Gap-Languages and Log-Time Complexity Classes

Kenneth W. Regan[*]
State University of New York at Buffalo

Heribert Vollmer[†]
Universität Würzburg

April 1996

Abstract

This paper shows that classical results about complexity classes involving "delayed diagonalization" and "gap languages," such as Ladner's Theorem and Schöning's Theorem and independence results of a kind noted by Schöning and Hartmanis, apply at very low levels of complexity, indeed all the way down in Sipser's log-time hierarchy. This paper also investigates refinements of Sipser's classes and notions of log-time reductions, following on from recent work by Cai, Chen, and others.

1 Introduction

Many theorems about the structure of familiar complexity classes such as P, NP, and PSPACE have been obtained by a technique called *delayed diagonalization* [Lad75, CM81, Sch82, MY85, Amb85a] (see also [BDG88]). For instance, there are languages E in PSPACE such that E is not in LOGSPACE and E is not PSPACE-complete under log-space reductions ($\leq_m^{log}$). Moreover, the structure of such languages E under $\leq_m^{log}$ embeds all countable partial orders. On hypothesis NP $\neq$ P, there is a similar rich structure of languages between those in P and those that are NP-complete, under polynomial-time many-one reducibility ($\leq_m^p$) as well as under $\leq_m^{log}$. The delayed diagonalization technique is also used to obtain independence results from strong formal systems $\mathcal{F}$ that are sound and axiomatizable, such as Peano Arithmetic (PA) or set theory (ZF). For example, for any sound, axiomatizable $\mathcal{F}$, there are languages $E \in$ PSPACE $\setminus$ LOGSPACE such that for all Turing machines M accepting E, $\mathcal{F}$ cannot prove the first-order arithmetical sentence '$L(M) \notin$ LOGSPACE.' Schöning [Sch82] observed an analogous result for unprovable non-membership of NP languages in P, on hypothesis NP $\neq$ P. For further results of this type, see Hartmanis [Har85] and also [KOR87, Reg88].

[*]Supported in part by the National Science Foundation under Grant CCR-9409104. Author's current address: Computer Science Department, 226 Bell Hall, UB North Campus, Buffalo, NY 14260-2000. Email: *regan@cs.buffalo.edu*

[†]Supported in part by an Alexander von Humboldt fellowship, while the author held a visiting position at the University of California at Santa Barbara. Author's current address: Theoretische Informatik, Universität Würzburg, Am Exerzierplatz 3, D-97072, Germany. Email: *vollmer@informatik.uni-wuerzburg.de*.

The languages E constructed above are commonly known as "gap languages." To determine which language classes $\mathcal{C}$ admit construction of such "gap languages," Schmidt [Sch85] formulated a definition of $\mathcal{C}$ being *recursive gap closed*. Schmidt showed that the class L_1 of languages accepted by log-space Turing machines whose input tape is *one-way*, which is a proper subclass of LOGSPACE, is recursive gap closed. Regan [Reg88, Reg92a] generalized the notion of a "gap language" to a function h from Σ^* to natural numbers, so as to extend the main theorem of [Sch82] from diagonalization over two to infinitely many classes, and showed that such functions h can be computed by log-space bounded machines that run in *real time*, i.e., where the input head moves right in every step. Vollmer [Vol90] showed that "gap languages" can be constructed in DLOGTIME. This answered an open question in [Reg92a] about Immerman's class FO, which is currently regarded as the best notion of "uniform" AC^0 [Imm87, BIS90], since DLOGTIME is contained in FO [BIS90]. This also implies that any class that is closed under DLOGTIME many-one reductions, including FO and nonuniform AC^0, is recursive gap closed.

This paper extends the main result in Vollmer [Vol90] from two to infinitely many classes. We give further applications for diagonalization and independence results in low-level complexity classes. We construct languages $E \in \mathrm{NC}^1$ that are not in the *log-time hierarchy* of [CKS81, Sip83], but that are not hard for any level above DLOGTIME either. Indeed, E does not give any "computational help" to these levels. Moreover, given a formal system $\mathcal{F}$, E can be constructed so that the assertions "E is in the log-time hierarchy" and "E is NC^1-complete" are not disprovable by $\mathcal{F}$.

A second contribution of this paper is the study of classes within the log-time hierarchy, together with some notions of "DLOGTIME many-one reductions" that are sharper than the one standardly defined and used in [CKS81, Bus87, Tor88, BIS90, JMT94]. The standard one is not transitive and does not preserve membership in individual levels of the hierarchy. Ours, which extend a suggestion of Cai and Chen [CC95] on how to define log-time languages, remedy these lacks and seem to suffice for most applications of DLOGTIME reductions and DLOGTIME uniformity in the literature.

2 How to define log time?

To define Turing machines M that operate in logarithmic time, the basic idea is to give M "random-access" to its input x via a special *index tape*. The index tape has alphabet $\Sigma = \{0, 1\}$ and encodes a natural number as a string using the bijection *str* defined by $str(0) = \lambda$, $str(1) = 0$, $str(2) = 1$, $str(3) = 00$, $str(4) = 01$, and so on. We suppose that the length n of x is initially given on a worktape designated as an "auxiliary input tape" (or on the index tape itself). When M enters a special query state $q_?$, M receives in response bit x_a of the input x, where a is the number currently on the index tape. The index tape is not erased by such an operation. Here is where several authorities diverge, as summarized by Cai and Chen [CC95]:

($\mathcal{U}$) (for "unrestricted") M may enter $q_?$ at any time, and is charged 1 time unit for the query.

($\mathcal{S}$) (for "Sipser") M is charged $\lceil \log n \rceil$ time units for the query.

($\mathcal{R}$) (for "Ruzzo") M may enter $q_?$ only once in any computation path; without loss of generality at the end of the path.

