, on some input depending on N. This works because complementing a Π_k-machine is equivalent to running a Σ_k-machine for the same

number of steps. Thus, U only needs to simulate Σ_k-machines that run in time $O(n^a)$, which it can do in time n^b.

The universal machine U takes as input a pair $\langle x, y \rangle$, interprets x as the description of a Σ_k-machine, and simulates that machine on input y. The construction of U involves reducing the number of tapes of Σ_k-machines to a constant, as U can only have a fixed number of tapes. By interleaving tape cells we can simulate every Σ_k-machine with an arbitrary number of tapes on a Σ_k-machine with a fixed number ℓ of tapes. The simulation only requires a subpolynomial overhead in time.

Consider the Σ_k-machine M that takes an input x and runs U on input $\langle x, x \rangle$. We clock M such that it runs in time $|x|^b$. The language L decided by M lies in $\Sigma_k\mathrm{T}(n^b)$ by construction.

Consider an arbitrary Π_k-machine N that runs in time n^a. By swapping the existential/universal characteristics of the states, as well as the accept/reject characteristics, we transform N into a Σ_k-machine that accepts the complementary language of N. Since there are infinitely many equivalent descriptions of machines, there are infinitely many strings x that describe a Σ_k-machine that does the opposite of N and runs in the same time as N. For large enough strings x in that sequence, U finishes its computation on input $\langle x, x \rangle$ before the clock kicks in, and therefore M does the opposite of what N does on input x. Thus, N cannot decide the same language L that M decides. Since the latter holds for every Π_k-machine N running in time $O(n^a)$, L is not in $\Pi_k\mathrm{T}(n^a)$. $\square$

As much as possible, we will cast the lower bound arguments as indirect diagonalizations that use Lemma 3.1 for the direct diagonalization result in step 3. The proof of Lemma 3.1 is not only simple but also carries through when the machines running in the smaller time n^a receive up to n bits of advice at length n. This translates into lower bounds that hold even for machines that take a subpolynomial amount of advice. See the end of Section 6 for more details about the translation.

For a few lower bounds, we will also need the time hierarchy theorem for alternating machines, which says that for any fixed type of machines we can decide strictly more languages when given a bit more time. For the same reasons as above, we only state the result for polynomial time bounds. Various proofs are known; we briefly sketch the one by Zak [Zak83]. The proofs are more complicated than the proof of Lemma 3.1 due to the difficulty of complementation on alternating machines of a fixed type. In the case of Lemma 3.1 the complementation step is easy because we switch from one type of machine to the complementary type.

Lemma 3.2 (Cook [Coo73], Seiferas-Fischer-Meyer [SFM78], Zak [Zak83]) *Let k be a positive integer and a, b be reals such that $1 \leq a < b$. Then*

$$\Sigma_k\mathrm{T}(n^a) \subsetneq \Sigma_k\mathrm{T}(n^b).$$

Proof Sketch: We use the efficient universal machine U from the proof of Lemma 3.1. Instead of reserving single inputs x in order to diagonalize against a given machine N that runs in time $O(n^a)$, we now use large intervals I of lexicographically successive inputs.

Here is the idea. Although complementation may take a long time, we can make the interval I sufficiently long such that on the input x_f that forms the end of the interval, $|x_f|^b$ is enough time for M to deterministically complement N on the input x_i that forms the beginning of the interval. On the rest of the interval we can define M in such a way that if N agrees with M, N is forced to copy the behavior of M at the end of the interval all the way down to the beginning of the interval. This cannot be since we constructed $M(x_f)$ to be different from $N(x_i)$.

The copying process is realized as follows. On every input $x \in I$ except the last string of I, M uses U to simulate N on the lexicographically next input. As before, we clock M such that it runs in time n^b. If N runs in time $O(n^a)$, M will be able to finish the simulations in time for sufficiently large x and thus realize the copying process under the assumption that M and N agree on I. $\quad \square$

Apart from the fact that the proof of Lemma 3.2 is more complicated than the one of Lemma 3.1, it can also only handle a constant rather than n bits of advice on the smaller time side. See [vMP07] for more details.

In the quantum setting, we will make use of the hierarchy theorem for randomized computations with unbounded error. The proof of that result is similar to the one of Lemma 3.1 except simpler because complementation is very easy on randomized machines with unbounded error. As in the case of Lemma 3.1, the argument can handle up to n bits on advice on the smaller time side.

Lemma 3.3 *Let a, b be reals such that $1 \leq a < b$. Then*

$$\mathrm{PT}(n^a) \subsetneq \mathrm{PT}(n^b).$$

3.2   Speeding Up Space-Bounded Computations Using Alternations

We now start our discussion of the tools we use to derive from our initial hypothesis a contradiction with one of the direct diagonalization results of the previous section. The first tool is speeding up computations by allowing more alternations. We know how to do this in general for space-bounded computations. The technique consists of a divide-and-conquer strategy. It has been known for a long time and has been applied extensively in computational complexity, for example, in the proof of Savitch's theorem [Sav70].

Let us explain the idea for *nondeterministic* computations first. Suppose we have a nondeterministic machine M that runs in space s. We are given two configurations C and C' of M on an input x, and would like to know whether M can go from C to C' in t steps. One way to do this is to run the machine for t steps from configuration C and check whether we can end up in configuration C'. In other words, we fill in the whole tableau in Figure 3(a) row by row.

Using the power of alternation, we can speed up this process as follows. We can break up the tableau into b equal blocks, guess configurations $C_1, C_2, \ldots, C_{b-1}$ for the common borders of the blocks, treat each of the blocks i, $1 \leq i \leq b$, as a subtableau and verify that M on input x can go from configuration C_{i-1} to C_i in t/b steps. See Figure 3(b).

In terms of logical formulas, we are using the following property of configurations:

$$C \vdash^t C' \Leftrightarrow (\exists\, C_1, C_2, \ldots, C_{b-1})(\forall\, 1 \leq i \leq b)\, C_{i-1} \vdash^{t/b} C_i, \tag{2}$$

where $C_0 \doteq C$ and $C_b \doteq C'$. We can perform this process on a Σ_3-machine using time $O(bs)$ for guessing the $b - 1$ intermediate configurations of size s each in the first existential phase, time $O(\log b)$ to guess the block i we want to verify in the universal phase, and time $O(t/b)$ to nondeterministically run M for t/b steps to verify the ith block in the final existential phase. Using the notation we introduced in Definition 2.1, we obtain

$$\mathrm{NTS}(t, s) \subseteq \exists^{bs} \forall^{\log b} \mathrm{NTS}(t/b, s) \subseteq \Sigma_3 \mathrm{T}(bs + t/b). \tag{3}$$

The running time of the Σ_3-machine is minimized (up to a constant) by choosing $b = \sqrt{t/s}$, resulting in

$$\mathrm{NTS}(t, s) \subseteq \Sigma_3 \mathrm{T}(\sqrt{ts}). \tag{4}$$
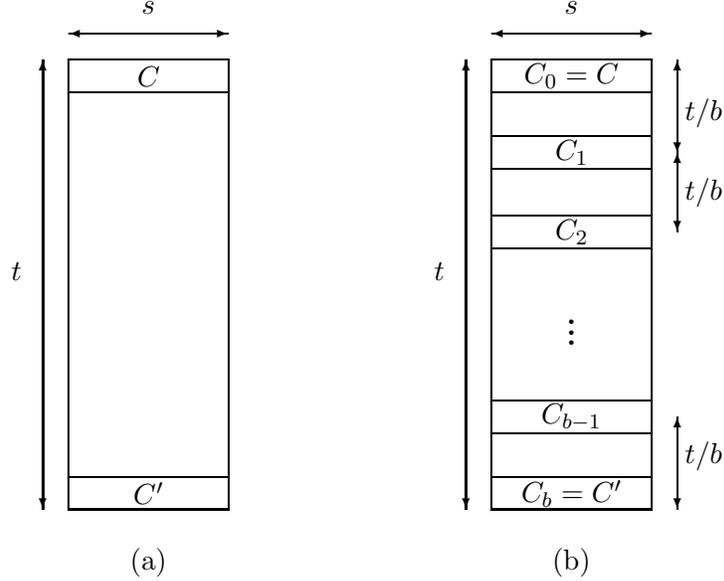
18

Figure 3: Tableaus of a computation using time t and space s

We point out for future reference that the final phase of the computation only needs access to the global input x and two configurations (denoted C_{i-1} and C_i in (2)), not to any of the other configurations guessed during the first phase.

The final phase of our simulation consists of an easier instance of our original problem, namely nondeterministically checking whether M can go from one configuration to another in a certain number of steps. Therefore, we can apply the divide-and-conquer strategy again, and again. Each application increases the number of alternations by 2. k recursive applications with block numbers $b_1, b_2, \ldots, b_k$, respectively, yield:

$$\text{NTS}(t, s) \subseteq \exists^{b_1 s} \forall^{\log b_1} \exists^{b_2 s} \forall^{\log b_2} \ldots \exists^{b_k s} \forall^{\log b_k} \text{NTS}(t / \prod_i b_i, s)$$

$$\subseteq \Sigma_{2k+1} \text{T}\left((\sum_i b_i)s + t/(\prod_i b_i) \right). \tag{5}$$

The running time of the Σ_{2k+1}-machine is minimized (up to a constant) by picking the block numbers all equal to $(t/s)^{1/(k+1)}$. We obtain:

$$\text{NTS}(t, s) \subseteq \Sigma_{2k+1} \text{T}((ts^k)^{1/(k+1)}). \tag{6}$$

We point out for later reference that minimizing the running time of the Σ_{2k+1}-machine may not be the best thing to do if this simulation is just an intermediate step in a derivation. In particular, in several applications the optimal block numbers will not all be equal.

One application of (6) is Nepomnjascii's theorem [Nep70], which states that $\text{NTS}(n^{O(1)}, n^{1-\epsilon})$ is included in the linear-time hierarchy for every positive real ϵ.

Lemma 3.4 (Nepomnjascii [Nep70]) *For every real d and positive real ϵ there exists an integer k such that*

$$\text{NTS}(n^d, n^{1-\epsilon}) \subseteq \Sigma_k \text{T}(n).$$

For *alternating* space-bounded machines M that are more complicated than nondeterministic machines, we can apply a similar strategy to each of the phases of the computation. For an existential phase, we can guess the configuration C_b at the end of the phase, apply (2), and verify that C_b is an accepting configuration on the given input. For the latter we can use a complementary strategy and handle the subsequent universal phase of the computation, etc. This leads to a generalization of (3) and of Nepomnjascii's Theorem to an arbitrary constant number of alternations.

Lemma 3.5 (Kannan [Kan84]) *For every integer k, real d, and positive real ϵ, there exists an integer ℓ such that*

$$\Sigma_k \mathrm{TS}(n^d, n^{1-\epsilon}) \subseteq \Sigma_\ell \mathrm{T}(n).$$

For *deterministic* machines, the same divide-and-conquer strategy (2) as for nondeterministic machines applies, leading to the inclusions:

$$\mathrm{DTS}(t,s) \subseteq \exists^{bs} \forall^{\log b} \mathrm{DTS}(t/b, s) \subseteq \Sigma_2 \mathrm{T}(bs + t/b) \tag{7}$$

and

$$\mathrm{DTS}(t,s) \subseteq \Sigma_2 \mathrm{T}(\sqrt{ts}). \tag{8}$$

Note that we have one fewer alternation in (7) and (8) than in the corresponding (3) and (4) because the final phase is now deterministic rather than nondeterministic. In the recursive applications corresponding to (5) and (6) we can do with even fewer alternations – we can realize the same savings in running time as in (5) and (6) with roughly only half the number of alternations. This is because deterministic computations are closed under complementation, which allows us to align adjacent quantifiers in successive applications of the basic speedup (2) by complementing between applications; that way we induce only one instead of two additional quantifier alternations per application.

Another way to view this is as exploiting the following property of deterministic computations.

$$C \vdash^t C' \Leftrightarrow (\forall C'' \neq C') \, C \not\vdash^t C''. \tag{9}$$

That is, a deterministic machine M goes from a configuration C to a configuration C' in t steps iff for every configuration C'' different from C', M cannot reach C'' from C in t steps. To verify the latter we use the divide-and-conquer strategy from Figure 3. We replace the matrix of (9) by the negation of the right-hand side of (2) and rename C'' to C_b for convenience.

$$C \vdash^t C' \Leftrightarrow (\forall C_b \neq C')(\forall C_1, C_2, \ldots, C_{b-1})(\exists\, 1 \leq i \leq b)\, C_{i-1} \not\vdash^{t/b} C_i, \tag{10}$$

where C_0 denotes C. In terms of the tableau of Figure 4, M reaches C' from C in t steps iff the following holds: If we break up the tableau into b blocks then for every choice of intermediate configurations C_i, $1 \leq i \leq b-1$, and of a final configuration C_b other than C', there has to be a block i that cannot be completed in a legitimate way.

Applying this idea recursively amounts to replacing the matrix $C_{i-1} \not\vdash^{t/b} C_i$ of the Π_2-formula (10) by a Σ_2-formula which is the negation of a formula of the same type as the whole right-hand side of (10). The existential quantifiers merge and the resulting formula is of type Π_3. In general, k recursive applications result in a Π_{k+1}-formula. If we denote the block numbers for the successive
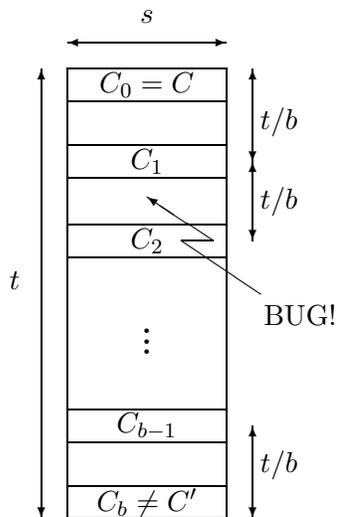
Figure 4: Saving alternations

recursive applications by $b_1, b_2, \ldots, b_k$, we conclude in a similar way as in (5) that

$$\mathrm{DTS}(t,s) \quad \subseteq \quad \forall^{b_1 s} \exists^{\log b_1} \exists^{b_2 s} \forall^{\log b_2} \ldots Q^{\log b_{k-1}} Q^{b_k s} \overline{Q}^{\log b_k} \mathrm{DTS}(t/\prod_i b_i, s) \tag{11}$$

$$\subseteq \quad \Pi_{k+1} \mathrm{T}\left((\sum_i b_i)s + t/(\prod_i b_i) \right), \tag{12}$$

where $Q = \forall$ for odd k, $Q = \exists$ for even k, and $\overline{Q}$ denotes the complementary quantifier of Q. By picking the block numbers in a way to minimize the running time in (12), we obtain

$$\mathrm{DTS}(t,s) \subseteq \Pi_{k+1} \mathrm{T}((ts^k)^{1/(k+1)}). \tag{13}$$

As promised, we realize the same speed-up as in (5) and (6) with only about half as many alternations.

3.3 Eliminating Alternations

The other tool we use to derive more unlikely inclusions of complexity classes from our hypothesis consists of the opposite of what we just did – eliminating alternations at a moderate cost in running time.

In general, we only know how to remove an alternation at an exponential cost in running time. However, a hypothesis like $\mathrm{NT}(n) \subseteq \mathrm{coNT}(n^c)$ for a small real $c \geq 1$ means that we can efficiently simulate nondeterminism conondeterministically and thus eliminate alternations at a moderate expense.

Proposition 3.6 *Let k be a positive integer, $c \geq 1$ be a real, and t be a function. If*

$$\mathrm{NT}(n) \subseteq \mathrm{coNT}(n^c)$$

then

$$\Sigma_{k+1}\mathrm{T}(t) \subseteq \Sigma_k\mathrm{T}((t+n)^c). \tag{14}$$

Proof: We give the proof for $k = 1$. Consider a Σ_2-machine running in time t on an input x of length n. Its acceptance criterion can be written as

$$(\exists\, y_1 \in \{0,1\}^t)\ \underbrace{(\forall\, y_2 \in \{0,1\}^t)\, R(x, y_1, y_2)}_{(*)}, \tag{15}$$

where R denotes a predicate computable in deterministic linear time. Part $(*)$ of (15) defines a conondeterministic computation on input x and y_1. The running time is $O(t) = O(t+n)$, which is linear in the length of the combined input $\langle x, y_1 \rangle$. Therefore, our hypothesis implies that we can transform $(*)$ into a nondeterministic computation on input x and y_1 taking time $O((t+n)^c)$. All together, (15) then describes a nondeterministic computation on input x of time complexity $O(t + (t+n)^c) = O((t+n)^c)$. $\qquad\square$

We point out that the term n in the right-hand side of (14) is necessary, i.e., for sublinear t we can only guarantee a running time of n^c rather than t^c. Although for sublinear t every computation path of $(*)$ in the proof of Proposition 3.6 can only access t bits of the input x, which bits are accessed depends on the computation path, so all of x needs to be input to the conondeterministic computation $(*)$. For our applications, a situation in which we apply Proposition 3.6 with sublinear t is suboptimal. The alternations we spent to reduce the running time below linear are wasted since we would have achieved the same running time after the application of Proposition 3.6 if we had not spent those alternations. For running times that are at least linear, the hypothesis allows us to eliminate one alternation at the cost of raising the running time to the power c.

3.4 A Concrete Example

In this section we give a concrete instantiation of the paradigm of indirect diagonalization we presented at the beginning of Section 3. So far we have seen techniques to trade alternations for time and to trade time for alternations. What remains is to combine them in the right way so as to reduce the resources enough and get a contradiction with a direct diagonalization result.

Our example is due to Kannan [Kan84], who used the paradigm avant la lettre to investigate the relationship between deterministic time $O(t)$ and nondeterministic time $O(t)$ for various time bounds t. In the case of linear time bounds he showed that $\mathrm{NT}(n) \nsubseteq \mathrm{DTS}(n, n^e)$ for every real $e < 1$. We cast his argument in our indirect diagonalization paradigm and slowly go through the steps, providing more details than we will in our later applications.

Step 1 We assume by way of contradiction that

$$\mathrm{NT}(n) \subseteq \mathrm{DTS}(n, n^e). \tag{16}$$

Step 2 Consider the class $\mathrm{DTS}(t, t^e)$ for some polynomial $t \geq n$ to be determined. By first speeding up as in (8) and then removing an alternation as in (14) with $k = 1$ and $c = 1$, we obtain the following unlikely inclusion:

$$\mathrm{DTS}(t, t^e) \subseteq \Sigma_2\mathrm{T}(t^{(1+e)/2}) \subseteq \mathrm{NT}(t^{(1+e)/2}). \tag{17}$$

22

We can apply (14) with $k = 1$ and $c = 1$ because the hypothesis implies that $\mathrm{NT}(n) \subseteq \mathrm{DT}(n)$; the application is valid provided that $t^{(1+e)/2}(n) \geq n$.

Step 3 We can pad the hypothesis (16) to time t as given by Proposition 2.2. Combined with the closure of DTS under complementation and with (17) we obtain

$$\mathrm{NT}(t) \subseteq \mathrm{DTS}(t, t^e) = \mathrm{coDTS}(t, t^e) \subseteq \mathrm{coNT}(t^{(1+e)/2}).$$

This contradicts Lemma 3.1 as long as $1 > (1 + e)/2$, i.e., for $e < 1$.

Setting $t(n) = n^2$ satisfies all the conditions we needed. We conclude that $\mathrm{NT}(n) \not\subseteq \mathrm{DTS}(n, n^e)$ for reals $e < 1$.

In step 3 we used the closure under complementation of deterministic computations. We made that step explicit in the sequence of inclusions; from now on we will do it implicitly.