Proviso ($\mathcal{U}$) was used in the seminal paper by Chandra, Kozen, and Stockmeyer [CKS81], and is regarded as "standard" after uses in [Bus87, Tor88, BIS90, JMT94] and others. An observation credited to [Dow86] in [Bus87] and noted also in [Tor88] is that it is unnecessary to provide n to the machine: if one allows that queries to a outside the range $[0 \dots n-1]$ return a special symbol '\$', then M can calculate n in time $O(\log n)$ by binary search. Proviso ($\mathcal{S}$) is equivalent to Sipser's stipulation in [Sip83, BS90] that all addresses are encoded by binary strings of length $\lceil \log n \rceil$, and after each query, the index tape is indeed erased. Proviso ($\mathcal{R}$) was defined by Ruzzo [Ruz81] for alternating machines.

A fourth proviso, intermediate in power between $\mathcal{U}$ and $\mathcal{S}$, has also been defined by Cai and Chen et al. [CCDF94, CCDF95, CC95], taking an idea from the "Block Transfer" model of [ACS87]:

($\mathcal{B}$) (for "block read/write") M writes two addresses i, j with $i \leq j$ on its index tape, and receives the string $x_i \dots x_j$ on that tape, at a cost of $\lceil \log n \rceil + (j - i)$ time units.

This is equivalent to the "$\mathcal{B}_c$" formalism in [CC95]. In order for M to run in $O(\log n)$ time, all intervals $[i \dots j]$ addressed must have $O(\log n)$ size. The additive $\lceil \log n \rceil$ term, analogous to proviso ($\mathcal{S}$), ensures that only constant-many such queries can be made along any one computation path.

Still following Cai and Chen, we write $\Sigma_k^{\mathcal{U}}$, $\Sigma_k^{\mathcal{S}}$, $\Sigma_k^{\mathcal{R}}$, and $\Sigma_k^{\mathcal{B}}$ for the classes of languages accepted by Σ_k-alternating Turing machines under the above four provisos, respectively. We write $\Pi_k^{\mathcal{U}}$ for $\{ L : \widetilde{L} \in \Sigma_k^{\mathcal{U}} \}$, $\Delta_k^{\mathcal{U}}$ for $\Sigma_k^{\mathcal{U}} \cap \Pi_k^{\mathcal{U}}$, and similarly for the other provisos. By definition, $\Sigma_0^{\mathcal{U}} = \Pi_0^{\mathcal{U}} = \Delta_0^{\mathcal{U}}$, etc. We write DLOGTIME for $\Delta_0^{\mathcal{U}}$, NLOGTIME for $\Sigma_1^{\mathcal{U}}$, DLT for $\Delta_0^{\mathcal{B}}$, and Σ_k^{dlt} for $\Sigma_k^{\mathcal{B}}$. The first two proper containments in the following are easy to see:

$$\Delta_0^{\mathcal{R}} \subset \Delta_0^{\mathcal{S}} \subset \text{DLT} \subset \text{DLOGTIME}. \tag{1}$$

For the last, let L_{bs} be the language of strings x, of length $2^d - 1$ for some $d \geq 0$, in which the following binary search ends at a '1': Start with the first bit; call it x_1. If $x_1 = 0$ then go to x_2, else go to x_3. At any bit a, if $x_a = 0$ then go to x_{2a}, else go to x_{2a+1}. Stop when $2^{d-1} \leq a \leq 2^d - 1$, and accept iff $x_a = 1$.

Proposition 2.1 *L_{bs} belongs to* DLOGTIME *but not to* DLT.

Proof. Membership in DLOGTIME follows via the algorithm that defines the language. Now suppose M is a DLT-machine that accepts L_{bs} in time $k \log n$, for some fixed $k \geq 0$. Consider the operation of M on input $x_0 = 0^n$, for sufficiently large n of the form $2^d - 1$. M rejects x_0. Call the bit sets $\{1\}$, $\{2, 3\}$, $\{4, 5, 6, 7\}$,..., $\{2^{d-1}, \dots, 2^d - 1\}$ *levels*, and the last, the *bottom level*. Say that M on input x_0 *examines* a level if it reads at least one bit in that level.

Then the number of examined levels is at most $\log(k \log n) + 2k$, because the first $k \log n$ bits can be regarded as read "for free," and then since all other levels have size at least $k \log n$, at most two of those can be examined per block read. Hence the number m of unexamined levels is $\Omega(\log n)$. Also note that the DLT machine M reads at most $k \log n$

bits in the bottom level on input x_0. Now by changing bits of x_0 in the m unexamined levels, we can create 2^m different strings such that the binary searches on those strings end at 2^m distinct bits in the bottom level. Since $m > k \log n$, at least one such string x' gives the same computation by M as on x_0. But then changing the target bit of x' to '1' yields a member of L_{bs} that M rejects, giving the desired contradiction. □

In fact, this argument shows that a machine accepting L_{bs} under proviso $\mathcal{B}$ must take time $\Omega(\log^2 n)$.

Cai, Chen, and Håstad [CCH95] show that for all $k \geq 1$,

$$\Sigma_k^{\mathcal{R}} \subset \Sigma_k^{\mathcal{S}} \subset \Sigma_k^{\mathcal{B}} \subseteq \Sigma_k^{\mathcal{U}} \subset \Sigma_{k+1}^{\mathcal{R}}, \tag{2}$$

thereby refuting Sipser's claim [Sip83, BS90] that $\Sigma_k^{\mathcal{R}} = \Sigma_k^{\mathcal{S}}$. (That $\Sigma_k^{\mathcal{S}} \subset \Sigma_k^{\mathcal{B}}$ falls out of their stated proof of $\Pi_k^{\mathcal{S}} \subset \Pi_k^{\mathcal{U}}$; we suspect that $\Sigma_k^{\mathcal{B}} \subset \Sigma_k^{\mathcal{U}}$ can be shown by combining their techniques with the binary-search idea above.) Thus the provisos give different classes at all individual levels of the logarithmic time hierarchy, though the class LOGH $=_{\text{def}} \cup_k \Sigma_k^{\mathcal{U}}$ is the same under all.

Our main motivation for interest in the three classes $\Delta_0^{\mathcal{R}}$, $\Delta_0^{\mathcal{S}}$, and DLT is that they have reducibility relations associated to them that remedy some major defects of DLOGTIME reductions.