Separations of the form $\mathrm{NT}(n) \not\subseteq \mathrm{DTS}(n, n^e)$ are not strong enough for Corollary 2.4 to give us lower bounds for satisfiability. Kannan used the separations to derive other results about the relationship between $\mathrm{DT}(t)$ and $\mathrm{NT}(t)$ for nonlinear t. We do not state these results. Instead, we move on and see how we can use the same paradigm to derive lower bounds for $\mathrm{NT}(n)$ that are strong enough to imply lower bounds for satisfiability.

4 Deterministic Algorithms

In this section we discuss lower bounds on deterministic machines. We first derive the results for nondeterministic linear time, implying lower bounds for satisfiability, and then cover closely related classes and corresponding problems.

4.1 Satisfiability

Our goal is to prove statements of the form

$$\mathrm{NT}(n) \not\subseteq \mathrm{coNT}(n^c) \cap \mathrm{DTS}(n^d, n^e) \tag{18}$$

for reals $d \geq c > 1$ and $e > 0$. Following the paradigm of indirect diagonalization from Section 3, we assume the opposite, i.e., that

$$\mathrm{NT}(n) \subseteq \mathrm{coNT}(n^c) \cap \mathrm{DTS}(n^d, n^e), \tag{19}$$

and derive a contradiction. Note that the hypothesis really states two inclusions. We refer to them as the first and the second hypothesis.

Fortnow's argument relies on Nepomnjascii's Theorem (Lemma 3.4). His original proof follows the more general scheme outlined at the beginning of Section 3 – he shows that the hypotheses lead to speedups within some higher level of the polynomial-time hierarchy. We recast his argument to fit the scheme of deriving speedups within the first level. It goes as follows: use the second hypothesis to put $\mathrm{NT}(t)$ for some super-linear t in $\mathrm{DTS}(n^{O(1)}, n^{1-\epsilon})$ for some positive real ϵ, then apply Nepomnjascii's Theorem to obtain a simulation somewhere in the linear-time hierarchy, and finally use the first hypothesis to eliminate all the induced alternations and return to the first level of the polynomial-time hierarchy. Eliminating alternations costs raising the running time to the

power c per alternation. For small enough c this process results in a net speedup of computations in the first level of the polynomial-time hierarchy, which is impossible in general.

In order to put $\mathrm{NT}(t)$ in $\mathrm{DTS}(n^{O(1)}, n^{1-\epsilon})$ by padding the second hypothesis, we need $t^e \leq n^{1-\epsilon}$. Since we want t to be super-linear, we set $t = n^{(1-\epsilon)/e}$ and require $e < 1 - \epsilon$. We have

$$
\begin{array}{rll}
\mathrm{NT}(t) & \subseteq & \mathrm{DTS}(t^d, t^e) \qquad\qquad\quad\;\; [\text{ hypothesis 2 }] \\
& \subseteq & \mathrm{coDTS}(n^{d(1-\epsilon)/e}, n^{1-\epsilon}) \quad [\text{ simplification }] \\
& \subseteq & \Pi_k \mathrm{T}(n) \qquad\qquad\qquad\;\;\; [\text{ Nepomnjascii's Theorem }] \\
& \subseteq & \mathrm{coNT}(n^{c^{k-1}}) \qquad\qquad\;\; [\text{ hypothesis 1 and Proposition 3.6 }]
\end{array}
$$

where k depends on d, e, and ϵ. No matter what k is, there are values of $c > 1$ such that $c^{k-1} < (1-\epsilon)/e$. For such values we obtain a contradiction with Lemma 3.1. We conclude that for every real d and $e < 1$ there exists a real $c > 1$ such that (18) holds. In particular, for every real $e < 1$ we have the time-space lower bound $\mathrm{NT}(n) \not\subseteq \mathrm{DTS}(n^{1+o(1)}, n^e)$.

The application of Nepomnjascii's Theorem is equivalent to multiple applications of the basic divide-and-conquer-strategy (2). The result of Kannan's we discussed in Section 3.4 involved one application in a setting with $d = 1$. Lipton and Viglas analyzed what a single application gives for larger values of d. The optimal form for one application is given by (8). Thus, for sufficiently large polynomials t we can derive

$$
\begin{array}{rll}
\mathrm{NT}(t) & \subseteq & \mathrm{DTS}(t^d, t^e) \qquad\quad [\text{ hypothesis 2 }] \\
& \subseteq & \Pi_2 \mathrm{T}(t^{(d+e)/2}) \qquad [\text{ (8) }] \\
& \subseteq & \mathrm{coNT}(t^{c(d+e)/2}) \quad [\text{ hypothesis 1 and Proposition 3.6 }]
\end{array}
$$

We obtain a contradiction with Lemma 3.1 as long as $c(d+e) < 2$. We conclude that for all reals c and d such that $cd < 2$, there exists a positive real e such that (18) holds. In particular, we obtain the following time-space lower bound for subpolynomial space: $\mathrm{NT}(n) \not\subseteq \mathrm{DTs}(n^{\sqrt{2}-o(1)})$.

At first sight, it may seem that one can easily improve the time lower bound for subpolynomial-space algorithms from $n^{\sqrt{2}-o(1)}$ to $n^{2-o(1)}$ by applying the optimal divide-and-conquer strategy recursively as in (12). In fact, there have been some unsubstantiated claims to that effect in the literature. The – fallacious – reasoning is the following. For subpolynomial space bounds, (8) allows us to realize a square-root speedup at the cost of introducing two alternations. We can apply this strategy ℓ times recursively and exploit the closure under complementation of deterministic computations to align adjacent quantifiers, as we did at the end of Section 3.2. This way, $\ell+1$ quantifiers are sufficient to reduce a $\mathrm{DTs}(t)$-computation to a $\mathrm{DTs}(t^{1/2^\ell + o(1)})$-computation. Eliminating all but one quantifier block using Proposition 3.6 leads to the conclusion that for sufficiently large polynomial t, $\mathrm{NT}(t) \subseteq \mathrm{coNT}(t^{c^\ell d/2^\ell + o(1)})$. For $c = d$ we obtain a contradiction with Lemma 3.1 as long as $d < 2$ and ℓ is sufficiently large. However, this reasoning is flawed because the input to the recursive levels does not only consist of the original input x of length n but also of the configurations and blocks guessed at previous levels. As a function of that input size, the running time of the computation on which we apply Proposition 3.6 for the complementation becomes sublinear. In that case the cost of alternation elimination becomes more than a power c as a function of the running time. As a result, the net effect of introducing one more level of recursion and eliminating it again is not as good as a factor of $c/2$ in the exponent of the running time. This issue of input size is a recurring theme and a bottleneck in all of the subsequent arguments. It is not obvious whether a quadratic time lower bound for subpolynomial-space deterministic algorithms can be

reached using the ingredients from Section 3 alone. It does seem like a quadratic bound is the best we can hope for given the optimality of (8).

Fortnow and Van Melkebeek analyzed how to pick the block sizes in ℓ recursive applications of the divide-and-conquer strategy so as to optimize the speedup of deterministic space-bounded computations on nondeterministic machines that results after eliminating all the alternations using Proposition 3.6. We give their analysis in the case of subpolynomial space bounds.

Lemma 4.1 (Fortnow-Van Melkebeek [FvM00, FLvMV05]) *If*

$$\mathrm{NT}(n) \subseteq \mathrm{coNT}(n^c)$$

for some real c, then for every nonnegative integer ℓ and for every sufficiently large polynomial t,

$$\mathrm{DTs}(t) \subseteq \mathrm{NT}(t^{\alpha_\ell + o(1)}),$$

where

$$
\begin{aligned}
\alpha_0 &= 1 \\
\alpha_{\ell+1} &= c\alpha_\ell/(1 + \alpha_\ell).
\end{aligned}
\tag{20}
$$

Proof: We give a proof by induction. The base case $\ell = 0$ holds trivially. For the induction step $\ell \to \ell + 1$ we have for every real $\alpha \in (0,1)$ and sufficiently large polynomial t

$$
\begin{aligned}
\mathrm{DTs}(t) &\subseteq \exists^{t^{\alpha+o(1)}} \forall^{\log t} \underbrace{\mathrm{DTs}(t^{1-\alpha})}_{(*)} & [\text{ (7) with } b = t^\alpha] \\
&\subseteq \exists^{t^{\alpha+o(1)}} \underbrace{\forall^{\log t} \mathrm{coNT}(t^{(1-\alpha)\alpha_\ell + o(1)})}_{(**)} & [\text{ induction hypothesis }] \\
&\subseteq \exists^{t^{\alpha+o(1)}} \underbrace{\mathrm{NT}((t^{\alpha+o(1)} + t^{(1-\alpha)\alpha_\ell + o(1)})^c)}_{(***)} & [\text{ hypothesis 1 }] \\
&\subseteq \mathrm{NT}((t^\alpha + t^{(1-\alpha)\alpha_\ell})^{c+o(1)}) & [\text{ simplification using } c \geq 1]
\end{aligned}
\tag{21}
$$

Note that the input to $(*)$ is only of length $n + t^{o(1)}$ since the computation only needs access to the original input x and two configurations. For the induction hypothesis to apply to $(*)$, $t^{1-\alpha}$ has to satisfy the "sufficiently large polynomial" condition at level ℓ, which will be the case for yet larger polynomials t (depending on the choice of α). The input to the computation $(**)$ consists of the original input x and the guess bits of the first existential phase, so it is of length $n + t^{\alpha+o(1)}$. This is why, for sufficiently large t, there is no extra term in the application of the induction hypothesis but there is an extra term of t^α in the application of the hypothesis of the lemma.

The final running time is optimized up to subpolynomial factors by equating the exponents of both terms, i.e., by setting $\alpha = (1 - \alpha)\alpha_\ell$, which leads to the recurrence (20) for $\alpha_{\ell+1}$. $\qquad\square$

We will encounter a fair number of recurrences of the type given by (20). The following facts are useful in studying their convergence behavior.

Proposition 4.2 *Let a, b, and ξ_0 be positive reals. The sequence defined by*

$$\xi_{\ell+1} = a\xi_\ell/(1 + b\xi_\ell)$$

for nonnegative integers ℓ converges monotonically, namely to 0 if $a \leq 1$ and to $(a-1)/b$ if $a \geq 1$. The sequence is decreasing iff $\xi_0 > (a-1)/b$.

25

Proof: Since the transformation $\xi \to a\xi/(1 + b\xi)$ on the reals is increasing, the sequence ξ_ℓ is monotone. Combined with the continuity of the transformation, this means the sequence has to converge to a fixed point of the function. The fixed points are 0 and $(a - 1)/b$. The first one is attractive iff $a \leq 1$. The sequence if decreasing iff ξ_0 is larger than the limit point. $\qquad\square$

Note that the mere monotonicity of the sequence implies that the smallest real the sequence can get close to is the minimum of the start value ξ_0 and the limit value ξ_∞. That value can be written as $\min(\xi_0, \max(0, (a-1)/b))$. For the particular sequence (20) Proposition 4.2 tells us that the sequence converges to $\alpha_\infty = c - 1$, and decreases monotonically for $c < 2$.

Given the speedup lemma, we can finish the argument in the same way as before. Starting from hypothesis (19) with $e = o(1)$, we have for sufficiently large polynomials t that

$$
\begin{aligned}
\mathrm{NT}(t) &\subseteq \mathrm{DTs}(t^d) && [\text{ hypothesis 2 }] \\
&\subseteq \mathrm{coNT}(t^{d\alpha_\ell}) && [\text{ Lemma 4.1 }]
\end{aligned}
\tag{22}
$$

which contradicts Lemma 3.1 if $d\alpha_\ell < 1$. By the monotonicity of the sequence α_ℓ, we only need to check $\ell = 0$ and $\ell \to \infty$. The first case only leads to a contradiction for $d < 1$. The second case contradicts Lemma 3.1 as long as $d\alpha_\infty = (c - 1)d < 1$. Thus, we have derived the first part of the Master Theorem for deterministic algorithms (Theorem 1.3) in the case of subpolynomial space bounds. In particular, we can conclude that $\mathrm{NT}(n) \not\subseteq \mathrm{DTs}(n^{\phi - o(1)})$, since $d(d - 1) = 1$ defines the golden ratio ϕ. A more careful analysis of our argument proves the first part of Theorem 1.3 in full generality. This result also captures Fortnow's $\mathrm{NT}(n) \not\subseteq \mathrm{coNT}(n^{1+o(1)}) \cap \mathrm{DTS}(n^{O(1)}, n^{1-\epsilon})$, as the condition $(c - 1)d < 1$ allows us to let d grow unboundedly for $c = 1 + o(1)$, in which case we can let e grow to 1.

Before moving on, let us make a few observations about the proof of Lemma 4.1. Recall from our discussion in Section 3.3 that it never makes sense for us to eliminate an alternation using Proposition 3.6 in a situation where the running time is sublinear. The proof of Lemma 4.1 exactly balances the input to $(**)$ and the running time. This suggests that if we somehow did not have to take the entire input into account, we could do better. In fact, the proof makes our earlier point about input size issues very explicit. If we could ignore the $t^{\alpha + o(1)}$ guess bits as input to $(**)$, we could set $\alpha_{\ell+1}$ as the solution α to $\alpha = (1 - \alpha)\alpha_\ell c$, which would result in a limit of $1 - 1/c$ and a contradiction with Lemma 3.1 for $(c - 1)d < c$. The latter condition reduces to $d < 2$ for $c = d$.

For future reference we also point out the following failed approach for improving Lemma 4.1. Consider the second to last line of (21) in the proof of Lemma 4.1. We could first apply the second hypothesis to $(***)$, transforming $(***)$ into a DTs-computation at the cost of raising the running time to the power d, and then apply the induction hypothesis to speed up the DTs-computation on a nondeterministic machine. The latter makes sense since the induction hypothesis gives us the best way we have found so far to speed up DTs-computations on nondeterministic machines. The resulting computation is of the same form as $(***)$ but hopefully has a smaller exponent for the second term of the running time. That would push the optimum for α to smaller values and thereby achieve a smaller final running time. Unfortunately, this application of the induction hypothesis is invalid. The induction hypothesis only applies when the running time of the computation is a sufficiently large polynomial in the input length. The largeness condition becomes more stringent when ℓ grows; a closer analysis shows that the polynomial needs to be of degree $\Omega(c^\ell)$ at level ℓ. The running time of $(***)$ is a relatively small polynomial in t^α. Since the t^α guess bits are part of the input to $(***)$, this means that, although t is a large polynomial, the running time of

(∗∗∗) is only a polynomial of relatively small degree in the length of the input to that part of the computation. Thus, we cannot apply the induction hypothesis as we envisioned. The underlying idea will be useful in another context, though, namely for the lower bounds on nondeterministic machines in Section 5.

In the current context, we can improve the indirect diagonalization strategy in a different way. Lemma 4.1 optimizes the following alternation trading process: introduce alternations to speed up the computation by applying the divide-and-conquer strategy (7) ℓ times recursively; next eliminate the induced alternations one by one by successively applying the first hypothesis and Proposition 3.6 to the last quantifier block and then merging that block with the previous block. The latter phase of alternation elimination is rather blunt. It it oblivious to the special structure of the computation (11). We can take more of the structure into account by extending our view during the alternation elimination phase from just the last block to the last two blocks. In other words, we iteratively complement the last two quantifier blocks and merge the second-to-last quantifier with the previous one.

The Π_2-computations we need to complement during the process all have the property that the final nondeterministic stage only needs access to a small part of the input to the Π_2-computation. The complementary property holds for Σ_2. Consider the case of odd k in (11). Recall that each of the universal quantifier blocks in (11) are of the form $\forall^{\log b_i}\forall^{b_{i+1}s}$. They get as input the global input x and the configurations guessed during stage i. Of those configurations only two are passed on from the $\forall^{\log b_i}$-part to the $\forall^{b_{i+1}s}$-part and thus to the subsequent $\exists^{\log b_{i+1}}$-part. In particular, the above property holds for the first Π_2-computation we need to complement, the one at the far right end of (11). Moreover, if we replace that Π_2-computation by a generic Σ_2-computation and merge the now-aligned existential quantifiers, the resulting Σ_2-computation at the end again has the above property. Thus, the above property is maintained during the process of successive complementations within the second level.

Using our hypotheses, we can efficiently complement Π_2-computations by first turning them into space-bounded deterministic computations and then speeding those up on Σ_2-machines using (8). More precisely, we have that

$$\Pi_2\mathrm{T}(\tau) \subseteq \mathrm{coNT}((\tau + \mu)^c) \subseteq \mathrm{DTs}(((\tau + \mu)^c + \nu)^d) \subseteq \Sigma_2\mathrm{T}(((\tau + \mu)^c + \nu)^{d/2+o(1)}), \qquad (23)$$

where τ denotes a possibly sublinear running time, ν denotes the size of the input to the Π_2-computation, and μ the size of the part of that input that is input to the final nondeterministic computation of the Π_2-computation. For generic Π_2-computations, $\mu = \nu$ and it does not make sense for us to consider sublinear running times τ as it would mean that we wasted some alternations – see the discussion after Proposition 3.6 in Section 3.3. However, the Π_2-computations we need to complement have the property that $\mu \ll \nu$. In that case, it does make sense to consider sublinear τ. In fact, due to the binding nature of the input size issue, we can take advantage of sublinear τ. The following lemma shows how.

The lemma can be viewed as a counterpart to Lemma 4.1 but where we eliminate alternations by complementing computations within the second rather than the first level of the polynomial-time hierarchy. The lemma analyzes how to optimally select the parameters in the process of speeding up DTs-computations using (11) and subsequently removing alternations by complementing within the second level so as to achieve the smallest running time for the final simulation. Note that the process ends at the second level of the polynomial-time hierarchy rather than the first one as in Lemma 4.1 since the smallest number of quantifier blocks we can reduce to by complementing

within the second level is two. For future use, we parameterize the lemma with the efficiency of the speedup of deterministic sublinear-space computations in the second level of the polynomial-time hierarchy. The simulation (8) corresponds to $\sigma = 1/2 + o(1)$ in the lemma.

Lemma 4.3 (follows from [Wil07b]) *If*

$$\mathrm{NT}(n) \subseteq \mathrm{coNT}(n^c) \cap \mathrm{DTs}(n^d)$$

for some reals c and d, and if

$$\mathrm{DTs}(t) \subseteq \Sigma_2\mathrm{T}(t^\sigma + n) \tag{24}$$

for some real $\sigma \leq 1/d$ and all functions t, then for every nonnegative integer ℓ and for every sufficiently large polynomial t,

$$\mathrm{DTs}(t) \subseteq \Sigma_2\mathrm{T}(t^{\beta_\ell + o(1)}),$$

where

$$\begin{aligned} \beta_0 &= \sigma \\ \beta_{\ell+1} &= cd\sigma\beta_\ell/(1 + cd\sigma\beta_\ell). \end{aligned} \tag{25}$$

Proof: The proof is again by induction. The base case follows from (24). For the induction step $\ell \to \ell + 1$ we have for every real $\beta \in (0,1)$ and sufficiently large polynomial t

$$
\begin{aligned}
\mathrm{DTs}(t) \;\subseteq\; & \exists^{t^{\beta+o(1)}}\forall^{\log t}\underbrace{\mathrm{DTs}(t^{1-\beta})}_{(*)} && [\ (7) \text{ with } b = t^\beta\ ] \\
\subseteq\; & \exists^{t^{\beta+o(1)}}\forall^{\log t}\underbrace{\Pi_2\mathrm{T}(t^{(1-\beta)\beta_\ell+o(1)})}_{(**)} && [\ \text{induction hypothesis}\ ] \\
\subseteq\; & \exists^{t^{\beta+o(1)}}\underbrace{\forall^{\log t}\mathrm{coNT}(t^{(1-\beta)\beta_\ell c+o(1)})}_{(***)} && [\ \text{hypothesis 1 and Prop. 3.6}\ ] \\
\subseteq\; & \exists^{t^{\beta+o(1)}}\underbrace{\mathrm{DTs}((t^{\beta+o(1)} + t^{(1-\beta)\beta_\ell c+o(1)})^d)}_{(****)} && [\ \text{hypothesis 2}\ ] \\
\subseteq\; & \exists^{t^{\beta+o(1)}}\Sigma_2\mathrm{T}((t^{\beta+o(1)} + t^{(1-\beta)\beta_\ell c+o(1)})^{d\sigma} + t^{\beta+o(1)}) && [\ (24)\ ] \\
\subseteq\; & \Sigma_2\mathrm{T}(t^{(1-\beta)\beta_\ell cd\sigma+o(1)} + t^{\beta+o(1)}) && [\ \text{simplification using } d\sigma \leq 1\ ]
\end{aligned}
$$
(26)

Note that the input to $(*)$ and $(**)$ is only of length $n + t^{o(1)}$, which is less than $t^{1-\beta}$ or even $t^{(1-\beta)\beta_\ell}$ for sufficiently large polynomials t. The input size to $(***)$ and $(****)$ equals $n + t^{\beta+o(1)} = O(t^{\beta+o(1)})$ for sufficiently large polynomials t.