Definition 2.1. Given any two languages A and B, we write, respectively, (a) $A \leq_m^{\mathcal{U}} B$, (b) $A \leq_m^{dlt} B$, (c) $A \leq_m^{\mathcal{S}} B$, (d) $A \leq_m^{\mathcal{R}} B$, if there is a function $f : \Sigma^* \to \Sigma^*$ that many-one reduces A to B, and a deterministic log-time TM M that computes f in the following respective manner:

(a) M runs under proviso $\mathcal{U}$, and on any input x and auxiliary input n, j, M outputs bit j of $f(x)$, together with the length $|f(x)|$ of $f(x)$.

(b) M runs under proviso $\mathcal{B}$, and on any input x and auxiliary input n, i, j, where $j - i = O(\log n)$, M outputs bits i through j of $f(x)$, together with $|f(x)|$.

(c) M runs under proviso $\mathcal{S}$, and otherwise behaves as in (a).

(d) M runs under proviso $\mathcal{R}$, and otherwise behaves as in (a).

Here (a) is equivalent to the standard definition of f being a DLOGTIME reduction, which is that the language

$$A_f := \{ (x, i, b) : \text{bit } i \text{ of } f(x) \text{ equals } b \}$$

belongs to DLOGTIME, except for the extra clause about M computing the length of $f(x)$, which is met in all instances that we know. Also noteworthy is that (d) is equivalent to a uniform notion of *projection reductions* as defined by Valiant [Val82] (see also [SV85]). A projection reduction is given by a family of mappings $\pi_n : \{1, \ldots, n'\} \to \{0, 1, x_1, \neg x_1, \ldots, x_n, \neg x_n\}$. Intuitively, $\pi_n(j)$ either sets bit j of $f(x)$ to 0 or 1 depending only on n, or chooses some input bit x_i that the output bit depends on, and whether the output is x_i or $\neg x_i$. The uniformity is that in case (d), the mapping π_n itself is computed in log-time as a function of n—since the input x is not examined for this, the differences in proviso do not matter here (also, n' may depend only on n).

Why study the latter three reducibility relations? Our main motivation is that the first, which is the standard notion of DLOGTIME reductions, does not preserve membership in DLOGTIME, nor is it transitive: Define $L_1 := \{\, x : x$ begins with $\log n$ 1s$\,\}$ and $L_2 := \{\, x : x$ begins with $(\log n)^2$ 1s$\,\}$. Then $L_1 \in$ DLOGTIME (in fact, $L_1 \in$ DLT), and $L_2 \leq_m^{\mathcal{U}} L_1$, but $L_2 \notin$ DLOGTIME, as can be seen by an easy adversary argument. However, the other reducibilities have nice properties: (For two reducibilities $\leq_1$ and $\leq_2$, $\leq_1 \subset \leq_2$ means that the set $\{\,(A,B) : A \leq_1 B\,\}$ is properly contained in $\{\,(A,B) : A \leq_2 B\,\}$.)

Lemma 2.2 *(a) The relations $\leq_m^{dlt}$, $\leq_m^{\mathcal{S}}$, and $\leq_m^{\mathcal{R}}$ are transitive, and the equivalence classes of the language $1\Sigma^*$ under them are, respectively,* DLT, $\Delta_0^{\mathcal{S}}$, *and* $\Delta_0^{\mathcal{R}}$.

(b) For all $k \geq 0$, the class Σ_k^{dlt} is closed downward under $\leq_m^{dlt}$, $\Sigma_k^{\mathcal{S}}$ is closed downward under $\leq_m^{\mathcal{S}}$, and $\Sigma_k^{\mathcal{R}}$ is closed downward under $\leq_m^{\mathcal{R}}$.

(c) $\leq_m^{\mathcal{R}} \subset \leq_m^{\mathcal{S}} \subset \leq_m^{\mathcal{U}}$, and $\leq_m^{dlt} \subset \leq_m^{\mathcal{U}}$, but both $\leq_m^{\mathcal{R}}$ and $\leq_m^{\mathcal{S}}$ are incomparable with $\leq_m^{dlt}$.

Proof. Parts (a) and (b) are straightforward, while the proper containments in (c) follow from $\Delta_0^{\mathcal{R}} \subset \Delta_0^{\mathcal{S}}$ and the example with L_1 and L_2 above. A projection reduction in which bits $1, 2, 3, 4, 5, \ldots$ of the output depend, respectively, on bits $1, 2, 4, 8, 16, \ldots$ of the input is not DLT-computable, while a DLT reduction that uses $\log n$ input bits is not a Sipser reduction; we leave the reader to build languages showing the incomparabilities based on these ideas. $\square$

Thus the strong requirement that a machine computing a DLT reduction be able to output any log-many consecutive bits in log time, which is what makes the reducibility transitive, also makes it in some sense even "lower" than projection reductions. This motivates us to define what seems to be the sharpest sensible notion of reducibility, for studying very low complexity classes.

Definition 2.2. Given languages A and B, write $A \leq_{proj}^{dlt} B$ if A reduces to B by a Ruzzo reduction that is also a DLT reduction.

Then the relation $\leq_{proj}^{dlt}$ is transitive and preserves membership in all levels of all four hierarchies defined above. Our main results will construct these sharp reductions. We end this section by noting that the so-called "Sipser functions" F_d^n remain complete under these reductions. Following [Sip83], these are defined for n of the form m^d (m, d integral) by

$$F_d^n(x_1 \cdots x_n) = 1 \iff (\exists i_1 < m)(\forall i_2 < m)\ldots(Q i_d < m) x_{i_1 \cdot i_2 \cdots i_d} = 1; \qquad (3)$$

For n not of the form m^d, one may stipulate that F_d^n is identically 0. Then one defines the function $F_d : \{\,0,1\,\}^* \to \{\,0,1\,\}$ by $F_d(x) = F^{|x|}(x_1, \ldots, x_n)$. We identify F_d with the language $\{\, x : F_d(x) = 1\,\}$. For instance, the language F_1 equals $0^*1(0+1)^*$, and belongs to $\Sigma_1^{\mathcal{R}}$ (hence also to NLOGTIME), but not to $\Pi_1^{\mathcal{U}}$ (hence not to DLOGTIME).