The final running time is optimized up to subpolynomial factors by setting $\beta = (1-\beta)\beta_\ell cd\sigma$, which leads to the recurrence (25) for $\beta_{\ell+1}$. □

Let us connect the proof of the lemma with the discussion before. The net effect from the second to the fifth line in (26) is to transform the Π_2-computation described by $\forall^{\log t}(**)$ on the second line into an equivalent Σ_2-computation on the fifth line. The computation $\forall^{\log t}(**)$ is exactly of the type we described before the lemma: it has a running time $\tau = t^{(1-\beta)\beta_\ell+o(1)}$ that is sublinear in the size of its input, $\nu = n + t^{\beta+o(1)}$, but the effective input to the final nondeterministic phase is only $\mu = \tau \ll \nu$. Note also that $\mu = \tau$ means that the input size and running time are balanced in the application of Proposition 3.6, just as that was the case in the proof of Lemma 4.1.

We can wrap up the argument in a similar way as before. For sufficiently large polynomials t we have

$$\begin{aligned}
\mathrm{NT}(t) &\subseteq \mathrm{DTs}(t^d) & [\text{ hypothesis 2 }] \\
&\subseteq \Pi_2\mathrm{T}(t^{d\beta_\ell+o(1)}) & [\text{ Lemma 4.3 }] \\
&\subseteq \mathrm{coNT}(t^{d\beta_\ell c+o(1)}) & [\text{ hypothesis 1 }]
\end{aligned}$$

which contradicts Lemma 3.1 if $cd\beta_\ell < 1$ for some nonnegative integer ℓ. By Proposition 4.2, the only values we need to check are $\ell = 0$ and $\ell \to \infty$. The former leads to the simple condition that $cd\sigma < 1$ (since we do not need the condition $d\sigma \leq 1$ of Lemma 4.3 at the start). For the latter case, we need the additional condition that $d\sigma \leq 1$. By Proposition 4.2, $\beta_\infty = 0$ if $cd\sigma < 1$ and $\beta_\infty = 1 - 1/(cd\sigma)$ otherwise. The second possibility results in the conditions $d\sigma \leq 1$ and $cd < 1 + 1/\sigma$. As we can assume $c \geq 1$, the condition $cd\sigma < 1$ implies $d\sigma \leq 1$ and $cd < 1 + 1/\sigma$, so we can forget about $cd\sigma < 1$. In summary, we obtain a contradiction if $d\sigma \leq 1$ and $cd < 1 + 1/\sigma$.

Let us now see what we get when we plug in the general speedup (8) into Lemma 4.3, i.e., if we set $\sigma = 1/2+o(1)$ in (24). We obtain a contradiction if $d \leq 2$ and $cd < 3$. In particular, we obtain a time lower bound of $n^{\sqrt{3}-o(1)}$ for every subpolynomial-space algorithm solving satisfiability, which beats the golden ratio result.

We can do even better. Williams showed that the speedup (8), which holds unconditionally, can be improved under the hypothesis that $\mathrm{NT}(n) \subseteq \mathrm{DTs}(n^d)$ for values of $d < 2$: instead of the square-root speedup of (8) we can realize a $(d/(d-1))$st root, which is better for $d < 2$ since $d/(d-1) > 2$.

Lemma 4.4 (Williams [Wil05]) *If*

$$\mathrm{NT}(n) \subseteq \mathrm{DTs}(n^d)$$

for some real d, then for every time bound t

$$\mathrm{DTs}(t) \subseteq \Sigma_2\mathrm{T}(t^{(d-1)/d+o(1)} + n).$$

Proof: We prove by induction that

$$\mathrm{DTs}(t) \subseteq \Sigma_2\mathrm{T}(t^{\gamma_\ell+o(1)} + n)$$

for every nonnegative integer ℓ, where

$$\begin{aligned}
\gamma_0 &= 1/2 \\
\gamma_{\ell+1} &= d\gamma_\ell/(1 + d\gamma_\ell).
\end{aligned} \tag{27}$$

The proof is by induction on ℓ. The base case $\ell = 0$ holds because of (8). By Proposition 4.2 the sequence defined by (27) is nondecreasing for $d \geq 2$, so the base case trivially implies the remaining cases for $d \geq 2$. In the rest of the proof we only consider $d < 2$. For the induction step $\ell \to \ell + 1$ we have for every real $\gamma \in (0, 1)$

$$\begin{aligned}
\mathrm{DTs}(t) &\subseteq \exists^{t^{\gamma+o(1)}} \underbrace{\forall^{\log t}\mathrm{DTs}(t^{1-\gamma})}_{(*)} & [\text{ (7) with } b = t^\gamma] \\
&\subseteq \exists^{t^{\gamma+o(1)}} \underbrace{\mathrm{DTs}((n + t^{\gamma+o(1)} + t^{1-\gamma})^d)}_{(**)} & [\text{ hypothesis of the lemma }] \\
&\subseteq \exists^{t^{\gamma+o(1)}} \Sigma_2\mathrm{T}((n + t^\gamma + t^{1-\gamma})^{d\gamma_\ell+o(1)} + n + t^{\gamma+o(1)}) & [\text{ induction hypothesis }] \\
&\subseteq \Sigma_2\mathrm{T}(t^{\gamma+o(1)} + t^{(1-\gamma)d\gamma_\ell+o(1)} + n) & [\text{ simplification using } d\gamma_\ell < 1]
\end{aligned}$$

29

Note that $(*)$ represents a cononondeterministic computation on an input of size $n+t^{\gamma+o(1)}$ that runs in time $t^{1-\gamma}$, and that $(**)$ has the same input.

The final running time is optimized up to subpolynomial factors by equating the exponents of the two terms involving γ, i.e., by setting $\gamma = (1-\gamma)d\gamma_\ell$, which leads to the recurrence (27) for $\gamma_{\ell+1}$.

The sequence given by (27) decreases monotonically to $\gamma_\infty = 1 - 1/d$. It follows that the condition $d\gamma_\ell < 1$ is met for every ℓ. By exploiting the uniformity of the construction we can let ℓ grow slowly with the input size n at the cost of an additional small factor in the running time. Since t is at least logarithmic, we can absorb the latter factor in a term of the form $t^{o(1)}$, which gives the result. $\qquad\square$

By virtue of Lemma 4.4, we can now apply Lemma 4.3 and the subsequent analysis with $\sigma = (d-1)/d + o(1)$. We obtain a contradiction if $d \leq 2$ and $cd < 1 + d/(d-1)$. The condition $c(d-1) < 1$ together with $c \leq d$ implies that $(c-1)d < 1$ so it does not lead to new results. The latter condition is equivalent to $cd(d-1) - 2d + 1 < 0$ and coincides with the second condition in the Master Theorem for deterministic algorithms (Theorem 1.3). Note that we do not have to worry about the fact that we have the additional condition $d \leq 2$ here since we know that the first condition in the Master Theorem is binding for $d > 2$. In particular, we obtain a time lower bound of $n^{2\cos(\pi/7)-o(1)}$ for every subpolynomial-space algorithm solving satisfiability. A more careful analysis finishes the proof of Theorem 1.3 in full force. This marks the end of the story on lower bounds for satisfiability on deterministic machines so far...

4.2   Related Problems

We now discuss how the arguments of the previous section can be extended to $\Sigma_k\text{SAT}$ for every integer $k > 1$ and to the problem of counting the number of satisfying assignments modulo a fixed integer. We point out that Fortnow and Van Melkebeek [FvM00, FLvMV05] used the same ideas to obtain lower bounds for classes of the form $\text{NTS}(n, s)$ for sublinear s. We will not cover those results here as there are no natural computational problems known that correspond exactly to those classes.

We first consider $\Sigma_k\text{SAT}$. We would like to rule out hypotheses of the form

$$\Sigma_k\text{T}(n) \subseteq \Pi_k\text{T}(n^c) \cap \text{DTS}(n^d, n^e)$$

for certain values of c, d, and $e > 0$. We start with some simple observations. First, Lemma 3.1 shows that $c \geq 1$ and therefore also $d \geq 1$. The second hypothesis combined with (13) implies

$$\Sigma_k\text{T}(n) \subseteq \text{DTS}(n^d, n^e) \subseteq \Pi_k\text{T}(n^{(d+(k-1)e)/k}),$$

so we can assume without loss of generality that $c \leq (d + (k-1)e)/k$. Combined with Lemma 3.1 we have ruled out values such that $d + (k-1)e < k$. In the case of subpolynomial space bounds, which we focus on next, this means we can assume that $1 \leq c \leq d/k$, which implies $d \geq k$.

The statement and proof of Lemma 4.1 carry over verbatim if we replace the machine type N by Σ_k. As before, we can rule out values for which $(c-1)d < 1$. Taking into account that $c \leq d/k$, this leads to the time-space lower bound $\Sigma_k\text{T}(n) \not\subseteq \text{DTs}(n^d)$ for every real d such that $d(d-k) < k$. Note that the solution to $d(d-k) = k$ lies somewhere between k and $k+1$ and differs from $k+1$ by $O(1/k)$.

Lemma 4.3 also remains valid modulo the substitution of N by Σ_k, of Σ_2 by Σ_{k+1}, and of Π_2 by Π_{k+1}. Under the hypothesis that $\Sigma_k T(n) \subseteq DTs(n^d)$ for some real d, the generalization of Lemma 4.4 shows that for every time bound t, $DTs(t) \subseteq \Sigma_{k+1} T(t^{(d-k)/d+o(1)} + n)$. We obtain a contradiction with Lemma 3.1 if $d \leq k+1$ and $cd < 1 + d/(d-k)$. The latter condition is equivalent to $cd(d-k) - 2d + k < 0$ and is weaker than the condition $(c-1)d < 1$ iff $d < k+1$. As for a pure time lower bound for subpolynomial space algorithms, the solution to $d^2(d-k) - 2d + k = 0$ also lies somewhere between k and $k+1$ and converges to $k+1$ for large k, but does so more quickly than the solution to $d(d-k) = k$.

A more careful analysis leads to the following generalization of the Master Theorem for deterministic machines.

Theorem 4.5 (follows from [FvM00, FLvMV05, Wil07b]) *For every positive integer k and for all reals c and d such that $(c-1)d < 1$ or $cd(d-k) - 2d + k < 0$, there exists a positive real e such that $\Sigma_k SAT$ cannot be solved by both*

(i) a Π_k-machine with random access that runs in time n^c and

(ii) a deterministic random-access machine that runs in time n^d and space n^e.

Moreover, the constant e approaches 1 from below when c approaches 1 from above and d is fixed.

Recall that a machine of type (ii) implies the existence of a machine of type (i) with $c = (d + (k-1)e)/k$.

Williams considered the problem of counting the number of satisfying assignments modulo some integer $m > 1$. The language $Mod_m SAT$ consists of all Boolean formulas for which the number of satisfying assignments is divisible by m. By the first part of Toda's Theorem [Tod91], $Mod_m SAT$ is hard for the polynomial-time hierarchy under randomized reductions but it is consistent with current knowledge that the deterministic complexities of $Mod_m SAT$ and of $\Sigma_k SAT$ would be incomparable for all integers $m > 1$ and $k > 0$. In the other direction, we definitely do not expect lower bound statements for satisfiability to immediately imply similar statements for $Mod_m SAT$ in general. Nevertheless, Williams showed that the lower bounds from the previous section do carry over.

We can define a model of computation that has the same tight connection to $Mod_m SAT$ as nondeterministic computations have to satisfiability. The class $Mod_m TS(t, s)$ denotes all languages L for with there exists a nondeterministic machine M that runs in time $O(t)$ and space $O(s)$ such that a string x belongs to L iff the number of accepting computations of M on input x is divisible by m. The classes $Mod_m T(t)$ and $Mod_m Ts(t)$ are defined analogously. Lemma 2.3 holds for $Mod_m T(n)$ and $Mod_m SAT$ instead of $\Sigma_k T(n)$ and $\Sigma_k SAT$, as does Corollary 2.4.

Williams argued that all of the lower bound proofs in Section 4.1 carry through if we replace nondeterministic computations by Mod_p-computations and conondeterministic computations by Mod_q-computations where p and q are any two distinct primes. He shows that hypotheses like

$$\begin{cases} Mod_p T(n) \subseteq Mod_q T(n^c) \cap DTS(n^d, n^e) \\ Mod_q T(n) \subseteq Mod_p T(n^c) \cap DTS(n^d, n^e) \end{cases}$$

lead to a contradiction for the same settings of the reals c, d, and e as in the proof of Theorem 1.3. The critical ingredient in the translation argument is the equivalent of (7). It leads to the following time-space lower bound.

Theorem 4.6 (Williams [Wil07b]) *For every real $d < 2\cos(\pi/7)$ there exists a positive real e such that for every prime p except possibly one, $\mathrm{Mod}_p\mathrm{SAT}$ cannot be solved by a deterministic random-access machine that runs in time n^d and space n^e. Moreover, the constant e approaches 1 from below when d approaches 1 from above.*

Since Mod_p-computations trivially reduce to Mod_m-computations for every positive integer m that is a multiple of p, we can conclude the same lower bounds as in Theorem 4.6 for every integer $m > 1$ except possibly the powers of a single prime.

5   Nondeterministic Algorithms

In this section we discuss how the lower bound arguments from Section 4.1 for deterministic machines can be adapted to conondeterministic machines, resulting in the lower bounds for tautologies on nondeterministic machines as stated in Theorem 1.5.

Following the paradigm of indirect diagonalization, we start from a hypothesis of the form

$$\mathrm{coNT}(n) \subseteq \mathrm{NT}(n^c) \cap \mathrm{NTS}(n^d, n^e) \tag{28}$$

for certain reals c, d, and e. We aim to derive a contradiction with a direct diagonalization result like Lemma 3.1 by speeding up nondeterministic space-bounded computations with more alternations and eliminating the extra alternations using Proposition 3.6.

For starters, Lemma 3.1 immediately implies that $c \geq 1$ and that $d \geq 1$. We can also assume without loss of generality that $c \leq d$.

Speeding up nondeterministic space-bounded computations takes more alternations than deterministic ones. Recall that the closure under complementation of deterministic computations allowed us to realize the same speedup (12) in the deterministic case as (5) in the nondeterministic case with roughly only half the number of alternations. Each alternation we introduce needs to be eliminated later, which we can do using Proposition 3.6 at a cost of raising the running time to the power c. A need for more alternations for the same speedup results in weaker lower bounds in the end.

In the range of space bounds close to linear, the effect is not very noticeable – it is hidden in the statement of Theorems 1.3 and 1.5. For $e < 1$ and any positive real $\epsilon < 1 - e$ the argument of Section 4.1 carries through almost verbatim. Setting $t = n^{(1-\epsilon)/e}$ we have

$$\mathrm{NT}(t) \subseteq \mathrm{coNTS}(t^d, t^e) \subseteq \mathrm{coNTS}(n^{d(1-\epsilon)/e}, n^{1-\epsilon}) \subseteq \Pi_k\mathrm{T}(n) \subseteq \mathrm{coNT}(n^{c^{k-1}})$$

for some integer k depending on d and ϵ. The dependency is worse than in the deterministic case, but we can still conclude that for every real d and $e < 1$ there is a real $c > 1$ such that hypothesis (28) fails.

The deterioration in parameters becomes more tangible for smaller space bounds, in particular for subpolynomial ones. Fortnow and Van Melkebeek determined in general how to pick the block sizes in (5) such that if we subsequently eliminate all extra alternations one by one from the back to the front using Proposition 3.6, we end up with a speedup of nondeterministic space-bounded computations on nondeterministic machines that is the best one this process can give. We cover the case of subpolynomial space bounds.

Lemma 5.1 (Fortnow-Van Melkebeek [FvM00, FLvMV05]) *If*

$$\mathrm{NT}(n) \subseteq \mathrm{coNT}(n^c)$$

for some real c, then for every nonnegative integer ℓ and for every sufficiently large polynomial t,

$$\mathrm{NTs}(t) \subseteq \mathrm{NT}(t^{\delta_\ell + o(1)}),$$

where

$$
\begin{aligned}
\delta_0 &= 1 \\
\delta_{\ell+1} &= c^2 \delta_\ell / (1 + c\delta_\ell).
\end{aligned}
\tag{29}
$$

Proof: The proof is again one by induction with a trivial base case. For the induction step $\ell \to \ell+1$ we have for every real $\delta \in (0,1)$ and sufficiently large polynomial t

$$
\begin{aligned}
\mathrm{NTs}(t) \;\subseteq\;\; & \exists^{t^{\delta+o(1)}} \forall^{\log t} \underbrace{\mathrm{NTs}(t^{1-\delta})}_{(*)} & & [\,(3) \text{ with } b = t^\delta \,] \\
\subseteq\;\; & \exists^{t^{\delta+o(1)}} \forall^{\log t} \underbrace{\mathrm{NT}(t^{(1-\delta)\delta_\ell + o(1)})}_{(**)} & & [\,\text{induction hypothesis}\,] \\
\subseteq\;\; & \exists^{t^{\delta+o(1)}} \underbrace{\forall^{\log t} \mathrm{coNT}(t^{(1-\delta)\delta_\ell c + o(1)})}_{(***)} & & [\,\text{hypothesis of the lemma}\,] \\
\subseteq\;\; & \exists^{t^{\delta+o(1)}} \mathrm{NT}((t^{\delta+o(1)} + t^{(1-\delta)\delta_\ell c + o(1)})^c) & & [\,\text{hypothesis of the lemma}\,] \\
\subseteq\;\; & \mathrm{NT}((t^\delta + t^{(1-\delta)\delta_\ell c})^{c+o(1)}) & & [\,\text{simplification using } c \geq 1\,]
\end{aligned}
\tag{30}
$$

Note that the input to $(*)$ and to $(**)$ is only of length $n + t^{o(1)} = O(n)$; the input to the computation $(***)$ consists of the original input x and the guess bits of the first existential phase, so it is of length $n + t^{\delta+o(1)}$.

The final running time is optimized up to subpolynomial factors by equating δ and $(1-\delta)\delta_\ell c$, which results in the recurrence (29). $\qquad\square$

We conclude that for sufficiently large polynomials t

$$
\begin{aligned}
\mathrm{NT}(t) \;&\subseteq\; \mathrm{coNTs}(t^d) & & [\,\text{hypothesis 2}\,] \\
&\subseteq\; \mathrm{coNT}(t^{d\delta_\ell}) & & [\,\text{Lemma 5.1}\,]
\end{aligned}
\tag{31}
$$

which contradicts Lemma 3.1 if $d\delta_\ell < 1$. By Proposition 4.2 the sequence defined by (29) converges monotonically to $\delta_\infty = c - 1/c$, and is decreasing for $c < \phi$. By the monotonicity, we only need to check $\ell = 0$ and $\ell \to \infty$. The initial value only leads to a contradiction for $d < 1$; the limit leads to one for $(c^2 - 1)d < c$. By setting $c = d$, we obtain a time lower bound of $n^{\sqrt{2}-o(1)}$ for subpolynomial-space nondeterministic algorithms for $\mathrm{coNT}(n)$. Recall that the corresponding argument in the deterministic setting yields an exponent of ϕ rather than $\sqrt{2}$.