Lemma 2.3 (cf. [CC95, CCH95]) *For all $d \geq 1$,*

(a) *F_d belongs to $\Sigma_d^{\mathcal{R}}$ but not to $\Pi_d^{\mathcal{U}}$.*

(b) *F_d is complete for $\Sigma_d^{\mathcal{U}}$ under $\leq_m^{\mathcal{U}}$, complete for Σ_d^{dlt} under $\leq_m^{dlt}$, complete for $\Sigma_d^{\mathcal{S}}$ under $\leq_m^{\mathcal{S}}$, and complete for $\Sigma_d^{\mathcal{R}}$ under $\leq_{proj}^{dlt}$.*

(c) *For all languages A, if $A \leq_m^{\mathcal{U}} F_d$ then $A \in \Sigma_d^{\mathcal{U}}$; if $A \leq_m^{dlt} F_d$ then $A \in \Sigma_d^{dlt}$; if $A \leq_m^{\mathcal{S}} F_d$ then $A \in \Sigma_d^{\mathcal{S}}$; and if $A \leq_m^{\mathcal{R}} F_d$ then $A \in \Sigma_d^{\mathcal{R}}$.*

Proof. Part (a) follows from methods and results in [CCH95], improving Sipser's theorem that $F_d \in \Sigma_d^{\mathcal{R}} \setminus \Pi_d^{\mathcal{S}}$ [Sip83]. The main point in (b) is that the projection reduction given by Sipser [Sip83] is also a DLT reduction, basically because any $O(\log n)$ consecutive leaves in a balanced binary tree can be visited in $O(\log n)$ moves in the tree. Part (c) for $\leq_m^{\mathcal{U}}$ is Theorem 7.4 in [CC95], and the closures for the other reductions follow from Lemma 2.2(b). $\square$

Of all our refinements of DLOGTIME reductions, we draw special attention to DLT reductions. These capture in a natural way the idea of "local replacement" used in many reductions among NP-complete problems. For instance, a reduction from one graph problem to another may involve constant-size neighborhoods of certain vertices v. The vertex v is encoded by an $O(\log n)$-bit number, which is why the ability to read $O(\log n)$-many consecutive bits in a DLT-reduction is appropriate. A second advantage of DLT is that it avoids a quirk in the relationship of DLOGTIME to linear time on standard deterministic multitape Turing machines, which is denoted by DLIN. DLT is properly contained in DLIN, but it is unknown whether DLOGTIME is contained in DLIN at all! The proviso $\mathcal{U}$ allows a DLOGTIME-machine M to "jump around" a lot on the input tape by editing bits toward the middle of its index tape, and we do not see how to simulate this in better than $O(n \log n)$ time on a standard TM.

3 Strong time-constructibility and main lemmas

Definition 3.1. We say a function $g\colon \mathbb{N} \to \mathbb{N}$ is *strongly time constructible* if there is a standard (not indexing) Turing machine M such that for all $n \in \mathbb{N}^*$, M computes $g(n)$ within $O(|n|+|g(n)|)$ steps, and additionally there is an $\epsilon > 0$ such that $g(0) > 0$, $g(1) > 1$, and for all $n \geq 2$, $g(n) \geq \lceil n^{1+\epsilon} \rceil$. The input to g is given in binary notation.

For comparison, the notion of g being "[fully] time constructible" from [HU79] would require that for all n and all x of length n, $M(x)$ runs for [exactly] $g(n)$ steps. For functions g, g' on $\mathbb{N}$, say that g' *majorizes* g if for all n, $g'(n) \geq g(n)$. To keep this section self-contained, we give the following slightly modified form of Lemma 5.2 in [Vol90].

Lemma 3.1 *For every recursive function $g\colon \mathbb{N} \to \mathbb{N}$ there is a strongly time constructible g' that majorizes g.*

Proof. Define $g''(n) = \max\{g(n), n^2\}$. Then g'' is recursive, so there is a Turing machine M computing g''. Define M' to be a machine that simulates M and additionally, in every step of its computation prints the symbol '1' on its output tape. Then M' computes a function g' which certainly majorizes g'', and to compute g', M' needs exactly as many steps as there are symbols in the output. $\square$

Definition 3.2. A function f defined on strings is *strongly growing* if there is a strongly time constructible function $g\colon \mathbb{N} \to \mathbb{N}$ such that for all x, $f(x) = 1^{g(|x|)-1}$.

Lemma 3.2 *For every recursive function $f\colon \Sigma^* \to \Sigma^*$ there is a strongly growing function f' that majorizes f.*

Proof. Given $f\colon \Sigma^* \to \Sigma^*$, define $g\colon \mathbb{N} \to \mathbb{N}$ as $g(n) = \max_{|x|=n} |f(x)| + 1$. Then certainly $\hat{f}(x) =_{\text{def}} 1^{g(|x|)-1}$ majorizes f. Now find a strongly time-constructible function $g' \geq g$ as in Lemma 3.1, and define $f'(x) = 1^{g'(|x|)-1}$. $\square$

The following main lemma is the "infinite version" of Lemma 5.3 from [Vol90], in the sense of the diagonalization over infinitely many classes in [Reg92a].

Lemma 3.3 *For every strongly time-constructible function $g\colon \mathbb{N} \to \mathbb{N}$, there is a* DLT-*computable function $h\colon \Sigma^* \to \mathbb{N}$ with the following property: For all $\ell \in \mathbb{N}$ there are infinitely many $k \in \mathbb{N}$ such that for all y with $k \leq |y| < g(k)$, $h(y) = \ell$.*

Proof. Design a Turing machine M computing h as follows: On input x and $n = |x|$,

1. M computes 0, $g(0)$, $g(g(0))$, $\ldots$, until some k is found such that $g^{(k)}(0) \leq n < g^{(k+1)}(0)$.
2. M outputs the number of trailing zeroes in the binary representation of k.

To determine the running time of M of Step 1, observe that since $g^{(k)}(0) \leq n$ and g is strongly time-constructible, we know that the time needed to compute $g^{(k)}(0)$ is $O(\log n)$. Since $g(m) \geq \lceil m^{1+\varepsilon} \rceil$ we get $g^{(k-1)}(0) < n^{1/(1+\varepsilon)}$, thus the time to compute $g^{(k-1)}(0)$ is less than $c/(1+\varepsilon) \cdot \log n$ for some constant c. Repeating this argument, we see that the whole time needed to compute 0, $g(0)$, $g(g(0))$, $\ldots$, $g^{(k)}(0)$ is

$$\sum_{i=0}^{k} \frac{c}{(1+\varepsilon)^i} \cdot \log n = O(\log n).$$

Observe, that we don't have to compute the value $g^{(k+1)}(0)$, but we can simply stop that computation after $c' \log n + 1$ steps (for some constant c'), since we then know that $g^{(k+1)}(0) > n$. Keeping count of k requires $O(\log n)$ steps, since by a well known amortized cost analysis, to count from 0 to k can be done in time $O(k) = O(\log n)$. Thus the overall time needed for Step 1 is $O(\log n)$; and clearly this bound holds also for Step 2. Since for every number ℓ there are infinitely many k such that the number of trailing zeroes in the binary representation of k is equal to ℓ, the conclusion follows. $\square$

The preceding lemma can be stated in a form analogous to Theorem 4.1 from [Reg92a], which is more convenient for the theorems that follow. Here $\leq$ denotes the standard lexicographic order on strings.

Corollary 3.4 *For every strongly growing function f, we can construct a* DLT*-computable function $h\colon \Sigma^* \to \mathbb{N}$ with the following property: For all $\ell \in \mathbb{N}$ there are infinitely many $x \in \Sigma^*$ such that for all y with $x \le y \le f(x)$, $h(y) = \ell$.*

Proof. By hypothesis, there is a strongly time-constructible function $g\colon \mathbb{N} \to \mathbb{N}$ such that for all x, $f(x) = 1^{g(|x|)-1}$. Now apply the construction of the previous proof to g. $\square$

4 Applications

Before showing that several well-known structure theorems of the polynomial hierarchy carry over (unconditionally!) to the log-time hierarchy, we need to take care about how several basic operations on languages are coded for low-level classes. For one example, consider the "direct connection language" of a family $C = [C_n]$ of Boolean circuits, where the C_n have polynomial size $p(n)$ and gate labels in $\{1, \ldots, p(n)\}$, as defined and used in [BIS90]:

$$D(C) = \{ (g, h, t, y) : |y| = n, \text{gate } g \text{ gets input from gate } h \text{ in } C_n, \text{ and gate } g \text{ has type } t \}.$$

Let $D'(C)$ stand for the same definition with 0^m in place of y. The problem with $D'(C)$ is that a DLOGTIME machine cannot verify that the final m "padding bits" are all 0; this is why "y" in the definition of $D(C)$ may be an arbitrary string of length n. (Note also that $D(C) \in$ DLOGTIME $\iff$ $D(C) \in$ DLT, since the ability to ignore y leaves only the first $O(\log n)$ bits of the input needing to be read. Thus DLT-uniformity is the same as DLOGTIME-uniformity, another good point for DLT. Furthermore, let $D''(C)$ stand for the same definition with m in binary notation in place of y or 0^m. Then $D(C) \in$ DLOGTIME $\iff$ $D''(C) \in$ DLIN. These same remarks apply if the definitions are strengthened by changing the tuple to $(g, h, i, t, \cdot)$ and requiring also that g be the ith input to h under the label ordering, as done in [BI94] to come into line with the original definitions in [Ruz81].)

A second caveat about encodings concerns the treatment of tuples such as (g, h, t, y) here. If the first $k-1$ elements of a k-tuple $(x_1, \ldots, x_k)$ have length $O(\log |x_k|)$, then we may encode the tuple as $d(x_1)01d(x_2)01 \ldots d(x_{k-1})01x_k$, where $d(z)$ doubles each bit of an argument string z. For general tuples, however, one must beware the danger that the range of the encoding may not be recognizable by a deterministic log-time TM. This problem does not arise in the uses that follow.

A third example concerns the *join* of two languages A and B, which is standardly defined by $A \oplus B = \{ 0x : x \in A \} \cup \{ 1x : x \in B \}$. For nearly every reducibility relation $\le_r$ that has been studied, down to $\le_m^{dlt}$ and $\le_m^{\mathcal{S}}$, this join gives the least-upper-bound property

$$A \le_r C \ \wedge\ B \le_r C \Longrightarrow A \oplus B \le_r C. \tag{4}$$

This fails, however, for $\le_{proj}^{dlt}$ and other projection reductions, because of the need to examine the extra tag bit of a string in $A \oplus B$. We adjust by defining:

$$A \uplus B = \{ xz : x \in A, |z| = |x| \} \cup \{ xz : x \in B, |z| = |x| + 1 \}. \tag{5}$$

Then (4) holds for $A \uplus B$, because the log-time machine can read the length m of its input for odd-or-even and select the appropriate map involving the first $\lfloor m/2 \rfloor$ bits accordingly. Also, for any language A, $A \equiv^{dlt}_{proj} A \uplus \emptyset \equiv^{dlt}_{proj} A \uplus A$.

We similarly wish to encode an infinite sequence $A_1, A_2, A_3, \ldots$ of languages into one. For this we need a pairing function $\langle \cdot, \cdot \rangle$ that is computable and invertible in TM *linear* time in binary notation. We fix the example $\langle x, y \rangle =_{\rm def} x{\cdot}y{\cdot}str(2|x| + |y| - 2)$ from [Reg92b]. We can ignore the fact that the three values $\langle \lambda, \lambda \rangle$, $\langle \lambda, 0 \rangle$, and $\langle \lambda, 1 \rangle$ are undefined; on the rest of $\Sigma^* \times \Sigma^*$, $\langle \cdot, \cdot \rangle$ is bijectively onto Σ^*. Then we define:

$$A_\omega = \{\, xz : x \in A_k, \text{ where } |z| = \langle |x|, k \rangle - |x| \,\}. \tag{6}$$

Then given the length m of a string w to be examined for membership in A_ω, a TM can compute both n and k such that $\langle n, k \rangle = m$ in $O(\log m)$ time.