We point out that we could also have derived a contradiction for the same setting of the parameters from the hypothesis

$$\mathrm{NT}(n) \subseteq \mathrm{coNT}(n^c) \cap \mathrm{NTS}(n^d, n^e). \tag{32}$$

rather than from hypothesis (28). Lemma 5.1 only needs the first part of the hypothesis (28), which is equivalent to the first part of hypothesis (32). The argument similar to (31) now leads to a contradiction with the nondeterministic time hierarchy (Lemma 3.2) rather than with Lemma 3.1. As

a result, in the part of the Master Theorem for nondeterministic algorithms involving the condition $(c^2 - 1)d < 1$, the machine (ii) can also be conondeterministic rather than nondeterministic.

Lemma 5.1 optimizes an alternation trading process that acts in a rather myopic way during the alteration elimination phase. It only looks at the last quantifier block, complements it using Proposition 3.6, merges it with the previous quantifier block, and repeats. In doing so, it only relies on the first part of the hypothesis, the part involving time-efficient complementations of Σ_1-machines. In the deterministic setting of Section 4.1 we saw how we could do better by expanding our view during the alternation elimination phase from the last block to the last two blocks, and exploiting the second part of the hypothesis. We currently do not know of a fruitful analogue to that strategy in the nondeterministic setting. However, Diehl, Van Melkebeek, and Williams showed how we can use the second part of the hypothesis to our advantage while still just looking at the last quantifier block during each step of the alternation removal phase. They managed to get the idea to work which we briefly mentioned as a failed attempt in the deterministic setting, namely that of getting more mileage out of the induction hypothesis by applying it for a second time after making the computations space-bounded again using the second hypothesis. They achieve speedups of nondeterministic space-bounded computations on nondeterministic machines that are better than those provided by Lemma 5.1 for relatively small values of d.

Lemma 5.2 (Diehl-Van Melkebeek-Williams [DvMW07]) *If*

$$\mathrm{coNT}(n) \subseteq \mathrm{NT}(n^c) \cap \mathrm{NTs}(n^d)$$

for some reals c and d then for every nonnegative integer ℓ and for every sufficiently large polynomial t,

$$\mathrm{NTs}(t) \subseteq \mathrm{NT}(t^{\epsilon_\ell + o(1)}),$$

where

$$
\begin{aligned}
\epsilon_0 &= 1 \\
\epsilon_{\ell+1} &= cd\epsilon_\ell^2/(1 + d\epsilon_\ell^2).
\end{aligned}
\tag{33}
$$

Proof: As usual, the proof is by induction on ℓ and has a trivial base case. For the induction step $\ell \to \ell + 1$ we have for every real $\epsilon \in (0, 1)$ and sufficiently large polynomial t

$$
\begin{aligned}
\mathrm{NTs}(t) \quad &\subseteq \quad \exists^{t^{\epsilon + o(1)}} \forall^{\log t} \underbrace{\mathrm{NTs}(t^{1-\epsilon})}_{(*)} && [\ (3)\text{ with } b = t^\epsilon\ ] \\
&\subseteq \quad \exists^{t^{\epsilon + o(1)}} \forall^{\log t} \underbrace{\mathrm{NT}(t^{(1-\epsilon)\epsilon_\ell + o(1)})}_{(**)} && [\ \text{induction hypothesis}\ ] \\
&\subseteq \quad \exists^{t^{\epsilon + o(1)}} \forall^{\log t} \underbrace{\mathrm{coNTs}(t^{(1-\epsilon)\epsilon_\ell d + o(1)})}_{(***)} && [\ \text{hypothesis 2 of the lemma}\ ] \\
&\subseteq \quad \exists^{t^{\epsilon + o(1)}} \forall^{\log t} \underbrace{\mathrm{coNT}(t^{(1-\epsilon)\epsilon_\ell d\epsilon_\ell + o(1)})}_{(****)} && [\ \text{induction hypothesis}\ ] \\
&\subseteq \quad \exists^{t^{\epsilon + o(1)}} \mathrm{NT}((t^{\epsilon + o(1)} + t^{(1-\epsilon)\epsilon_\ell d\epsilon_\ell + o(1)})^c) && [\ \text{hypothesis 1 of the lemma}\ ] \\
&\subseteq \quad \mathrm{NT}((t^\epsilon + t^{(1-\epsilon)\epsilon_\ell^2 d})^{c + o(1)}) && [\ \text{simplification using } c \geq 1\ ]
\end{aligned}
\tag{34}
$$

Note that for all of $(*)$, $(**)$, and $(***)$, the input to the computation is only of length $n + t^{o(1)} = O(n)$. In particular, this means that by picking t to be a sufficiently large polynomial, we can make

34

sure that both $(*)$ and $(***)$ are valid applications of the induction hypothesis. The input to $(****)$ consists of the original input x and the guess bits of the first existential phase, so it is of length $n + t^{\epsilon + o(1)}$.

The final running time is optimized up to subpolynomial factors by equating ϵ and $(1 - \epsilon)\epsilon_\ell^2 d$, which results in the recurrence (33). $\hfill\square$

We point out that the second application of the induction hypothesis causes the alternation introducing and eliminating process to become considerably more complicated than the ones we have seen before. The process no longer consists of two separate phases where the first one introduces all alternations and the second one removes them by successively complementing the last block or the last two blocks of quantifiers. We also note that applying the induction hypothesis more than twice does not seem to work for the same reason why a second application in the deterministic case does not work.

Due to the extra application of the induction hypothesis, the transformation underlying the recurrence (33) is now a rational function of degree two rather than one. Proposition 4.2 only deals with transformations of degree one but some of the argumentation still applies. Since the transformation $\xi \to cd\xi^2/(1 + d\xi^2)$ over the reals is increasing, the sequence ϵ_ℓ is monotone. The sequence is decreasing iff $\epsilon_1 < \epsilon_0$, which is equivalent to $(c - 1)d < 1$. A simple calculation shows that the transformation $\xi \to cd\xi^2/(1 + d\xi^2)$ has 0 as its only fixed point for values of c and d such that $c^2 d < 4$. In that case, the sequence ϵ_ℓ decreases monotonically to zero. We conclude that for sufficiently large polynomials t

$$\mathrm{NT}(t) \subseteq \mathrm{coNTs}(t^d) \subseteq \mathrm{coNT}(t^{d\epsilon_\ell}),$$

which contradicts Lemma 3.1 for $c^2 d < 4$ and sufficiently large ℓ. Setting $c = d$, we get a time lower bound of $n^{\sqrt[3]{4} - o(1)}$ for subpolynomial-space nondeterministic algorithms for $\mathrm{coNT}(n)$. In the case where $c^2 d \geq 4$, the analysis becomes more involved [DvMW07] but it turns out that $d\epsilon_\ell \geq 1$ for every ℓ, so we do not obtain a contradiction.

Modulo a slightly closer analysis, the above arguments give us all the ingredients of the Master Theorem for nondeterministic algorithms (Theorem 1.5) and present the state-of-the-art on lower bounds for tautologies on nondeterministic machines.

6 Somewhat-Nonuniform Algorithms

This section investigates what the paradigm of indirect diagonalization from Section 3 can say about lower bounds for satisfiability in nonuniform models. We revisit the results from previous sections and show that, modulo some loss in parameters, certain uniformity requirements can be dropped or relaxed significantly. We will not succeed in obtaining lower bounds on fully nonuniform circuits but will obtain interesting lower bounds for specific types with relatively weak uniformity conditions.

Recall that the hypotheses we tried to rule out in the previous sections consist of two parts: that nondeterministic linear time can be simulated on conondeterministic machines in time n^c, and on deterministic or conondeterministic or nondeterministic machines that run in time n^d and space n^e. We used the first hypothesis for only one goal, namely reducing the number of alternations in alternating computations at a small cost in running time.

Karp and Lipton [KL82] showed that if satisfiability has polynomial-size circuits then the polynomial-time hierarchy collapses to the second level. Here is a more quantitative statement.

Lemma 6.1 (Karp-Lipton [KL82]) *If*

$$\mathrm{NT}(n) \subseteq \mathrm{SIZE}(n^c)$$

for some real c then

$$\Sigma_2\mathrm{T}(n) \subseteq \Pi_2\mathrm{T}(n^{c+o(1)}).$$

Proof: It is a bit easier to think about the lemma as transforming a linear-time Π_2-machine M into an equivalent Σ_2-machine N that runs in time $n^{c+o(1)}$. Let M' denote the nondeterministic machine induced by the existential and final deterministic phase of M. The Σ_2-machine N uses its existential phase to guess a circuit C of size n^c that purportedly solves the following problem: given a configuration of M' on inputs of size n, can M' reach an accepting halting configuration from there. N checks the validity of the circuit by verifying the following for every instance y of size $m = O(n)$: If the configuration y is halting then C accepts iff y is accepting; otherwise, C accepts y iff it accepts at least one of the configurations that can be reached in a single step from y. The machine N can guess the instance y during its universal phase and perform the check during its final deterministic phase in time $n^{c+o(1)}$. During its universal phase, N also makes the universal guesses M makes during its first phase. During its final deterministic phase, N uses the circuit C to simulate the computation of M during its existential and final deterministic phase. The latter takes time $n^{c+o(1)}$. □

As we described in Section 4.1, we can use an efficient complementation within the second level of the polynomial-time hierarchy to transform any alternating machine into an equivalent Σ_2-machine at a small cost in running time. Plugging in this component as a substitute for the efficient complementation within the first level, we obtain the following analogue to Lemma 4.1.

Lemma 6.2 *If*

$$\mathrm{NT}(n) \subseteq \mathrm{SIZE}(n^c)$$

for some real c, then for every nonnegative integer ℓ and for every sufficiently large polynomial t,

$$\mathrm{DTs}(t) \subseteq \Sigma_2\mathrm{T}(t^{\zeta_\ell+o(1)}),$$

where

$$
\begin{aligned}
\zeta_0 &= 1/2 \\
\zeta_{\ell+1} &= c\zeta_\ell/(1 + c\zeta_\ell).
\end{aligned}
\tag{35}
$$

Proof: The base case $\ell = 0$ of the induction proof holds by virtue of (8). For the induction step

$\ell \to \ell + 1$ we have for every real $\zeta \in (0,1)$ and sufficiently large polynomial t

$$
\begin{aligned}
\mathrm{DTs}(t) \quad &\subseteq \quad \exists t^{\zeta + o(1)} \forall^{\log t} \underbrace{\mathrm{DTs}(t^{1-\zeta})}_{(*)} && [\ (7)\ \text{with}\ b = t^{\zeta}\ ] \\
&\subseteq \quad \exists t^{\zeta + o(1)} \forall^{\log t} \underbrace{\Pi_2 \mathrm{T}(t^{(1-\zeta)\zeta_\ell + o(1)})}_{(**)} && [\ \text{induction hypothesis}\ ] \\
&\subseteq \quad \exists t^{\zeta + o(1)} \forall^{\log t} \underbrace{\Sigma_2 \mathrm{T}(t^{(1-\zeta)\zeta_\ell c + o(1)})}_{} && [\ \text{Lemma 6.1}\ ] && (36) \\
&\subseteq \quad \exists t^{\zeta + o(1)} \underbrace{\forall^{\log t} \exists t^{(1-\zeta)\zeta_\ell c + o(1)}}_{(***)} \mathrm{coNT}(t^{(1-\zeta)\zeta_\ell c + o(1)}) && [\ \text{expanding notation}\ ] \\
&\subseteq \quad \exists t^{\zeta + o(1)} \exists t^{(1-\zeta)\zeta_\ell c + o(1)} \forall^{\log t} \mathrm{coNT}(t^{(1-\zeta)\zeta_\ell c + o(1)}) && [\ \text{independence}\ ] \\
&\subseteq \quad \Sigma_2 \mathrm{T}(t^{\zeta + o(1)} + t^{(1-\zeta)\zeta_\ell c + o(1)}) && [\ \text{simplification}\ ]
\end{aligned}
$$

Note that the input to $(*)$ is only of length $n + t^{o(1)} = O(n)$ so the term $t^{\zeta + o(1)}$ does not appear in the running time of the Π_2-computation on the next line. The same holds for $(**)$ because we do not take up the $\forall^{\log t}$-part in the transformation. Not taking up those universal guesses fails to reduce the number of alternations in the next line. However, the guesses represented by $\exists t^{\zeta_\ell (1-\zeta)c + o(1)}$ in $(***)$ essentially correspond to a circuit for satisfiability on inputs of size $m = t^{\zeta_\ell (1-\zeta) + o(1)}$, which can be made independently of the choices in the preceding $\forall^{\log t}$-quantifier. Thus, the $\exists t^{\zeta_\ell (1-\zeta)c + o(1)}$-phase can be moved in front of the $\forall^{\log t}$-phase.

The final running time is optimized up to subpolynomial factors by equating ζ and $(1-\zeta)\zeta_\ell c$, which yields the recurrence (35). $\qquad\square$

By Proposition 4.2 the sequence (35) converges monotonically to $\zeta_\infty = 1 - 1/c$ for $c \geq 1$; the sequence is decreasing iff $c < 2$.

In order to continue the argument along the lines of (22) in Section 4.1, we need a second hypothesis that allows us to efficiently simulate Σ_2-computations (rather than Σ_1-computations as in Section 4.1) by deterministic machines that run in a small amount of space. If we assume $\Sigma_2 \mathrm{T}(n) \subseteq \mathrm{DTs}(n^d)$ as the second hypothesis, we can continue as follows: for every nonnegative integer ℓ and every sufficiently large polynomial t,

$$
\Sigma_2 \mathrm{T}(t) \subseteq \mathrm{DTs}(t^d) \subseteq \Pi_2 \mathrm{T}(t^{d\zeta_\ell + o(1)}).
$$

We obtain a contradiction with Lemma 3.1 for $k = 2$ if $d\zeta_0 < 1$ or $d\zeta_\infty < 1$, i.e., if $d < 2$ or $(c-1)d < c$. Since the existence of a $\mathrm{DTs}(n^d)$-algorithm for $\Sigma_2\mathrm{SAT}$ implies that satisfiability has circuits of size $n^{d+o(1)}$, we can assume that $c \leq d$. In that case $d < 2$ implies $(c-1)d < c$, so we can forget about the first condition.

We point out that the results of Section 4.2 allow us to directly obtain a somewhat weaker result without going through Lemma 6.2. By Lemma 6.1, our hypotheses imply

$$
\Sigma_2 \mathrm{T}(n) \subseteq \Pi_2 \mathrm{T}(n^{c+o(1)}) \cap \mathrm{DTs}(n^d).
$$

This is all we need to apply the generalization of the lower bound arguments for $\mathrm{NT}(n)$ from Section 4.1 to $\Sigma_2\mathrm{T}(n)$. In particular, the first part of Theorem 4.5 for $k = 2$ implies that we get a contradiction for $(c-1)d < 1$. In our setting, we managed to relax the latter condition to $(c-1)d < c$ using Lemma 6.2. The improvement is due to our exploiting the obliviousness of the

circuit we guess in the Karp-Lipton argument in the proof of Lemma 6.2. This allows us to shield part of the input to the Π_2-computation we need to complement in the third line of (36). We do not know of a way to exploit the obliviousness of the circuit in the context of Lemma 4.3, whose generalization yielded the second part of Theorem 4.5. We can still directly apply the second part of Theorem 4.5 for $k = 2$, though. All combined, we obtain the following counterpart to the Master Theorem for deterministic algorithms.

Theorem 6.3 *For all reals c and d such that $(c-1)d < c$ or $cd(d-2) - 2d + 2 < 0$, there exists a positive real e such that at least one of the following fails:*

(i) *satisfiability has circuits of size n^c and*

(ii) *$\Sigma_2\mathrm{SAT}$ has a deterministic random-access machine that runs in time n^d and space n^e.*

Moreover, the constant e approaches 1 from below when c approaches 1 from above and d is fixed.

Note the differences with Theorem 1.3: part (i) refers to circuits rather than conondeterministic machines, and part (ii) describes a machine for $\Sigma_2\mathrm{SAT}$ rather than for satisfiability. The true nonuniform equivalent of (i) in Theorem 1.3 uses conondeterministic rather than standard deterministic circuits. If we assume that satisfiability has small conondeterministic circuits, we can transform any alternating machine into an equivalent Σ_3-machine at a small cost in running time. This follows from an extension of the Karp-Lipton argument to conondeterministic circuits. Yap [Yap83] showed that if satisfiability has conondeterministic circuits of polynomial size then the polynomial-time hierarchy collapses to the third level. The proof is the same as the one for Lemma 6.1 except that the validity check for the circuit now takes an extra alternation. The argument yields the following quantitative statement similar to Lemma 6.1.

Lemma 6.4 (Yap [Yap83]) *If*
$$\mathrm{NT}(n) \subseteq \mathrm{coNSIZE}(n^c)$$
for some real c then
$$\Sigma_3\mathrm{T}(n) \subseteq \Pi_3\mathrm{T}(n^{c+o(1)}).$$

The resulting equivalent of Lemma 6.2 has the same parameters but uses the hypothesis $\mathrm{NT}(n) \subseteq \mathrm{coNSIZE}(n^c)$, yields simulations of DTs on Σ_3-machines, and can start with $\zeta_0 = 1/3$. We can also apply Theorem 4.5 with $k = 3$. The resulting equivalent to Theorem 6.3 reads as follows.

Theorem 6.5 *For all reals c and d such that $(c-1)d < c$ or $cd(d-3) - 2d + 3 < 0$, there exists a positive real e such that at least one of the following fails:*

(i) *satisfiability has conondeterministic circuits of size n^c and*

(ii) *$\Sigma_3\mathrm{SAT}$ has a deterministic random-access machine that runs in time n^d and space n^e.*

Moreover, the constant e approaches 1 from below when c approaches 1 from above and d is fixed.

Theorems 6.3 and 6.5 do not only involve satisfiability but also $\Sigma_2\mathrm{SAT}$ or $\Sigma_3\mathrm{SAT}$. We can deduce from Theorems 6.3 and 6.5 results that only refer to satisfiability. First, we can immediately replace (ii) by a more efficient algorithm for satisfiability, namely a $\mathrm{DTS}(n^{\sqrt{d}}, n^{e/\sqrt{d}})$-algorithm in the case

of Theorem 6.3 and a $\mathrm{DTS}(n^{\sqrt[3]{d}}, n^{e/d^{2/3}})$-algorithm in the case of Theorem 6.5. Alternatively, we can impose uniformity conditions on the circuits for satisfiability from part (i). If the uniformity conditions are sufficiently strong, we can trivially drop part (ii) all together; if they are weaker, the fact of having small fairly uniform circuits for satisfiability can still help us to relax the conditions on the fully uniform algorithm for satisfiability in part (ii). This is what happens in the proof of Theorem 1.6, which corresponds to a generalization to alternating machines of Theorem 6.5 for deterministic machines and a similar statement for nondeterministic machines corresponding to the results from Section 5 (which we will not elaborate on). Theorem 1.6 only refers to the setting with small values of c, though. This is because in that setting we know how to handle large values of d and the value of d ultimately figures in the bound on the running time of the uniformity algorithm for the circuits. Since we are shooting for as weak and natural uniformity conditions as possible, we would like to accommodate arbitrarily large polynomial running times for the uniformity algorithm, which forces us to the setting with small values of c.