The above pairing function is also computable in log time by a two-input-tape variant of the model developed in Section 2: Let M have two index tapes, one accessing an input tape holding x and the other an input tape holding y, and let $|x|$ and $|y|$ be initially given in binary on two worktapes of M. Then M can compute $\langle x, y \rangle$ in log time under both the 1-bit $\mathcal{R}$ proviso and the proviso for DLT projections, since the "$2|x| + |y|$" portion has length logarithmic in $|x| + |y|$.

Hence we may use this pairing function freely in our final spate of definitions: A language U is called a *universal language* for a class $\mathcal{C}$ if $\mathcal{C}$ is exactly the class of all sets $U_k = \{\, x : \langle x, k \rangle \in U \,\}$ for $k \in \mathbb{N}$. Observe that by $x \mapsto \langle x, k \rangle$, we get $U_k \leq^{dlt}_m U$. We say that $\mathcal{C}$ is *recursively presentable* (r.p.) if there is a recursive universal language U for $\mathcal{C}$. In that case, we call $[U_k]_{k=1}^\infty$ a *recursive presentation* of $\mathcal{C}$. (Note that U_ω is then the same as U.) A sequence $\mathcal{C}_1, \mathcal{C}_2, \ldots$ of *classes* is recursively presentable if there is a recursive language U such that for each k, U_k is universal for $\mathcal{C}_k$. Finally, a class $\mathcal{C}$ is *closed under finite variations* (cfv.) if for every $A \in \mathcal{C}$ and every B such that $(A \cup B) \setminus (A \cap B)$ is finite, we have $B \in \mathcal{C}$.

Lemma 4.1 *All of the classes defined in Section 2, together with* NC^1 *(which stands for* DLOGTIME-*uniform* NC^1*) are recursively presentable and cfv. Let $\leq_r$ be any of the reducibilities defined in Section 2, let $\mathcal{C}$ be any r.p. cfv. class, and let A be any recursive language. Then the classes*

$$\{\, L : L \leq_r A \,\} \qquad \textit{and} \qquad \{\, L \in \mathcal{C} : A \leq_r L \,\}$$

are r.p. and cfv., unless the latter is empty. The union and intersection of two r.p. cfv. classes is also r.p. cfv., unless the latter is empty.

Proof. All of these facts are standard (see e.g. [Sch82]); we give details for the case of $\{\, L \in \mathcal{C} : A \leq_r L \,\}$. Let $[T_j]$ be a recursive presentation of machines that compute the reducibility $\leq_r$ in question, let $[U_k]$ be a recursive presentation of $\mathcal{C}$, and fix some $L_0 \in \mathcal{C}$ such that $L_0 \leq_r A$. For all j and k, let M_{jk} be a machine that operates as follows on any input x: M_{jk} first spends $n = |x|$ steps checking whether the condition "$y \in U_k \iff T_j(y) \in A$" holds for $y = \lambda, 0, 1, \ldots$. If the condition is found to fail for some y within those n steps, then M_{jk} accepts x iff $x \in L_0$. If it holds for those n steps, then

M_{jk} accepts x iff $x \in U_k$. Thus for all j and k, either $U_k \leq_r A$ via T_j and $L(M_{jk}) = U_k$, or T_j does not reduce U_k to A, in which case $L(M_{jk})$ is a finite variation of L_0. Thus $L(M_{jk}) \leq_r A$, since all of the reducibilities $\leq_r$ in question allow "patching" on strings of finitely many lengths. Since $\mathcal{C}$ is cfv., also $L(M_{jk}) \in \mathcal{C}$. Thus the recursive presentation $[L(M_{jk})]$ captures exactly those languages in $\mathcal{C}$ that reduce to A under $\leq_r$. □

We first state a log-time version of the "uniform diagonalization theorem" of Schöning [Sch82] (see also [BDG88, Reg88]), and then state and prove an extension to infinitely many classes along the lines of Theorem 5.2(a) in [Reg92a]. The main point is that now the reduction is a DLT projection reduction.

Theorem 4.2 *Let $\mathcal{C}_1, \mathcal{C}_2$ be recursively presentable cfv. classes, and let A_1, A_2 be recursive languages such that $A_1 \notin \mathcal{C}_1$ and $A_2 \notin C_2$. Then we can construct a recursive language E such that $E \notin \mathcal{C}_1 \cup \mathcal{C}_2$, and yet $E \leq^{dlt}_{proj} A_1 \uplus A_2$.*

Theorem 4.3 *Let $[A_k]_{k=1}^{\infty}$ and $[\mathcal{C}_k]_{k=1}^{\infty}$ be recursive presentations of languages and classes, respectively, such that for all k, $\mathcal{C}_k$ is cfv. and $A_k \notin \mathcal{C}_k$. Then we can construct E such that $E \notin \bigcup_{k=1}^{\infty} \mathcal{C}_k$ and $E \leq^{dlt}_{proj} A_\omega$.*

Proof. Let U be the recursive language such that for all k, $\mathcal{C}_k = \{ U_{k\ell} : k, \ell \in \mathbb{N} \}$, where $U_{k\ell}$ technically stands for $(U_k)_\ell$. Define the function

$$f(x) = \max_{k,\ell \leq x} \min\{ y : y \in A_k \bigtriangleup U_{k\ell} \}.$$

Take the function g from Lemma 3.2 that is strongly-growing and majorizes f, and then take h from Corollary 3.4. Define

$$E = \bigcup_{k=1}^{\infty} h^{-1}(k) \cap A_k.$$

Then E reduces to A_ω by the map $x \mapsto \langle x, h(x) \rangle$, which is a DLT projection reduction. That $E \notin \cup_k \mathcal{C}_k$ follows by the same analysis as in Theorem 5.2(a) of [Reg92a]. The proof of Theorem 4.2 is similar. □

Because the log-time hierarchy is proper, as in equations (2) and (1), we obtain analogues of results about the polynomial hierarchy in [Lad75, Sch82], but without any unproven hypotheses about non-collapse of the latter. Part (a) is an unconditional "Ladner's theorem."

Corollary 4.4 *(a) There exist languages $E \in$ NLOGTIME $\setminus$ DLOGTIME that are not complete for NLOGTIME under DLT reductions.*

(b) There exist languages $E \in \mathrm{NC}^1 \setminus$ LOGH that are not hard for any level of the log-time hierarchy above DLOGTIME.

Proof. Recall the Sipser languages F_d defined in (3) following Definition 2.2.

(a) In Theorem 4.2, take $\mathcal{C}_1 =$ DLOGTIME, $A_1 = F_1$, $\mathcal{C}_2 = \{ L \in \text{NLOGTIME} : L \leq^{\mathcal{U}}_m F_1 \}$, $A_2 = \emptyset$. These choices satisfy the hypotheses. The conclusion in fact gives an

$E \in \Sigma_1^{\mathcal{R}}$ that is neither in DLOGTIME nor complete under DLOGTIME reductions, with $E \leq_{proj}^{dlt} F_1$.

(b) First we note that the language F_ω defined from the sequence of Sipser languages belongs to NC^1. This is because strings in F_d have length m^d for some integral m, and so the unbounded fan-in expression defining F_d for a given m translates into a Boolean formula (fan-in 2) of size $O(m^d)$. Now take

$$\begin{array}{lllllll} \mathcal{C}_1 & := & \mathrm{LOGH}, & \qquad & A_1 & := & F_\omega, \\ \mathcal{C}_2 & := & \{\, L \in \mathrm{NC}^1 : F_1 \leq_m^{\mathcal{U}} L \,\}, & \qquad & A_2 & := & \emptyset, \end{array}$$

and the conclusion follows. □

In (a), suppose we now re-define $\mathcal{C}_1$ to be the class of languages $L \in$ NLOGTIME such that $E \leq_m^{\mathcal{U}} L$, and re-define A_2 to be E. Then one obtains E' such that $E' \leq_{proj}^{dlt} E$ but $E \not\leq_{proj}^{dlt} E'$, indeed $E \not\leq_m^{\mathcal{U}} E'$. In like manner it follows that for each of the reducibilities $\leq_r$ in Section 2, the *degrees* of $\leq_r$ (i.e., the equivalence classes under $A \equiv_r B =_{\mathrm{def}} A \leq_r B \;\wedge\; B \leq_r A$) are dense. Given $A <_{proj}^{dlt} E <_{proj}^{dlt} B$, we can create a "diamond" by constructing D such that $A <_{proj}^{dlt} D <_{proj}^{dlt} B$, but D is incomparable with E (even under $\leq_m^{\mathcal{U}}$ reductions if $B \not\leq_m^{\mathcal{U}} E \not\leq_m^{\mathcal{U}} A$). This is done by taking $\mathcal{C}_1 = \{\, L : A \uplus L \leq_m^{\mathcal{U}} E \,\}$, $A_1 = B$, $\mathcal{C}_2 = \{\, L \in \mathrm{NLOGTIME} : E \leq_m^{\mathcal{U}} A \uplus L \,\}$, and $A_2 = A$, getting E'' from Theorem 4.2, and defining $D = A \uplus E''$. Here is where the least-upper-bound property (4) begins to be used. Even more strongly:

Corollary 4.5 *One can embed every countable partial order P into the structure of languages $E \in$* NLOGTIME $\setminus$ DLOGTIME *so that: every related pair in P maps to two languages related by a* DLT *projection reduction, while every unrelated pair maps to two languages that are not related even by a* DLOGTIME *reduction.*

Proof. One can follow Mehlhorn's published proof [Meh76] of the corresponding result for polynomial-time degrees, or that in [Reg83] for embedding posets into p-isomorphism types of NP-complete sets. See also [MY85, Amb85b, Reg86, Amb88]; the thesis [Reg86] has full details for embedding upper semi-lattices under one fine reducibility $\leq_r$ and preserving incomparabilities under a coarser reducibility $\leq_{r'}$. □

Cai, Chen, and Håstad [CCH95] define refinements $\Sigma_k^{1\,bit} = \Sigma_m^{\mathcal{R}}, \Sigma_k^{2\,bits}, \Sigma_k^{3\,bits}, \ldots$ of $\Sigma_k^{\mathcal{S}}$, and show that for all $k \geq 0$, the containments in these classes are proper.

Corollary 4.6 *There exist languages $A, E \in \Delta_0^{2\,bits} \setminus \Delta_0^{\mathcal{R}}$ such that there is no uniform projection reduction from A to E.*

That is, any machine computing a reduction from A to E must examine at least two bits of its input x—which is the same as the power to solve $x \in A$ without bothering with E at all. Put another way, E gives no "help" to solving A. We can obtain similar results for classes $\Sigma_k^{m\,bits}$ versus $\Sigma_k^{\ell\,bits}$, with $k \geq 0$ and $m > \ell > 0$, whose formulation and proof we leave to the reader. (One can also embed all countable partial orders under $\leq_{proj}^{dlt}$ into the difference of the two classes, etc.)

Another notion of "computational help," motivated by the well-known *Switching Lemma* of [Hås89] for the Sipser languages F_d (variously defined), is the following: Say a language E "helps F_d to switch" if $F_d \leq_m^{dlt} \widetilde{F_d} \uplus E$ (where for a set A, $\widetilde{A}$ denotes the complement of A). Our next two results use the infinite case, Theorem 4.3.

Theorem 4.7 *There exists a language $E \in \mathrm{NC}^1 \setminus \mathrm{LOGH}$ that does not help any F_d to switch.*

Proof. For each $d \geq 1$, define

$$\begin{aligned} \mathcal{C}_{2d-1} &:= \{\, L \in \mathrm{NC}^1 : F_d \leq_m^{dlt} \widetilde{F_d} \uplus L \,\}, & A_{2d-1} &:= \widetilde{F_d}, \\ \mathcal{C}_{2d} &:= \{\, L \in \mathrm{NC}^1 : \widetilde{F_d} \leq_m^{dlt} F_d \uplus L \,\}, & A_{2d} &:= F_d. \end{aligned}$$

All of these classes are cfv. and are recursively presentable, via Lemma 4.1. For each k, $A_k \notin \mathcal{C}_k$. Hence the hypotheses of Theorem 4.3 are satisfied. We get $E \leq_m^{dlt} F_\omega$, so $E \in \mathrm{NC}^1$, but since $E \notin \mathcal{C}_k$ for all k, E does not help any F_d or $\widetilde{F_d}$ to switch. $\square$

Theorem 4.8 *In any recursive presentation $B_1, B_2, B_3, \ldots$ of NC^1, there must be some d such that B_d is a finite variation of the Sipser language F_d itself.*

Note that this is *not* the case with recursive presentations of LOGH—one can avoid every F_d at the dth step by interleaving presentations of $\Sigma_1^{\mathcal{U}}$, $\Sigma_2^{\mathcal{U}}$, $\Sigma_3^{\mathcal{U}}, \ldots$ slowly enough. Hence the inability to avoid this "F_d fixed point" in a recursive presentation of NC^1 is surprising. The governing factor is that the language F_ω belongs to NC^1. See section 7 of [Reg92a] for related results in the polynomial hierarchy and further discussion of fixed-point theorems.