We now show how to argue Theorem 1.6. The proof has some similarity with Fortnow's original proof of Theorem 1.1, which goes as follows. First, if $\mathrm{NT}(n) \subseteq \mathrm{coNT}(n^{1+o(1)})$ then the $n^{1+o(1)}$-time hierarchy collapses to the first level, so for every time bound $t \geq n$,

$$\cup_{a \geq 1} \Sigma_a \mathrm{T}(t^{1+o(1)}) = \mathrm{NT}(t^{1+o(1)}). \tag{37}$$

Second, if $\mathrm{NT}(n) \subseteq \mathrm{DTS}(n^d, n^e)$ for some real $e < 1$ then we can pick a positive real $\epsilon < 1 - e$ such that $t = n^{(1-\epsilon)/e}$ is a super-linear polynomial and by Lemma 3.4

$$\mathrm{NT}(t) \subseteq \mathrm{DTS}(n^{d(1-\epsilon)/e}, n^{1-\epsilon}) \subseteq \Pi_k \mathrm{T}(n)$$

for some integer k depending on d, e, and ϵ. Therefore, picking $a = k$ in (37) we obtain that $\Sigma_k \mathrm{T}(n^{(1-\epsilon)/e}) \subseteq \Pi_k \mathrm{T}(n^{1+o(1)})$, which contradicts Lemma 3.1 as $(1 - \epsilon)/e > 1$.

In our current setting, the hypothesis $\mathrm{NT}(n) \subseteq \mathrm{coNSIZE}(n^{1+o(1)})$ and repeated applications of Lemma 6.4 imply a collapse of the $n^{1+o(1)}$-time hierarchy to the third level, so for every time bound $t \geq n$ we have

$$\cup_{a \geq 1} \Sigma_a \mathrm{T}(t^{1+o(1)}) = \Sigma_3 \mathrm{T}(t^{1+o(1)}). \tag{38}$$

If our second hypothesis were $\Sigma_3 \mathrm{T}(n) \subseteq \mathrm{DTS}(n^d, n^e)$ for some real $e < 1$, we could finish the argument exactly as before. However, we only have $\mathrm{NT}(n) \subseteq \mathrm{DTS}(n^d, n^e)$ to work with. In the above results for NP-uniform circuits, we managed to bridge the gap by efficient simulations of Σ_3-computations by nondeterministic computations. We do not know of such simulations in the case of $\Sigma_k \mathrm{TS}(n^{O(1)}, n^{1-\epsilon})$-uniformity. However, we can directly show how to simulate $\Sigma_3 \mathrm{T}(t)$ for small but super-linear polynomials in some fixed level of the $n^{1+o(1)}$-time hierarchy, and that is all we need to finish the argument.

Consider the following attempt to obtain an efficient nondeterministic simulation of $\Sigma_3 \mathrm{T}(t)$. By hypothesis, we know that there exists a conondeterministic circuit C of size $t^{1+o(1)}$ that decides the final nondeterministic phase of the $\Sigma_3 \mathrm{T}(t)$-computation. The circuit C may be difficult to compute but once we have our hands on it, we can transform the $\Sigma_3 \mathrm{T}(t)$-computation into an equivalent $\Sigma_2 \mathrm{T}(t^{1+o(1)})$-computation. We can repeat this process using a circuit C' to reduce the $\Sigma_2 \mathrm{T}(t^{1+o(1)})$-computation further down to a $\mathrm{NT}(t^{1+o(1)})$-computation. Once we are there, we can run our $\Sigma_k \mathrm{TS}(n^d, n^e)$-algorithm for $\mathrm{NT}(n)$ to arrive at a situation where Lemma 3.5 applies, so we end up in some fixed level of the linear-time hierarchy. The only pieces of information we are missing to execute the above plan are the descriptions of the circuits C and C'. We can simply

consider those as additional inputs to the above process and generate the bits of the description as needed by the process, using the uniformity of the circuits.

More precisely, for every language $L \in \Sigma_3 \mathrm{T}(t)$ there exists a language L' in nondeterministic quasi-linear time such that

$$x \in L \Leftrightarrow \langle x, C, C' \rangle \in L',$$

where C and C' are the above circuits. By our second hypothesis there exists a Σ_k-machine M that decides whether $\langle x, C, C' \rangle \in L'$ in time N^d and space N^e, where $N = |\langle x, C, C' \rangle| = t^{1+o(1)}$. In order to decide L on input x, we run M on input $\langle x, C, C' \rangle$; each time M needs to access a bit of C or of C', we run the uniformity algorithm for the circuit to determine that bit on the fly.

If the circuits are $\mathrm{NTS}(n^d, n^e)$-uniform, the resulting algorithm for L is $\Sigma_k \mathrm{TS}(t^{2d+o(1)}, t^{e+o(1)})$. By Lemma 3.5, for polynomials t of degree less than $(1 - \epsilon)/e$, this puts L in some fixed level of the linear-time hierarchy, which leads to a contradiction with Lemma 3.1 since we also have (38).

If the uniformity algorithm is an alternating machine, the above simulation has a super-constant number of alternations and we do not know how to obtain a simulation in $\Sigma_\ell \mathrm{TS}(n^{O(1)}, n^{1-\epsilon})$ for some fixed ℓ and positive real ϵ, but we still obtain a simulation in a fixed level of the linear-time hierarchy. We can view our algorithm for L as an oracle computation, where M makes oracle queries to the uniformity oracle for the circuits C and C'. For $t = n^{(1-\epsilon)/e}$, both the base computation and the oracle computations lie in $\Sigma_k \mathrm{TS}(n^{(1-\epsilon)/e \cdot d + o(1)}, n^{1-\epsilon})$, so by Lemma 3.5 in $\Sigma_\ell \mathrm{T}(n)$, where ℓ is an integer which only depends on k, d, e, and ϵ. The following standard fact then shows that the overall computation lies in $\Sigma_{2\ell} \mathrm{T}(n^{1+o(1)})$. In fact, in our context we can shave off one alternation but we will not worry about that. For completeness we include a proof of the fact.

Proposition 6.6 *For all integers $\ell \geq 1$ and $m \geq 0$,*

$$\Sigma_\ell \mathrm{T}(n)^{\Sigma_m \mathrm{T}(n)} \subseteq \Sigma_{\ell+m} \mathrm{T}(n).$$

Proof: It suffices to provide a relativizable proof for $\ell = 1$ and arbitrary nonnegative integer m, since we can then argue by induction on ℓ that

$$\Sigma_{\ell+1} \mathrm{T}(n)^{\Sigma_m \mathrm{T}(n)} \subseteq \mathrm{NT}(n)^{\Sigma_\ell \mathrm{T}(n)^{\Sigma_m \mathrm{T}(n)} \oplus \Sigma_m \mathrm{T}(n)} \subseteq \mathrm{NT}(n)^{\Sigma_{\ell+m} \mathrm{T}(n)},$$

where $\oplus$ denotes disjoint union.

Consider an $\mathrm{NT}(n)$-machine M that makes oracle queries to a $\Sigma_m \mathrm{T}(n)$-machine O. We construct an equivalent $\Sigma_{m+1} \mathrm{T}(n)$-machine N as follows.

N starts out in an existential phase and runs M up to the point where M is about to output its decision b. Each time M makes an oracle query q, N makes a guess a for the answer, stores the pair $\langle q, a \rangle$ on a separate tape, and continues the simulation of M assuming the oracle answer a. When M is finished, N rejects outright if $b = 0$. Otherwise, N wants to accept iff all guesses for the answers to the queries were correct. It verifies the query-answer pairs in the following fashion. For all the queries q that were answered positively, it runs O on input q. For the other queries, it runs $\overline{O}$ on input q, where $\overline{O}$ denotes the $\Sigma_{m+1} \mathrm{T}(n)$-machine that complements the $\Sigma_m \mathrm{T}(n)$-machine O in the trivial way. N accepts iff all the runs accept. N executes these runs in parallel, synchronizing the existential and universal phases such that the overall simulation is of type Σ_{m+1}. Since the sum of the lengths of the queries M makes on a valid computation path is bounded by a linear function and O runs in linear time, we can clock N to run in linear time without affecting the strings N accepts. $\qquad \square$

We can then finish the indirect diagonalization argument as before. By choosing $a = 2\ell + 1$ in (38) we have

$$\Pi_{2\ell}\mathrm{T}(n^{(1-\epsilon)/e}) \subseteq \Sigma_{2\ell+1}\mathrm{T}(n^{(1-\epsilon)/e}) \subseteq \Sigma_3\mathrm{T}(n^{(1-\epsilon)/e+o(1)}) \subseteq \Sigma_{2\ell}\mathrm{T}(n^{1+o(1)}),$$

which contradicts Lemma 3.1 since $(1 - \epsilon)/e > 1$. This finishes the proof of Theorem 1.6.

We next argue the instantiations of the proof of Theorem 1.6 stated in Corollary 1.7. Circuits of size s and width w can be evaluated simultaneously in time $s \log^{O(1)} s$ and space $O(w \log s)$. SAC^1-circuits can be evaluated in NTS($n^{O(1)}, \log^2 n$) [Ruz80]. All of these simulations just need access to the input and the description of the circuit. Let us denote the latter circuit by C''. In the setting of the proof of Theorem 1.6, the input is of the form $\langle x, C, C' \rangle$ and the circuit C'' is the one from the same family as C and C' but for the input length $|\langle x, C, C' \rangle|$. The proof of Theorem 1.6 still works. The only modification is that the base machine M now corresponds to one of the above circuit-simulating algorithms, which all run in the first level of the polynomial-time hierarchy or better using space at most $n^{1-\epsilon}$. Apart from x and the descriptions of the circuits C and C', the circuit-simulating algorithms also need access to the description of C''. The bits of those descriptions are again generated on the fly using the uniformity algorithm for the circuit family. A similar application of Proposition 6.6 as in the proof of Theorem 1.6 is the key to complete the proof of Corollary 1.7.

Finally, we mention that there is another direction in which we can make the models to which our lower bound arguments apply a bit more nonuniform – most of the results we covered carry through when we supply the machines with a subpolynomial amount of advice. If our hypothetical machines solving hard problems have advice, we can pass along and collect all the advice strings we need during the various transformations. Since we only apply a constant number of transformations and each one involves at most a polynomial blowup in resources, the total amount of advice remains subpolynomial. In the case where we were aiming for a contradiction with Lemma 3.1, the final line of our proof now becomes of the form

$$\Sigma_k\mathrm{T}(n^b) \subseteq \Pi_k\mathrm{T}(n^a)/n^{o(1)}$$

for some positive integer k and reals a and b with $1 \leq a < b$. Because the proof of Lemma 3.1 can handle up to n bits of advice on the smaller time side, we still obtain the contradiction we want. The same holds when we use Lemma 3.3 in Section 8. However, we do not know whether Lemma 3.2 holds with $n^{o(1)}$ bits of advice on the smaller time side. Of the arguments we present in this survey, only one relies on Lemma 3.2, namely the extension of Theorem 1.5 we mentioned in Section 5 (see (32)). And even there, we could perform another complementation and use Lemma 3.1 instead at the cost of some deterioration in parameters.

7 Randomized Algorithms

This section describes the known lower bounds on randomized machines with bounded error. The simplest problems for which we have nontrivial results are Σ_2SAT in the case of two-sided error, and tautologies in the case of one-sided error. We first discuss those results and then mention stronger lower bounds for Σ_kSAT with $k > 2$.

7.1 Satisfiability and Σ_2SAT

We again follow the paradigm of indirect diagonalization as described in Section 3, using the polynomial-time hierarchy as the substrate for intermediate computations. We can bring the ideas behind the deterministic lower bounds to bear in the randomized setting by exploiting efficient simulations of randomized algorithms on Σ_2-machines. The standard simulations, due to Sipser-Gacs [Sip83] and Lautemann [Lau83], transform a BPTS(t, s)-machine into a Σ_2-machine that runs a DTS$(t^{O(1)}, s)$-computation after making its existential and universal guesses. The hope is to apply the unconditional speedups of deterministic space-bounded computations from Section 3.2 to this final deterministic phase and eliminate the induced alternations so as to obtain a net speedup that contradicts Lemma 3.1 or another result of the time hierarchy ilk.

It turns out that we need to modify the standard simulations somewhat in order to obtain lower bounds for Σ_2SAT. Let us start by analyzing the complexity of Lautemann's simulation, paying attention to its dependence on the error bound.

Let M denote a BPTS(t, s)-machine that uses r random bits and decides a language L with error on both sides bounded by ϵ. When ϵ is small enough in comparison to r, we can characterize the inputs x that are in the language L defined by M as those for which we can cover the entire universe of possible random strings of length r by a small number of shifts of the set of random strings that lead M to accept on input x. Let us denote the number of shifts by σ. If $x \in L$, we can guarantee that such shifts exist as long as $\epsilon^\sigma < 1/2^r$. If $x \notin L$, no choice of σ shifts can cover the universe of random strings as long as $\epsilon < 1/\sigma$. For such ϵ and σ, these complementary conditions provide a Σ_2-predicate that defines L.

Lemma 7.1 (Lautemann [Lau83]) *Let L be a language recognized by a randomized machine M that runs in time t, space s, and uses r random bits with error bounded on both sides by ϵ. Then for any function σ such that $\epsilon < 1/\max(2^{r/\sigma}, \sigma)$, we have that*

$$L \in \exists^{\sigma r} \forall^r \mathrm{DTS}(\sigma t, s + \log \sigma). \tag{39}$$

In his proof that BPP lies in the second level of the polynomial-time hierarchy, Lautemann starts from an algorithm deciding L with error less than the reciprocal of the number r' of random bits it uses. Such an error probability can be achieved from a standard randomized algorithm with error bounded by $1/3$ and using r random bits by taking the majority vote of $O(\log r)$ independent trials, which results in $r' = O(r \log r)$. For $\epsilon < \frac{1}{r'}$, choosing $\sigma = r'$ satisfies the conditions of Lemma 7.1. This shows that a BPTS(t, s)-computation using r random bits can be simulated in

$$\exists^{(r \log r)^2} \forall^{r \log r} \mathrm{DTS}(tr(\log r)^2, s). \tag{40}$$

Once we have transfered the computation to the polynomial-time hierarchy, the idea is to use the techniques from Section 3.2 to speed it up. However, as r can be of the same order as t, the final deterministic phase may take $\Omega(t^2)$ time. Since the techniques from Section 3.2 can realize a square-root speedup at best, overall we would not gain anything in computation time. Moreover, even if we could speed up the final deterministic phase, the existential and universal phases could still take $\Omega(t^2)$ time. The solution to both issues is to reduce the number of random bits without increasing the time or space by much. Since we are working in a space-bounded setting, we have good pseudorandom generators at our disposal to do so. We state the properties we need of Nisan's pseudorandom generator [Nis92].

Lemma 7.2 (Nisan [Nis92]) *Every randomized machine M running in time t and space s with error ϵ can be simulated by another randomized machine that runs in time $O(t \cdot (\log t)^{O(1)})$ and space $O(s \log t)$ and uses only $O(s \log t)$ random bits. The error of the simulation is $\epsilon + 1/2^s$, and is one-sided if M has one-sided error.*

Note that we do not apply Lemma 7.2 to deterministically simulate the randomized machine. Instead, we use it to reduce the randomness required by a $\mathrm{BPTS}(t, s)$-machine to $O(s \log t)$. Combined with (40), we conclude that

$$\mathrm{BPTS}(t, s) \subseteq \exists^{s^2 (\log t)^3} \forall^{s (\log t)^2} \mathrm{DTS}(ts(\log t)^{O(1)}, s \log t). \tag{41}$$

Let us illustrate how this simulation leads to a time-space lower bound for $\Sigma_2 \mathrm{T}(n)$. For simplicity we only consider the range of subpolynomial space. Assume by way of contradiction that

$$\Sigma_2 \mathrm{T}(n) \subseteq \mathrm{BPTs}(n^d)$$

for some real d. First notice that the hypothesis combined with (41) and the closure of BPTs under complementation allows us to efficiently complement computations within the second level of the polynomial-time hierarchy. More specifically, we have that

$$\Sigma_2 \mathrm{T}(n) \subseteq \mathrm{BPTs}(n^d) \subseteq \Pi_2 \mathrm{T}(n^{d+o(1)}). \tag{42}$$

By Lemma 3.1, this means that d cannot be less than 1. Moreover, we know that the latter Π_2-computation only guesses $n^{o(1)}$ bits in its existential and universal phases, and then runs a $\mathrm{DTs}(n^{d+o(1)})$-computation on an input of size $n + n^{o(1)}$. Thus, starting from a $\Sigma_2 \mathrm{T}(n^2)$-computation instead of a $\Sigma_2 \mathrm{T}(n)$-computation in (42), we can apply the square-root speedup (8) to the final $\mathrm{DTs}(n^{2d+o(1)})$-phase on the right-hand side of (42) and obtain a simulation which makes one more alternation than we started with but only runs a $\mathrm{DTs}(n^{d+o(1)})$-computation in its final deterministic phase. To balance the number of alternations, we eliminate one of them. We do so by viewing the part of the simulation after the first universal phase as a $\Sigma_2 \mathrm{T}(n^{d+o(1)})$-computation on an input of size $n + n^{o(1)}$ and turn it into an equivalent Π_2-computation using the efficient complementation (42) once more. Merging the resulting adjacent existential stages yields that

$$\Sigma_2 \mathrm{T}(n^2) \subseteq \Pi_2 \mathrm{T}(n^{d^2+o(1)}). \tag{43}$$

For $d < \sqrt{2}$, this results in a net speedup, which is a contradiction to Lemma 3.1. For values of d in the range $[\sqrt{2}, 2)$, the conclusion (43) does not immediately yield a contradiction with Lemma 3.1 but it gives us a complementation of Σ_2-computations that is more efficient than the one given by (42), at least for running times that are quadratic or higher. We then go through the argument again using the more efficient complementation, which allows us to rule out larger values of d and yields an even more efficient complementation for values of d less than 2, and so on. We analyze this bootstrapping argument in the next lemma.

Lemma 7.3 (Diehl-Van Melkebeek [DvM06]) *If*

$$\Sigma_2 \mathrm{T}(n) \subseteq \mathrm{BPTs}(n^d)$$

for some real d, then for every nonnegative integer ℓ and every sufficiently large polynomial t

$$\Sigma_2 \mathrm{T}(t) \subseteq \Pi_2 \mathrm{T}(t^{\eta_\ell + o(1)}),$$

where $\eta_\ell = (d/2)^\ell d$.

Proof: Let us consider the induction step $\ell \to \ell + 1$ first.

$$
\begin{aligned}
\Sigma_2\mathrm{T}(t) \;\subseteq\;& \mathrm{BPTs}(t^d) && [\text{ hypothesis of the lemma }] \\
\subseteq\;& \forall^{t^{o(1)}} \exists^{t^{o(1)}} \underbrace{\mathrm{DTs}(t^{d+o(1)})}_{(*)} && [\,(41)\,] \\
\subseteq\;& \forall^{t^{o(1)}} \underbrace{\exists^{t^{o(1)}} \Sigma_2\mathrm{T}(t^{d/2+o(1)})}_{(**)} && [\,(8)\,] \\
\subseteq\;& \forall^{t^{o(1)}} \Pi_2\mathrm{T}((t^{d/2})^{\eta_\ell+o(1)}) && [\text{ induction hypothesis }] \\
\subseteq\;& \Pi_2\mathrm{T}(t^{d/2\cdot\eta_\ell+o(1)}) && [\text{ simplification }]
\end{aligned}
$$

Note that the input to $(*)$ and to $(**)$ is of size $n+t^{o(1)} = O(n)$. This justifies setting $\eta_{\ell+1} = (d/2)\eta_\ell$. The base case follows from the first two lines of the induction step. $\qquad\square$

For values of $d < 2$, the sequence η_ℓ converges to 0, so Lemma 7.3 yields a contradiction with Lemma 3.1 for $k = 2$. This establishes Theorem 1.8 for the case of subpolynomial space bounds. The same construction works for space bounds that are a bit larger. In order to show that the constant e in Theorem 1.8 converges to $1/2$ when d approaches 1, a different simulation than (41) is needed, namely one where the dependence on s of the number of guesses in the initial phase is only linear rather than quadratic. Such a simulation can be obtained using randomness-efficient methods for error reduction. With such methods, the error can be made as small as $1/2^r$ while using only $O(r)$ random bits. Specifically, the algorithm runs $O(r)$ trials which are obtained from the labels of vertices on a random walk of length $O(r)$ in an easily constructible expander graph, and accepts if a majority of these trials accept. We can use the Gabber-Galil family of expanders [GG81], a construction based on the Margulis family [Mar73], where the vertices are connected via simple affine transformations on the labels. The easy form of the edge relations ensures that the walk is efficiently computable in time $O(r^2)$ and space $O(r)$.

Lemma 7.4 ([CW89, IZ89]) *Let M be a randomized machine with constant error bounded away from $1/2$ that runs in time t, space s, and uses r random bits. Then M can be simulated by another randomized machine M' that runs in time $O(rt)$ and space $O(r+s)$, while using only $O(r)$ random bits to achieve error at most $1/2^r$.*

Once an algorithm has been amplified as in Lemma 7.4, a constant number σ of shifts suffice in Lautemann's theorem (Lemma 7.1). This shows that a $\mathrm{BPTS}(t,s)$-computation that uses r random bits can be simulated in $\exists^r\forall^r\mathrm{DTS}(tr, r+s)$. Combining this simulation with Nisan's pseudorandom generator (Lemma 7.2) shows that

$$
\mathrm{BPTS}(t,s) \subseteq \exists^{s\log t}\forall^{s\log t}\mathrm{DTS}(ts(\log t)^{O(1)}, s\log t). \tag{44}
$$

Using this simulation instead of (41) in the above argument allows us to establish Theorem 1.8 for space bounds close to $\sqrt{n}$.

In the case of randomized algorithms with one-sided error (i.e., there can only be errors on inputs that are in the language), we can establish lower bounds for a problem that is easier than $\Sigma_2\mathrm{SAT}$, namely tautologies. The results from Section 5 trivially imply such lower bounds, since randomized computations with one-sided error are special cases of nondeterministic computations.

In particular, Theorem 1.5 implies a lower bound of $n^{\sqrt[3]{4}-o(1)}$ for every subpolynomial-space one-sided error randomized algorithm for tautologies. But we can do better. In fact, Diehl and Van Melkebeek observed that we can match every deterministic time-space lower bound that carries over to conondeterministic algorithms that guess few bits. The reason we can match such bounds is that we can use Nisan's pseudorandom generator from Lemma 7.2 to transform a one-sided error randomized algorithm for tautologies into an equivalent conondeterministic algorithm that guesses few bits and takes only marginally more time and space. To date, all lower bound arguments for satisfiability on deterministic machines carry over to conondeterministic machines with few guess bits. For example, the arguments from Section 4.1 can be adapted to show that $\mathrm{NT}(n) \not\subseteq \forall^{n^{o(1)}} \mathrm{DTs}(n^{2\cos(\pi/7)-o(1)})$, which implies that $\mathrm{coNT}(n) \not\subseteq \mathrm{RTs}(n^{2\cos(\pi/7)-o(1)})$. More generally, we obtain the following.

Theorem 7.5 (follows from [Wil07b, DvM06]) *For all reals c and d such that $(c-1)d < 1$ or $cd(d-1) - 2d + 1 < 0$, there exists a positive real e such that tautologies cannot be solved by both*

   (i) a randomized random-access machine with one-sided error that runs in time n^c and

   (ii) a randomized random-access machine with one-sided error that runs in time n^d and space n^e.

Moreover, the constant e approaches 1 from below when c approaches 1 from above and d is fixed.

7.2 Related Problems

The lower bound argument for $\Sigma_2\mathrm{SAT}$ on randomized machines with two-sided error readily generalizes to $\Sigma_k\mathrm{SAT}$ for values of $k > 2$. Under the hypothesis $\Sigma_k\mathrm{T}(n) \subseteq \mathrm{BPTs}(n^d)$, we can apply the bootstrapping strategy from Section 7.1 to show that, for every nonnegative integer ℓ and sufficiently large polynomial t, $\Sigma_k\mathrm{T}(t) \subseteq \Pi_k\mathrm{T}(t^{\eta'_\ell+o(1)})$, where $\eta'_\ell = (d/k)^\ell \cdot d/(k-1)$. This leads to a contradiction with Lemma 3.1 as long as $d < k$.

We can handle larger space bounds for $k > 2$ than for $k = 2$. Recall that Theorem 1.8 only worked for space bounds of the form n^e for $e < 1/2$. For $k > 2$, we can handle every real $e < 1$. The reason is that we can exploit the extra alternation to establish a more time-efficient simulation of $\mathrm{BPTS}(t,s)$-computations than (41) and (44). More specifically, we add an alternation to the latter Σ_2-simulation and eliminate the time blowup incurred by running the $O(s \log t)$ trials required by the error reduction of Lemma 7.4. Rather than deterministically simulate all of these trials, we use the power of alternation to efficiently verify that a majority of these trials accept. This leads to the simulation

$$\mathrm{BPTS}(t,s) \subseteq \exists^{s\log t} \forall^{s\log t} \exists^{\log s} \mathrm{DTS}(t(\log t)^{O(1)}, s\log t). \tag{45}$$

As in Section 7.1, this simulation admits a speedup of BPTS on Σ_3-machines as well as an efficient complementation of Σ_3. Under the hypothesis $\Sigma_k\mathrm{T}(n) \subseteq \mathrm{BPTS}(n^d, n^e)$, the latter eliminates alternations essentially at the cost of raising the running time to the power of d. For values of d close to 1, this cost is small enough to alleviate the effects of the extra alternation in (45). In this case, the better dependence of the running time of the simulation on the space parameter allows us to derive contradictions for larger values of e than we can by using (44). On the other hand, for larger values of d, the extra alternation in (45) has a greater impact, and eventually prevents us from reaching a contradiction. In this case, switching to the more alternation-efficient simulation given by (44) allows us to do better. All combined we obtain the following statement.

Theorem 7.6 (Diehl-Van Melkebeek [DvM06]) *For every integer $k \geq 3$ and every real $d < k$ there exists a positive real e such that $\Sigma_k\mathrm{SAT}$ cannot be solved by a randomized random-access machine with two-sided error that runs in time n^d and space n^e. Moreover, e approaches 1 from below as d approaches 1 from above.*

The simulation (45) also plays a role in an extension of the time-space lower bound for $\Sigma_3\mathrm{SAT}$ in the range of small time bounds. Viola [Vio07] showed that $\Sigma_3\mathrm{T}(n) \not\subseteq \mathrm{BPTS}(n^{1+o(1)}, n^{1-\epsilon})$ also holds in the more powerful model of space-bounded randomized computation where the machine has sequential two-way access to the random bits (and random access to everything else). The proof uses the pseudorandom generator by Impagliazzo et al. [INW94] instead of Nisan's pseudorandom generator. The latter pseudorandom generator critically depends on the one-way access to the random bits; the former one also works in the two-way access model but only in a very limited range. This is why only the result for small time bounds carries over.

Finally, Diehl [Die07] and others observed that the lower bounds for $\Sigma_2\mathrm{T}(n)$ immediately imply somewhat weaker lower bounds for a class that lies between $\mathrm{NT}(n)$ and $\Sigma_2\mathrm{T}(n)$, namely $\mathrm{MA}_1\mathrm{T}(n)$, which stands for Merlin-Arthur protocols with perfect completeness that run in linear time. Merlin-Arthur protocols [Bab85, BM88] are proof systems for a language L in which Merlin presents a purported proof of membership of the input x to Arthur, who then verifies the proof using randomness. The requirements for the proof system are: (perfect completeness) for inputs $x \in L$, Merlin has to be able to convince Arthur with probability one, and (soundness) for inputs $x \notin L$, Arthur can be convinced of membership only with bounded probability, no matter whether Merlin follows the protocol or not. The running time of the protocol is the time Arthur spends in the communication with Merlin and in the verification. We can argue that if $\Sigma_2\mathrm{T}(n) \not\subseteq \mathrm{BPTs}(n^d)$ then $\mathrm{MA}_1\mathrm{T}(n) \not\subseteq \mathrm{BPTs}(n^{\sqrt{d}-o(1)})$. The reason is similar as to why in the deterministic setting $\Sigma_2\mathrm{T}(n) \not\subseteq \mathrm{DTs}(n^d)$ implies $\mathrm{NT}(n) \not\subseteq \mathrm{DTs}(n^{\sqrt{d}-o(1)})$ and is perhaps a bit easier to understand in the contrapositive. Assuming that $\mathrm{MA}_1\mathrm{T}(n) \subseteq \mathrm{BPTs}(n^d)$, we have:

$$
\begin{aligned}
\Sigma_2\mathrm{T}(n) \quad &\subseteq \quad \exists^n \underbrace{\forall^n \mathrm{DT}(n)}_{} \\
&\subseteq \quad \exists^n \underbrace{\mathrm{BPTs}(n^d)}_{} && [\ \mathrm{NT}(n) \subseteq \mathrm{MA}_1\mathrm{T}(n) \text{ and hypothesis }] \\
&\subseteq \quad \exists^n \exists^{n^{o(1)}} \underbrace{\mathrm{coЯ}^{n^{o(1)}}}_{(*)} \mathrm{DTs}(n^{d+o(1)}) && [\ (40)\ ] \\
&\subseteq \quad \mathrm{MA}_1\mathrm{T}(n^{d+o(1)}) && [\ \text{rewriting }] \\
&\subseteq \quad \mathrm{BPTs}(n^{d^2+o(1)}) && [\ \text{hypothesis }]
\end{aligned}
$$

The funny coЯ-quantifier in $(*)$ indicates a special property of Lautemann's simulation from Lemma 7.1, namely that for inputs outside of L each choice of shifts can only cover a small fraction of the universe, provided the error bound is small enough. This means that the resulting simulation represents a Merlin-Arthur protocol with perfect completeness.

By Theorem 1.8, we conclude from the above reasoning that $\mathrm{MA}_1\mathrm{T}(n) \not\subseteq \mathrm{BPTs}(n^{\sqrt{2}-o(1)})$. In fact, Watson used a direct argument along the lines of Lemma 4.1 to show that $\mathrm{MA}_1\mathrm{T}(n) \not\subseteq \mathrm{BPTs}(n^{\phi-o(1)})$. Unfortunately, we do not know of a specific problem that captures the complexity of $\mathrm{MA}_1\mathrm{T}(n)$ in a similar way as satisfiability does for $\mathrm{NT}(n)$, and the result does not seem to yield better lower bounds for natural computational problems. The same applies to the lower bounds which Diehl and Van Melkebeek [DvM06] derive for classes of the form $\Sigma_k\mathrm{TS}(n, s)$ for sublinear s. For that reason we do not cover those results.

8   Quantum Algorithms

This section describes the known time and space lower bounds on quantum models, namely for MajMajSAT. They follow from similar lower bounds on randomized machines with unbounded error. We first describe the latter results and then show how they translate to the quantum setting.

8.1   Randomized Algorithms with Unbounded Error

Randomized machines with unbounded error do not represent a realistic model of computation but they allow us to capture the complexity of counting problems in a fairly tight way. We can obviously decide languages in PP when given the ability to count NP-witnesses, and the latter is something we can do using binary search when given access to an oracle for PP.

As we mentioned in Section 2.3, MajSAT captures the complexity of the class PP in the same way as satisfiability captures the complexity of NP. MajSAT is complete for quasi-linear time on randomized machines with unbounded error under the very efficient reductions given in Lemma 2.3. As a result, time and space lower bounds for MajSAT and for $\mathrm{PT}(n)$ are equivalent up to polylogarithmic factors.

If we would like to establish lower bounds for $\mathrm{PT}(n)$ following the indirect diagonalization paradigm from Section 3, Toda's Theorem suggests that we should look for another substrate than the polynomial-time hierarchy. Instead, we use the so-called counting hierarchy, which is a hierarchy built on top of PP in the same way as the polynomial-time hierarchy is built on top of NP. Just like the polynomial-time hierarchy has alternate characterizations in terms of (α) uniform Boolean formulas of polynomial size with existential and universal quantifiers, and (β) uniform constant-depth circuits of exponential size consisting of AND and OR gates, the counting hierarchy can be characterized in terms of (α) uniform Boolean formulas of polynomial size with majority quantifiers, and (β) uniform constant-depth circuits of exponential size consisting of threshold gates.

Allender et al. developed an analogue of Fortnow's result in the counting hierarchy. Fortnow relies on Nepomnjascii's Theorem to show that, under the hypothesis $\mathrm{NT}(n) \subseteq \mathrm{NTS}(n^d, n^e)$ for some reals d and $e < 1$, there exists a super-linear polynomial t such that computations in the first level of the polynomial-time hierarchy that run in time t can be simulated in linear time in some higher level of the polynomial-time hierarchy. He then uses the hypothesis $\mathrm{NT}(n) \subseteq \mathrm{coNT}(n^{1+o(1)})$ to show that the $n^{1+o(1)}$-time hierarchy collapses to the first level. All together we obtain a speedup of computations within the first level from time t to time $n^{1+o(1)}$, which contradicts the time hierarchy for nondeterministic computations (Lemma 3.2). The critical ingredient in the analogue by Allender et al. is the equivalent of Nepomnjascii's Theorem in the counting hierarchy. They use their equivalent to prove that, under the hypothesis $\mathrm{PT}(n) \subseteq \mathrm{PTS}(n^d, n^e)$ for some reals d and $e < 1$, there exists a super-linear polynomial t such that computations in the first level of the counting hierarchy that run in time t can be simulated in linear time in some higher level of the counting hierarchy. They then employ another hypothesis that implies a collapse of the $n^{1+o(1)}$-time counting hierarchy to the first level. The net result is again a speedup of computations within the first level of the hierarchy from time t to time $n^{1+o(1)}$, which is ruled out by the time hierarchy for randomized machines with unbounded error (Lemma 3.3).

The hypothesis Allender et al. need to collapse the $n^{1+o(1)}$-time counting hierarchy to the first level is a bit different than what Fortnow needs for the similar collapse in the polynomial-time hierarchy. This is due to one salient difference between the two hierarchies – whereas complementation within a fixed level of the polynomial-time hierarchy is believed to be difficult, it is

trivial in the case of the counting hierarchy. Indeed, even the linear-time levels of the counting hierarchy are closed under complement. Thus, the equivalent of Fortnow's collapse hypothesis in the counting hierarchy, namely $\mathrm{PT}(n) \subseteq \mathrm{coPT}(n^{1+o(1)})$ does not buy us anything. Instead, we can use the hypothesis that linear-time computations in the second level of the counting hierarchy can be simulated in $\mathrm{PT}(n^{1+o(1)})$ to collapse the $n^{1+o(1)}$-time counting hierarchy. Capitalizing on the completeness under very efficient reductions of MajSAT and MajMajSAT for the first and the second level of the counting hierarchy, respectively, Allender et al. obtain the following more precise statement.

Theorem 8.1 (Allender et al. [AKR$^+$01]) *For every real d and positive real ϵ there exists a real $c > 1$ such that it cannot be the case that both*

  (i) MajMajSAT *has a randomized algorithm with unbounded error that runs in time n^c and*

  (ii) MajSAT *has a randomized algorithm with unbounded error that runs in time n^d and space $n^{1-\epsilon}$.*

Corollary 8.2 *For every positive real ϵ there exists a real $d > 1$ such that* MajMajSAT *does not have a randomized algorithm with unbounded error that runs in time n^d and space $n^{1-\epsilon}$.*

As usual, we will focus on the less precise version of Theorem 8.1 with $c = 1 + o(1)$. Our presentation is somewhat more elementary than the one by Allender et al., who rely on the hardness under very efficient reductions of MajMajSAT for the second level of the counting hierarchy, i.e., for $\mathrm{PP}^{\mathrm{PP}}$. Although it is straightforward to show that MajMajMajSAT is hard for $\mathrm{PP}^{\mathrm{PP}}$, it takes more work to show that MajMajSAT is [Tor91]. It turns out that the full power of the hardness result for MajMajSAT is not needed and can be sidestepped by arguing in terms of Boolean formulas with majority quantifiers rather than in terms of relativized randomized computations with unbounded error. That is the approach we will follow.

Before doing so, let us first cast the two steps of Fortnow's argument in terms of manipulating Boolean formulas with existential and universal quantifiers.

  1. If satisfiability is in $\mathrm{coNT}(n^{1+o(1)})$ then we can transform in time $n^{1+o(1)}$ a Boolean formula $\varphi(x, y, z)$ into another Boolean formula $\varphi'(x, y')$ such that for every setting of x,

$$(\exists y)(\forall z)\varphi(x, y, z) \Leftrightarrow (\exists y')\varphi'(x, y').$$

  2. If $\mathrm{NT}(n) \subseteq \mathrm{NTS}(n^d, n^e)$ for some reals d and $e < 1$, then for every real $f < 1/e$ there exists an integer ℓ such that for every language $L \in \mathrm{NT}(n^f)$ and every input length n, we can generate in time $n^{1+o(1)}$ a Boolean formula $\varphi(x, y_1, y_2, \ldots, y_\ell)$ such that for every input x of length n,

$$x \in L \Leftrightarrow (\exists y_1)(\forall y_2) \ldots (\overline{Q}y_{\ell-1})(Qy_\ell)\varphi(x, y_1, y_2, \ldots, y_{\ell-1}, y_\ell),$$

  where Q denotes $\forall$ if ℓ is even, Q denotes $\exists$ if ℓ is odd, and $\overline{Q}$ denotes the complement of Q.

We now lift the above two steps to the counting hierarchy using majority formulas, which we inductively define as follows.

Definition 8.3 *A Boolean formula is a majority formula. If $\varphi(x, y)$ denotes a majority formula on variables x and y, then $(\mathsf{W}y)\varphi(x, y)$ is a majority formula on variables x. For any given setting of x, $(\mathsf{W}y)\varphi(x, y)$ holds if for at least half of the settings of y, $\varphi(x, y)$ holds.*

We refer to the symbol W introduced in Definition 8.3 as a majority quantifier. It is a mnemonic for "majority" or "most" in the same way that $\exists$ is for "exists" and $\forall$ for "all." Majority quantifiers form a resource similar to alternations in regular quantified formulas.

The equivalent of the above two steps in Fortnow's approach in terms of majority formulas reads as follows.

1. If MajMajSAT $\in$ PT$(n^{1+o(1)})$ then we can transform in time $n^{1+o(1)}$ a Boolean formula $\varphi(x, y, z)$ into another Boolean formula $\varphi'(x, y')$ such that for every setting of x,

$$(\mathsf{W}y)(\mathsf{W}z)\varphi(x, y, z) \Leftrightarrow (\mathsf{W}y')\varphi'(x, y'). \tag{46}$$

2. If PT$(n) \subseteq$ PTS(n^d, n^e) for some reals d and $e < 1$, then for every real $f < 1/e$ there exists an integer ℓ such that for every language $L \in$ PT(n^f) and input length n, we can generate in time $n^{1+o(1)}$ a Boolean formula $\varphi(x, y_1, y_2, \dots, y_\ell)$ such that for every input x of length n,

$$x \in L \Leftrightarrow (\mathsf{W}y_1)(\mathsf{W}y_2)\dots(\mathsf{W}y_{\ell-1})(\mathsf{W}y_\ell)\varphi(x, y_1, y_2, \dots, y_{\ell-1}, y_\ell). \tag{47}$$

Once we have those two steps, the proof of Theorem 8.1 is straightforward. Part (ii) of the statement yields that PT$(n) \subseteq$ PTS$(n^{d+o(1)}, n^e)$ for some $e < 1$. Pick f to be a real in the range $(1, 1/e)$ and consider any language $L \in$ PT(n^f). By the last part of Lemma 2.3 and the second step, there exists an integer ℓ independent of L such that on an input x of length n we can generate in time $n^{1+o(1)}$ a Boolean formula φ such that (47) holds. By part (i) of the theorem, we can apply the first step $\ell - 1$ times to the Boolean formula and end up with a Boolean formula φ' such that

$$x \in L \Leftrightarrow (\mathsf{W}y')\varphi'(x, y'). \tag{48}$$

The whole process only takes time $n^{1+o(1)}$; in particular, φ' is of size $n^{1+o(1)}$. The right-hand side can be easily evaluated on a randomized machine with unbounded error that runs in time $n^{1+o(1)}$. Since L was an arbitrary language in PT(n^f), we conclude that PT$(n^f) \subseteq$ PT$(n^{1+o(1)})$, which contradicts Lemma 3.3 as $f > 1$.

The proof of the first step follows from the obliviousness of the majority formula produced by the proof of the last part of Lemma 2.3 since our hypothesis gives a PT$(n^{1+o(1)})$-algorithm to decide the left-hand side of (46).

All the meat is in the proof of the second step, which takes up the rest of this section. Given $f < 1/e$, there exists an integer g and a positive real ϵ such that PT$(n^f) \subseteq$ PTS$(n^g, n^{1-\epsilon})$. Consider a computation M of the latter type on an input x of length n. We assume that the computation is normalized such that the possible computation paths form a full binary tree.

The first step is to construct a uniform arithmetic formula of linear-exponential size that computes the number of accepting computation paths of M on input x. We can apply the divide-and-conquer strategy from Figure 3 to reduce counting the number of computation paths of length t from configuration C to configuration C', to instances of the same problem but for paths of length t/b – just replace the existential quantifier in (2) by a summation over all possible choices of the configurations C_i, and the universal quantifier by a product over all choices of i. Setting $t = n^g$ and applying this strategy $k = g/\epsilon$ times recursively with $b = n^\epsilon$, we end up with trivial counting problems for paths of length 1. The number of accepting computation paths on input x can be expressed as an arithmetic formula with $2k$ alternating layers of addition gates with fanin 2^n (corresponding to all sequences of b configurations of size $n^{1-\epsilon}$ each) and multiplication gates with

49

fanin b. The formula is of size no more than 2^{2kn} and is logtime-uniform. The values of the leaves can be computed in time almost-linear in $n^{1-\epsilon}$, so in time linear in n.

The next step is to transform this arithmetic formula into a uniform constant-depth threshold circuit of linear-exponential size that computes the same number on the same inputs. We can do so by replacing each addition and multiplication gate with logtime-uniform constant-depth threshold circuitry. The existence of such circuitry is folklore in the case of addition gates; for multiplication gates this was shown by Hesse [Hes01, HAB02], building on earlier work by Chiu [Chi95, CDL01]. AND, OR, and negation gates can be easily transformed into threshold gates, and the latter into majority gates. We can also make sure the majority gates all have fanin exactly 2^m for some integer $m = O(n)$. As a result, we end up with a logtime-uniform circuit of size $2^{O(kn)}$ that only consists of majority gates with fanin 2^m and has depth $\ell - 1$, where $\ell = O(k)$ is a constant. The circuit outputs many bits but we are only interested in the bits that correspond to the two most significant bits of the output, as they allow us to determine whether at least half of the computation paths are accepting.

Finally, we transform the threshold circuit into a majority formula with free variables x. We can describe a path from the output gate to an input gate of the circuit by a sequence of binary strings $y_1, y_2, \ldots, y_{\ell-1}$, each of length m, where y_i describes which child we pick at level $i - 1$. Given a string $y = y_1 y_2 \ldots y_{\ell-1}$, we can determine the value of the input gate y leads to in nondeterministic time $O(n)$: guess the labels of the gates on the path, verify them in deterministic time $O(n)$ using the logtime uniformity of the circuit, and compute the value of the input gate the path leads to in deterministic time $O(n)$. This almost gives us a majority formula – we just need to replace the final calculation by a fixed majority formula with free variables x, and $y_1, y_2, \ldots, y_{\ell-1}$. Since $\mathrm{NT}(n) \subseteq \mathrm{PT}(n)$, we can do so by introducing an additional block of variables y_ℓ, applying the last part of Lemma 2.3, and exploiting the obliviousness of the Boolean formula φ it produces. This gives us the Boolean formula $\varphi(x, y_1, y_2, \ldots, y_\ell)$ we need in time $n^{1+o(1)}$.

8.2   Connection with Quantum Algorithms

In terms of power the quantum model is believed to lie between randomized machines with bounded error and randomized machines with unbounded error. This is well-known in the time-bounded setting. In the time- and space-bounded setting, Van Melkebeek and Watson recently showed how to efficiently simulate quantum machines on randomized machines with unbounded error. Once we have that result, the lower bounds of Theorem 8.1 readily translate into the lower bounds of Theorem 1.9.

We first need to say a few words about the model of quantum computation. We assume that the reader is familiar with the basic notions and notation of quantum computing [NC00, KSV02]. We will not review those. We do want to point out a few model-related aspects that become pertinent when we care about concrete time and space bounds.

There are a number of choices to be made. First, which elementary unitary operations with which transition amplitudes we can use. Second, there is the issue of intermediate measurements: are they allowed or can we only make a measurement at the end? Third, there are the spatial constraints on the qubits on which the elementary operations act: can we only operate on neighboring qubits or can we assume random access? Finally, there is the question of whether the elementary quantum operations that are performed should be oblivious to the input and only depend on the input length (as in ordinary circuits), or can depend on the input but be oblivious to the computa-

tion path (as in circuits that are generated based on the input), or can depend on the computation path (as in Turing machines).

For the purpose of proving meaningful lower bounds, we would like our model to be as powerful as possible. The standard model of a quantum Turing machines [BV97] does a good job but only accommodates measurements at the end. Intermediate measurements are handled by introducing ancilla qubits that are entangled with the measured qubit. The entanglement prevents computation paths with different measurement outcomes from interfering and therefore allows us to postpone the measurement till the end. However, the ancilla qubits take up space and can deteriorate the space complexity of the algorithm to the extent that it becomes of the same order as its time complexity. This is detrimental for establishing time-space lower bounds. Let us also point out that important quantum algorithms like Shor's do make intermediate measurements, as does the simulation of randomized machines with bounded error on quantum machines. In particular, if we want $\mathrm{BPTS}(t, s) \subseteq \mathrm{BQTS}(t, s)$ to hold, we need to allow intermediate measurements.

Alternately, we can use quantum circuits consisting of elementary unitary operations from a universal set and measurement operators that act on a quantum register of size depending on the length of the input. This is an appealing model for several reasons, including the fact that algorithms like Shor's are oblivious and can be naturally expressed in it. The model also allows us to measure time and space in a simple way, namely as the total number of operations and the number of qubits in the register. These measures do not account for the resources needed to generate the circuit, though. If we care about concrete time and space bounds, we need to make sure that no complexity is hidden in the generation of the circuit. We will do so by insisting that the time and space needed to generate the circuit do not dominate the size and width, respectively.

Definition 8.4 *For all bounds t and s, we define $\mathrm{BQTS}(t, s)$ as the class of languages decided by bounded-error quantum circuits consisting of $O(t)$ gates of type CNOT, H, G, $G^\dagger$, M, and Q that act on a quantum register consisting of $O(s)$ qubits and such that the type and operands of the ith gate on a given input x can be generated in simultaneous time $t^{o(1)}$ and space $O(s)$. CNOT denotes the two-qubit gate that performs a controlled not, H represents the Hadamard gate, G is the one-qubit unitary gate with transition matrix*

$$\begin{bmatrix} 1 & 0 \\ 0 & \frac{3}{5} + \frac{4}{5}\sqrt{-1} \end{bmatrix},$$

$G^\dagger$ denotes the adjoint of G, M is a one-qubit measurement gate, and Q denotes the unitary query operator acting as follows on $\lceil \log n \rceil + 1$ qubits: $|i, b\rangle \to |i, b \oplus x_i\rangle$, where x_i denotes the ith bit of the input x.

The key points about the model are that it allows intermediate measurements and random access to the input. The other details are less important. In particular, the specific universal set consisting of CNOT, H, G and $G^\dagger$ is chosen to convenience the derivation below but is irrelevant. The Kitaev-Solovay Theorem [Kit97] shows that we can use any finite universal set that is closed under adjoint and such that the transition amplitudes can be computed up to $\log t$ bits of accuracy in time $t^{o(1)}$ and space $O(s)$ – we can simulate such computations in our model with only a subpolynomial cost in time and a constant factor cost in space.

Now that we have our model in place, we can prove the following simulation.

Lemma 8.5 (Van Melkebeek-Watson [vMW07]) *For all bounds t and s*

$$\mathrm{BQTS}(t,s) \subseteq \mathrm{PTS}(t^{1+o(1)}, s + \log t).$$

Lemma 8.5 combined with Theorem 8.1 immediately yields Theorem 1.9. The proof of Lemma 8.5 is a modification of a simulation due to Adleman et al. [ADH97]. The latter authors show that $\mathrm{BQT}(t) \subseteq \mathrm{PT}(t^{1+o(1)})$ by constructing nondeterministic machines in $\mathrm{NT}(t^{1+o(1)})$ whose accepting computation paths correspond to specific types of computation paths through the quantum circuits. Their construction does not handle intermediate measurements adequately as the authors assume that all measurements are postponed using ancilla qubits. Van Melkebeek and Watson show how to accommodate intermediate measurements in such a way that the nondeterministic machines run in $\mathrm{NTS}(t^{1+o(1)}, s)$.

In order to prove Lemma 8.5, we construct a certain type of computation tree of the circuit on a given input x of length n. There is a level in the tree corresponding to every gate and we label every node with the configuration that the computation path from the root to that node leads to. Note that there can be multiple nodes at a given level with the same configuration label – this is what enables quantum interference. We would like to set up the tree in such a way that the products of the amplitudes along paths in the tree that are consistent with a prescribed sequence μ of measurement outcomes can only take on a constant number of values. In order to achieve that, we exploit the special transition amplitudes of our chosen universal set of unitary gates, and we multiplicate nodes compared to the natural computation tree. The nodes at an H-level all have two children, corresponding to the two possible base values of the qubit after the operation of the gate. We label the edges with the corresponding gate transition amplitudes. At a G-level, we introduce 5 children with edges labeled $1/5$ in the case the qubit is zero in the configuration of the node; otherwise we introduce 7 children of which 3 are connected with an edge labeled $1/5$ and the other 4 are connected with an edge labeled $\sqrt{-1}/5$. A $\mathrm{G}^\dagger$-level is treated in a similar way. At a CNOT-, M-, and Q-level, every node has only one child; the one edge is labeled 1. Note that this construction realizes the goal we set out, namely that the transition amplitudes of the paths can only take on a constant number of values. Given a path π from the root to a leaf, let us denote by $\alpha(\pi)$ the product of all the edge labels on that path. We can write $\alpha(\pi)$ as

$$\alpha(\pi) = \frac{a(\pi)}{\sqrt{2}^{h(x)} 5^{g(x)}},$$

where $h(x)$ denotes the number of H-levels in the tree, and $g(x)$ denotes the number of G- and $\mathrm{G}^\dagger$-levels, and $a(\pi)$ only takes on the values of the four complex roots of unity, namely $\zeta \in \{1, -1, \sqrt{-1}, -\sqrt{-1}\}$.

Let μ denote a fixed binary sequence. The probability that the ith measurement gate outputs the bit μ_i for every i can be written as

$$\sum_\gamma \left| \sum_{\pi \sim (\mu, \gamma)} \alpha(\pi) \right|^2,$$

where γ ranges over all possible configurations and $\pi \sim (\mu, \gamma)$ denotes that the measurement outcomes on the path π are as prescribed in μ and that the path ends in a leaf that is labeled

with the configuration γ. We can assume without loss of generality that the last measurement determines acceptance. Thus, we can write the probability of acceptance as

$$\frac{1}{2^{h(x)}5^{2g(x)}} \cdot \underbrace{\sum_{\mu \text{ ending in } 1} \sum_{\gamma} \underbrace{\left| \sum_{\pi \sim (\mu,\gamma)} a(\pi) \right|^2}_{(*)}}_{(**)}.$$

Let us denote by $P_\zeta(x, \mu, \gamma)$ the number of paths π on input x that satisfy $\pi \sim (\mu, \gamma)$. We can construct a nondeterministic machine N that counts $P_\zeta(x, \mu, \gamma)$ as follows: on input $\langle x, \mu, \gamma, \zeta \rangle$, N runs through the computation tree on input x, checking for consistency with μ in case of a measurement, and selecting an edge arbitrarily if it has a choice; N keeps track of the value $a(\pi)$ of the path it has constructed thus far and of the current configuration; N accepts iff all the consistency checks with μ are passed, the final configuration is γ, and $a(\pi) = \zeta$ at that point. Note that $\#N(x, \mu, \gamma, \zeta) = P_\zeta(x, \mu, \gamma)$, where $\#N(y)$ denotes the number of accepting computation paths of the nondeterministic machine N on input y. Given the uniformity conditions of the quantum circuit, the machine N runs in time $O(t^{1+o(1)})$ and space $O(s + \log t)$. It needs to access each bit of μ only once and accesses them in order. Since $a(\pi)$ can only take on the values $\zeta \in \{1, -1, \sqrt{-1}, -\sqrt{-1}\}$, we can rewrite $(*)$ as

$$|P_1(x, \mu, \gamma) - P_{-1}(x, \mu, \gamma) + \sqrt{-1}P_{\sqrt{-1}}(x, \mu, \gamma) - \sqrt{-1}P_{-\sqrt{-1}}(x, \mu, \gamma)|^2$$
$$= \left(\sum_{\zeta^4 = 1} P_\zeta(x, \mu, \gamma)^2 \right) - 2 \left(P_1(x, \mu, \gamma)P_{-1}(x, \mu, \gamma) + P_{\sqrt{-1}}(x, \mu, \gamma)P_{-\sqrt{-1}}(x, \mu, \gamma) \right).$$

As a result, we can write $(**)$ as

$$\sum_{\zeta^4 = 1} S_{\zeta,\zeta}(x) - 2(S_{1,-1}(x) + S_{\sqrt{-1},-\sqrt{-1}}(x)), \tag{49}$$

where

$$S_{\zeta,\zeta'}(x) = \sum_{\mu \text{ ending in } 1} \sum_{\gamma} P_\zeta(x, \mu, \gamma)P_{\zeta'}(x, \mu, \gamma).$$

Now here is the crux: We can construct a nondeterministic machine O such that $\#O(x, \mu, \gamma, \zeta, \zeta') = P_\zeta(x, \mu, \gamma)P_{\zeta'}(x, \mu, \gamma)$ and such that $O(x, \mu, \gamma, \zeta, \zeta')$ runs in time $O(t^{1+o(1)})$ and space $O(s + \log t)$ and only needs to access each bit of μ once.

If we have such a machine O, we can create for each of the terms $S_{\zeta,\zeta'}$ in (49) a nondeterministic Turing machine in $\mathrm{NTS}(t^{1+o(1)}, s + \log t)$ such that the number of accepting computation paths of that machine on input x equals the corresponding term in (49). On input x, the machine simply guesses and stores γ and then runs $O(x, \mu, \gamma, \zeta, \zeta')$, guessing each bit of μ as needed by O without storing it. Note that the latter is made possible by the fact that O only needs to access each bit of μ once. Once we have all of those machines, we can combine them to obtain machines M_+ and M_- of the same complexity such that the probability of acceptance of x is given by

$$\frac{\#M_+(x) - \#M_-(x)}{2^{h(x)}5^{2g(x)}}.$$

Combined with the fact that the functions h and g can be computed in time $t^{1+o(1)}$ and space $O(s)$, we can construct a machine M in $\mathrm{NTS}(t^{1+o(1)}, s + \log t)$ that has its number of accepting computation paths offset by the right amount such that the fraction of accepting computation paths is at least half iff the quantum circuit accepts x. This finishes the proof of Lemma 8.5 modulo the construction of the machine O.

For the construction of O the usual strategy runs $N(x, \mu, \gamma, \zeta)$ followed by a run of $N(x, \mu, \gamma, \zeta')$ and accepts iff both runs accept. This yields the correct count but does not do the job – O would need to access each bit of μ twice, which would require the machines corresponding to each of the terms in (49) to store μ and thereby use too much space. However, we can execute the two runs *in parallel*, keeping them in synch such that both runs access the same bit of μ at the same point in time. This gives us the required machine O.

9 Future Directions

We have seen how the paradigm of indirect diagonalization from Section 3 allows us to derive time-space lower bounds for well-motivated problems on deterministic, nondeterministic, randomized, and quantum models with random access. In each setting we focused on the simplest computational problem for which we can establish nontrivial lower bounds; for those we presented the state-of-the-art results. The time lower bounds we obtained are still way below what we believe the true time complexity to be even without explicit space bounds. Obtaining nontrivial time lower bounds without space constraints seems beyond the reach of the ideas we presented, as the arguments critically rely on sublinear space bounds. Moreover, all the ingredients we use relativize and relative to the oracle of true fully quantified Boolean formulas all of the models we consider can be simulated deterministically in quasi-linear time. There is good hope for quantitative improvements in time-space lower bounds, though.

In the deterministic setting, there is a sense that a quadratic time lower bound is the best we can hope the paradigm to give for satisfiability on subpolynomial-space machines. Due to input size issues it is not obvious that such a bound can be reached using the ingredients we used. We have been able to alleviate some of those issues by exploiting special properties of the divide-and-conquer process that underlies the approach. There still seems to be room for further improvements. First, there is some slack in the arguments due to the fact that some of the intermediate results like Lemma 4.1 and Lemma 4.3 give improvements for time bounds up to quadratic. We also only analyzed the simplest cases of the alternation trading scheme, namely where we first introduce all alternations to obtain a speedup and then remove then in a linear fashion from the back to the front. We did extend our window during the alternation elimination phase from the last quantifier block to the last two blocks but that remains rather myopic. Moreover, there are more complicated processes than the linear ones we analyzed. They may do better. Williams [Wil07a] implemented an automated search for such processes by abstracting the properties of the two ingredients from Section 3 that have been used in the arguments so far. A computer can then exhaustively search for proofs with a given number of applications of the corresponding rules and keep track of the one that yields the strongest time lower bound for subpolynomial space. The output exhibits patterns that, when extrapolated, lead to the best results we currently have, but nothing better. This may be an indication that we need to exploit more of the special properties of the divide-and-conquer process, such as the fact that the universal quantifier in (7) only ranges over a logarithmic number of variables. The arguments to date only exploit that fact that the number of variables is linearly

bounded. What else can we do if satisfiability is easy that we can use in this context besides efficiently eliminating alternations?

The paradigm of indirect diagonalization also offers the possibility to build in nonrelativizing ingredients. Very few nonrelativizing results exist and they all derive from inclusion-type results like the PCP Theorem [ALM+98]. We can plug in those results when we are deriving more and more unlikely inclusions of complexity classes based on our contradiction hypothesis. Does that help?

The situation in the nondeterministic setting is similar. In particular, the slack in Lemma 5.1 suggests that we may be able to boost the exponent of the time lower bound for tautologies on nondeterministic subpolynomial-space machines up to the golden ratio.

The lower bound situation in the nonuniform setting is, of course, worse than in the uniform one. We know of no nontrivial lower bounds for satisfiability on unrestricted fully nonuniform circuits. We have seen how the indirect diagonalization paradigm can be used to obtain lower bounds for circuits with natural uniformity conditions that are considerably weaker than the corresponding circuit parameters. However, the size lower bounds are only slightly super-linear. As we pointed out in Section 1.1, we have much stronger lower bounds for satisfiability on fully nonuniform restricted types of circuits, such as constant-depth circuits, branching programs, and formulas. These results follow easily from known lower bounds for simple problems in nondeterministic quasi-linear time. Given that those models have some connection to space-bounded computation, it may be possible to use some of the ideas in this survey to capitalize on the computational power of satisfiability and improve the lower bounds for satisfiability on those models.

In the randomized setting with two-sided error, the simplest natural problem for which we have been able to establish nontrivial lower bounds is $\Sigma_2\text{SAT}$. The bounds match those that follow from the simplest argument in the deterministic setting but not the current deterministic record. Can we take advantage of the special properties of the divide-and-conquer process in the randomized setting as we did in the deterministic setting?

Apart from trying to match the deterministic bounds for the simplest problem for which we can get nontrivial randomized bounds, we would also like to handle simpler problems than $\Sigma_2\text{SAT}$, e.g., satisfiability. As we mentioned at the end of Section 7.2, we have made some progress toward satisfiability by translating the known lower bound for $\Sigma_2\text{T}(n)$ into a lower bound for $\text{MA}_1\text{T}(n)$, which lies between $\text{NT}(n)$ and $\Sigma_2\text{T}(n)$. The reason is that

$$\text{MA}_1\text{T}(n) \subseteq \text{BPTs}(n^d) \Rightarrow \Sigma_2\text{T}(n) \subseteq \text{BPTs}(n^{d2+o(1)}). \tag{50}$$

Since we know how to rule out the right-hand side of (50) for $d < \sqrt{2}$, we obtain nontrivial lower bounds for $\text{MA}_1\text{T}(n)$ on randomized machines with two-sided error. One may wonder why we cannot simply replace $\text{MA}_1\text{T}(n)$ in (50) by $\text{NT}(n)$ and that way obtain nontrivial lower bounds for satisfiability on randomized machines with two-sided error. After all, in the deterministic setting we know that $\text{NT}(n) \subseteq \text{DTs}(n^d)$ implies that $\Sigma_2\text{T}(n) \subseteq \text{DTs}(n^{d^2})$. However, in the time-bounded setting the best implication we know of that type is the following:

$$\text{NT}(n) \subseteq \text{BPT}(n^d) \Rightarrow \Sigma_2\text{T}(n) \subseteq \text{BPT}(n^{d(d+1)}), \tag{51}$$

which does not allow us to obtain any nontrivial lower bounds for $\text{NT}(n)$ based on a quadratic lower bound for $\Sigma_2\text{T}(n)$. The additional factor of n^d comes from the overhead in the best known simulations of Merlin-Arthur protocols by Arthur-Merlin protocols [BM88]. Suppose that $\text{NT}(n) \subseteq$

BPT(n^d), and consider a Σ_2T(n)-computation. Using the hypothesis, we can replace the computation after the existential phase by a BPT(n^d)-computation, resulting in an overall Merlin-Arthur protocol that guesses $O(n)$ bits and runs in time $O(n^d)$. We then transform the Merlin-Arthur protocol into an equivalent Arthur-Merlin protocol. As we discuss below, the best transformation we know blows up the running time by the number of bits Merlin guesses, i.e., we obtain an Arthur-Merlin protocol that runs in time $O(n^{d+1})$. Applying the hypothesis once more to the Merlin phase of the latter protocol results in a final BPT($n^{d(d+1)}$)-algorithm.

The most efficient way we know to transform a Merlin-Arthur protocol into an equivalent Arthur-Merlin protocol first reduces the error probability of the Merlin-Arthur protocol to less than one out of the number of possible messages Merlin can send to Arthur. Once the probability of convincing Arthur of an incorrect claim is so small, switching the parties' order does not give Merlin an unfair advantage. The error reduction requires taking the majority vote of a number of parallel trials which is of the order of the number of bits that Merlin sends to Arthur, resulting in such an overhead factor for the final Arthur-Merlin simulation. Other known transformations, e.g., based on extractors [GZ97] or on pseudorandom generators [NW94], involve overhead factors that are at least as large. In fact, Diehl [Die07] argued that every simulation of the above type that is blackbox – which all of the known simulations are – has to incur an overhead factor of the number of bits Merlin sends to Arthur. This suggest that we should either look for non-blackbox transformations or exploit the space bound in a substantial way in the transformation. As for the latter, Viola [Vio07] established a similar quadratic time lower bound for blackbox simulations of BPT(n) on Σ_2-machines, but we have seen in Section 7.1 that we can simulate BPTs(n) on Σ_2-machines in time $n^{1+o(1)}$.

In the quantum setting, the simplest problem for which we have nontrivial lower bounds is MajMajSAT. The result follows from a similar lower bound on randomized machines with unbounded error. One might hope that our paradigm would allow us to deduce lower bounds for the simpler problem MajSAT on randomized machines with unbounded error. In the quantum setting we have some tools at our disposal that may help. For example, we know that BQP is low for PP, i.e., PP$^{\text{BQP}} \subseteq$ PP [FR90]. Thus, an efficient quantum algorithm for MajSAT implies an efficient quantum algorithm for MajMajSAT since the latter is in

$$\text{PP}^{\text{PP}} \subseteq \text{PP}^{\text{MajSAT}} \subseteq \text{PP}^{\text{BQP}} \subseteq \text{PP} \subseteq \text{P}^{\text{MajSAT}} \subseteq \text{BQP}.$$

Unfortunately, the lowness result involves time-expensive amplification and the time bounds for MajMajSAT we can handle are only slightly super-linear so we cannot conclude lower bounds for MajSAT right away. There is hope, though.

An different problem we may be able to handle in the quantum setting is Mod$_2$SAT. A more ambitious goal would be to obtain lower bounds on quantum machines for Σ_kSAT or even for satisfiability. It seems unlikely that our paradigm can yield those without first showing how to efficiently simulate quantum computations in the polynomial-time hierarchy, which we currently do not know and about which there is not even a consensus in the community that such simulations should exist.

Finally, besides quantitative improvements in the lower bounds for relatively simple problems on various models of computation, there is another direction we can take: establishing qualitatively much stronger bounds for much harder problems. For example, in the deterministic setting it makes sense to consider problems that are as close as possible to linear space in power. For problems in linear space the space hierarchy theorem trivially implies time lower bounds on sublinear-space

algorithms – no amount of time will suffice – so it does not make sense to consider those. Good candidate problems include MajSAT and $\mathrm{Mod_2SAT}$. By Toda's Theorem [Tod91], $\Sigma_k\mathrm{SAT}$ efficiently reduces to $\mathrm{Mod_2SAT}$ under randomized reductions and to MajSAT under deterministic reductions. In Section 4.2 we stated a time lower bound that lies somewhere between n^k and n^{k+1} for $\Sigma_k\mathrm{SAT}$ on subpolynomial-space machines. This suggests an approach for establishing superpolynomial time lower bounds for algorithms that solve $\mathrm{Mod_2SAT}$ or MajSAT in subpolynomial space.

A critical question is whether the reductions from $\Sigma_k\mathrm{SAT}$ to $\mathrm{Mod_2SAT}$ and MajSAT are sufficiently efficient. Recall that Toda's Theorem consists of two steps, namely (i) the inclusion of the polynomial-time hierarchy in $\mathrm{BPP^{Mod_2SAT}}$ and (ii) the inclusion of the latter class in $\mathrm{P^{PP}}$. The first step seems to be the bottleneck for our purposes. Say that the first step shows that

$$\Sigma_k\mathrm{T}(n) \subseteq \mathrm{BPT}(n^{a_k})^{\mathrm{Mod_2SAT}} \tag{52}$$

for some constants a_k. Given the above lower bounds for $\Sigma_k\mathrm{SAT}$ we need a_k to be roughly under k to have any hope in succeeding. Moreover, having $a_k = o(k)$ suffices to realize our goal. This is because (52) and the hypothesis that MajSAT has a $\mathrm{DTs}(n^d)$-algorithm lead to the inclusions

$$\Pi_k\mathrm{T}(n^k) \subseteq \mathrm{BPT}(n^{ka_k})^{\mathrm{Mod_2SAT}} \subseteq \mathrm{DTs}(n^{ka_k(d+1)d+o(1)}) \subseteq \Sigma_k\mathrm{T}(n^{a_k(d+1)d+o(1)}). \tag{53}$$

The second inclusion follows by applying the hypothesis twice based on the fact that MajSAT allows us to efficiently compute $\mathrm{Mod_2SAT}$ using binary search, and to simulate randomized computations with two-sided error. The last inclusion follows from the simple speedup mentioned at the beginning of Section 4.2. The resulting inclusion (53) contradicts Lemma 3.1 as long as $a_k(d+1)d < k$, which happens for every d and large enough k provided $a_k = o(k)$.

Toda's original construction [Tod91] yields $a_k = \Theta(2^k)$. It iterates a quadratic randomized reduction from satisfiability to $\mathrm{Mod_2SAT}$ based on the Valiant-Vazirani randomized pruning procedure [VV86]. If we replace that reduction by a quasi-linear variant [NRS95], we can reduce a_k to $k + o(1)$. Other proofs that yield about the same value for a_k exist (see [KVVY93], for example). Reducing a_k further seems to require getting around the same amplification bottleneck that underlies the efficient transformation of Merlin-Arthur into equivalent Arthur-Merlin protocols as discussed earlier in this section. Using operator notation, the variant of Toda's construction yields the inclusion $\Sigma_k\mathrm{T}(n) \subseteq (\mathrm{BP} \cdot \oplus)^k \cdot \mathrm{DT}(n^{1+o(1)})$. Whereas before we had to efficiently swap Arthur and Merlin operators, here we need to efficiently swap BP-operators and parity-operators ($\oplus$) so as to end up with a final $\mathrm{BP} \cdot \oplus$-computation. We know how to do each swap at the cost of a factor n in running time using amplification. The question is how to do better.

Acknowledgments

References

[ADH97]   L. Adleman, J. DeMarrais, and M. Huang. Quantum computability. *SIAM Journal on Computing*, 26:1524–1540, 1997.

[Ajt99]   M. Ajtai. A non-linear time lower bound for Boolean branching programs. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science*, pages 60–70. IEEE, 1999.

[AKR+01]   E. Allender, M. Koucky, D. Ronneburger, S. Roy, and V. Vinay. Time-space tradeoffs in the counting hierarchy. In *Proceedings of the 16th IEEE Conference on Computational Complexity*, pages 295–302. IEEE, 2001.

[ALM+98]   S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998.

[Bab85]   L. Babai. Trading group theory for randomness. In *Proceedings of the 17th ACM Symposium on the Theory of Computing*, pages 421–429. ACM, 1985.

[BM88]   L. Babai and S. Moran. Arthur-Merlin games: a randomized proof system, and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, 36:254–276, 1988.

[BS90]   R. Boppana and M. Sipser. Complexity of finite functions. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 758–804. MIT Press, 1990.

[BSSV03]   P. Beame, M. Saks, X. Sun, and E. Vee. Time-space trade-off lower bounds for randomized computation of decision problems. *Journal of the ACM*, 50(2):154–195, 2003.

[BV97]   E. Bernstein and U. Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 26:1411–1473, 1997.

[CDL01]   A. Chiu, G. Davida, and B. Litow. Division in logspace-uniform NC^1. *Theoretical Informatics and Applications*, 35:259–276, 2001.

[Chi95]   A. Chiu. Complexity of parallel arithmetic using the Chinese remainder representation. Master's thesis, University of Wisconsin-Milwaukee, 1995.

[CLRS01]   T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, second edition, 2001.

[Coo71]   S. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd ACM Symposium on the Theory of Computing*, pages 151–158. ACM, 1971.

[Coo73]   S. Cook. A hierarchy theorem for nondeterministic time complexity. *Journal of Computer and System Sciences*, 7:343–353, 1973.

[Coo88]   S. Cook. Short propositional formulas represent nondeterministic computations. *Information Processing Letters*, 26:269–270, 1988.

[CW89]   A. Cohen and A. Wigderson. Dispersers, deterministic amplification, and weak random sources. In *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science*, pages 14–19. IEEE, 1989.

[Die07]   S. Diehl. Lower bounds for swapping Arthur and Merlin. In *Proceedings of the 11th International Workshop on Randomized Techniques in Computation*, pages 449–463. Springer-Verlag, 2007.

[DvM06]   S. Diehl and D. van Melkebeek. Time-space lower bounds for the polynomial-time hierarchy on randomized machines. *SIAM Journal on Computing*, 36:563–594, 2006.

[DvMW07] S. Diehl, D. van Melkebeek, and R. Williams. A new time-space lower bound for nondeterministic algorithms solving tautologies. Technical Report 1601, Department of Computer Sciences, University of Wisconsin-Madison, 2007.

[FLvMV05] L. Fortnow, R. Lipton, D. van Melkebeek, and A. Viglas. Time-space lower bounds for satisfiability. *Journal of the ACM*, 52:835–865, 2005.

[For00]   L. Fortnow. Time-space tradeoffs for satisfiability. *Journal of Computer and System Sciences*, 60:337–353, 2000.

[FR90]   L. Fortnow and J. Rogers. Complexity limitations on quantum computations. *Journal of Computer and System Sciences*, 59:240–252, 1990.

[FvM00]   L. Fortnow and D. van Melkebeek. Time-space tradeoffs for nondeterministic computation. In *Proceedings of the 15th IEEE Conference on Computational Complexity*, pages 2–13. IEEE, 2000.

[GG81]   O. Gabber and Z. Galil. Explicit constructions of linear-sized superconcentrators. *Journal of Computer and System Sciences*, 22:407–420, 1981.

[GZ97]   O. Goldreich and D. Zuckerman. Another proof that BPP $\subseteq$ PH (and more). Technical Report TR-97-045, Electronic Colloquium on Computational Complexity, 1997.

[HAB02]   W. Hesse, E. Allender, and D. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *Journal of Computer and System Sciences*, 65:695–712, 2002.

[Has98]   J. Hastad. The shrinkage exponent of de Morgan formulas is 2. *SIAM Journal on Computing*, 27:48–64, 1998.

[Hes01]   W. Hesse. Division is in uniform TC^0. In *Proceedings of the 28th International Colloquium On Automata, Languages and Programming*, pages 104–114. Springer-Verlag, 2001.

[HS66]   F. Hennie and R. Stearns. Two-tape simulation of multitape Turing machines. *Journal of the ACM*, 13:533–546, 1966.

[INW94]   R. Impagliazzo, N. Nisan, and A. Wigderson. Pseudorandomness for network algorithms. In *Proceedings of the 26th ACM Symposium on the Theory of Computing*, pages 356–364. ACM, 1994.

[IZ89]   R. Impagliazzo and D. Zuckerman. How to recycle random bits. In *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science*, pages 248–253. IEEE, 1989.

[Kan84]   R. Kannan. Towards separating nondeterminism from determinism. *Mathematical Systems Theory*, 17:29–45, 1984.

[Kit97]   A. Kitaev. Quantum computations: Algorithms and error correction. *Russian Mathematical Surveys*, 52:1191–1249, 1997.

[KL82]   R. Karp and R. Lipton. Turing machines that take advice. *L'Enseignement Mathématique*, 28(2):191–209, 1982. A preliminary version appeared in STOC 1980.

[KSV02]   A. Kitaev, A. Shen, and M. Vyalyi. *Classical and Quantum Computation*. American Mathematical Society, 2002.

[KVVY93]   R. Kannan, H. Venkateswaran, V. Vinay, and A. Yao. A circuit-based proof of Toda's theorem. *Information and Computation*, 104:271–276, 1993.

[Lau83]   C. Lautemann. BPP and the polynomial hierarchy. *Information Processing Letters*, 17:215–217, 1983.

[LV99]   R. Lipton and A. Viglas. On the complexity of SAT. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science*, pages 459–464. IEEE, 1999.

[Mar73]   G. Margulis. Explicit construction of concentrators. *Problems of Information Transmission*, 9:325–332, 1973.

[vM04]   D. van Melkebeek. Time-space lower bounds for NP-complete problems. In G. Paun, G. Rozenberg, and A. Salomaa, editors, *Current Trends in Theoretical Computer Science*, pages 265–291. World Scientific, 2004.

[vMP07]   D. van Melkebeek and K. Pervyshev. A generic time hierarchy for semantic models with one bit of advice. *Computational Complexity*, 16:139–179, 2007.

[vMW07]   D. van Melkebeek and T. Watson. A quantum time-space lower bound for the counting hierarchy. Technical Report 1600, Department of Computer Sciences, University of Wisconsin-Madison, 2007.

[NC00]   M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[Nep70]   V. Nepomnjascii. Rudimentary predicates and Turing calculations. *Soviet Mathematics–Doklady*, 11:1462–1465, 1970.

[Nis92]   N. Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12:449–461, 1992.

[Nis94]   N. Nisan. RL ⊆ SC. *Computational Complexity*, 4:1–11, 1994.

[NRS95]   A. Naik, K. Regan, and D. Sivakumar. On quasilinear-time complexity theory. *Theoretical Computer Science*, 148:325–349, 1995.

[NW94]   N. Nisan and A. Wigderson. Hardness vs. randomness. *Journal of Computer and System Sciences*, 49:149–167, 1994.

[PF79]   N. Pippenger and M. Fischer. Relations among complexity measures. *Journal of the ACM*, 26:361–381, 1979.

[Rob91]   J. Robson. An $O(T \log T)$ reduction from RAM computations to satisfiability. *Theoretical Computer Science*, 82:141–149, 1991.

[Ruz80]   W. Ruzzo. Tree-size bounded alternation. *Journal of Computer and System Sciences*, 21:218–235, 1980.

[Sav70]   W. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4:177–192, 1970.

[SFM78]   J. Seiferas, M. Fischer, and A. Meyer. Separating nondeterministic time complexity classes. *Journal of the ACM*, 25:146–167, 1978.

[Sip83]   M. Sipser. A complexity theoretic approach to randomness. In *Proceedings of the 15th ACM Symposium on the Theory of Computing*, pages 330–335. ACM, 1983.

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

[Tor91]   J. Toran. Complexity classes defined by counting quantifiers. *Journal of the ACM*, 38:753–774, 1991.

[Tou01]   I. Tourlakis. Time-space lower bounds for SAT on nonuniform machines. *Journal of Computer and System Sciences*, 63(2):268–287, 2001.

[Vio07]   E. Viola. On approximate majority and probabilistic time. In *Proceedings of the 22nd IEEE Conference on Computational Complexity*, pages 155–168. IEEE, 2007.

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

[Wil05]   R. Williams. Better time-space lower bounds for sat and related problems. In *Proceedings of the 20th IEEE Conference on Computational Complexity*, pages 40–49. IEEE, 2005.

[Wil07a]   R. Williams. *Algorithms and Resource Requirements for Fundamental Problems*. PhD thesis, Carnegie Mellon Univesity, 2007.

[Wil07b]   R. Williams. Time-space tradeoffs for counting NP solutions modulo integers. In *Proceedings of the 22nd IEEE Conference on Computational Complexity*, pages 70–82. IEEE, 2007.

[Yap83]   C. Yap. Some consequences of non-uniform conditions on uniform classes. *Theoretical Computer Science*, 26:287–300, 1983.

[Zak83]   S. Zak. A Turing machine time hierarchy. *Theoretical Computer Science*, 26:327–333, 1983.