Proof. Suppose not; then with $A_d := F_d$, $\mathcal{C}_d := \{\, L : L \text{ is a finite variation of } B_d \,\}$, the hypotheses of Theorem 4.3 are satisfied. This gives a language E such that $E \leq_m^{\mathcal{U}} F_\omega$, so $E \in \mathrm{NC}^1$, but also $E \notin \cup_d \mathcal{C}_d = \mathrm{NC}^1$, a contradiction. $\square$

Now let $\mathcal{F}$ be a sound, recursively axiomatized system of logic that can formalize assertions about Turing machines, such as PA or ZF as mentioned in Section 1. Here "recursively axiomatized" (r.a.) implies that the predicate $P_{\mathcal{F}}(d,t) \equiv$ 'd is a proof of theorem t in $\mathcal{F}$' is decidable, and "sound" means that every theorem t about Turing machines proved by $\mathcal{F}$ is true. To formalize nontrivial properties Π of those languages in NC^1, we refer to a fixed enumeration $[Q_i]_{i=1}^{\infty}$ of NC^1-machines. Then the class of languages proved by $\mathcal{F}$ to have property Π is recursively presentable, since presented by:

$$U_{i,d} = L(Q_i) \text{ if } P_{\mathcal{F}}(d, \text{`}L(Q_i) \text{ has } \Pi\text{'}), \text{ otherwise } V,$$

where V is some fixed language that has property Π.

Theorem 4.9 *For all $k \geq 1$, there are languages $E \in \Sigma_k^{dlt} \setminus \Sigma_{k-1}^{\mathcal{U}}$ such that $\mathcal{F}$ cannot prove $E \notin \mathrm{DLT}$.*

Proof. Take $A_1 := \emptyset$ and $\mathcal{C}_1$ to be the class of languages in NC^1 that $\mathcal{F}$ can prove to be infinite, which contains the class of languages that $\mathcal{F}$ can prove to lie outside DLT, or outside $\Sigma_{k-1}^{\mathcal{U}}$ for that matter. Then $\mathcal{C}_1$ is recursively presentable, per above remarks. Since $\mathcal{F}$ is sound, $A_1 \notin \mathcal{C}_1$. Also take $A_2 := F_k$ and $\mathcal{C}_2 := \Sigma_{k-1}^{\mathcal{U}}$. The resulting language E has the desired properties. $\square$

Say that a property Π is "fv-nontrivial" if there is some language A such that all finite variations of A have property Π, and some A' such that all finite variations of A' do not have Π.

Theorem 4.10 *Let $\mathcal{C}$ be any class of languages that is closed under $\uplus$ and under $\leq_{proj}^{dlt}$. Then every fv-nontrivial property Π of languages in $\mathcal{C}$ is undecidable. Indeed, for every sound, r.a. system $\mathcal{F}$, there are languages $E \in \mathcal{C}$ such that $\mathcal{F}$ does not prove the true statement "E has Π" or "E lacks Π" (whichever applies to E).*

The proof is similar to the above. (For a treatment with attention to the most general details of how statements such as "E has Π" are formalized, see [Reg88].)

5 Concluding Discussion

Schmidt's definition of a class $\mathcal{C}$ being "recursive gap closed" is essentially the same as saying that for every $A_1, A_2 \in \mathcal{C}$ and recursive function f, there is a strongly-growing function g majorizing f such that if one defines the "gap language" G to be the set of x such that k in Lemma 3.3 is odd, then $(A_1 \cap G) \cup (A_2 \cap \widetilde{G})$ also belongs to $\mathcal{C}$. Put another way, $\mathcal{C}$ is closed under the operation of forming the language E in Theorem 4.2. We have identified DLOGTIME, DLT, and even $\Delta_0^{\mathcal{R}}$ as being very small classes that are recursive gap closed, and the same extends to the higher levels of the log-time hierarchy.

It is possible to go even lower. For any language A, define its associated "fat tally set" by

$$F(A) := \{\, y \in \Sigma^* : str(|y|) \in A \,\}.$$

Now define $\Delta_0^{0\,bits}$ to be the class of languages $F(A)$ for A in TM linear time. These are precisely the languages accepted by deterministic log-time TMs that look at *no* bits of their input, but decide everything based on the given length n of the input. This is a proper subclass of $\Delta_0^{\mathcal{R}}$. Since the function h in Theorem 4.2 does not depend on any bit of the input, it falls out that $\Delta_0^{0\,bits}$ is recursive gap closed.

With appropriate artifice, one can define "loglog-time Turing machines" (cf. the loglog-space TMs in Buss [Bus93]), so that the class of languages $F(F(A))$ is in loglog time. Then we claim that the mechanism of Lemmas 3.1 through 3.3 can be tweaked to run in loglog time on these machines, so that this class is recursive gap closed. With even more artifice, this can be taken down to a notion of "logloglog time," and so on.

From all this we can draw an interesting general conclusion: Complexity classes defined by bounds on running time (or space, or various other complexity measures) really are qualitatively different from classes in formal language theory. Properties such as "finiteness" tend to be decidable in moderate-size formal-language classes, for instance the context-free languages, whereas they are undecidable in any class that is recursive gap closed. We have shown that the "recursion-theoretic structure" that makes all similar properties undecidable goes all the way down in complexity theory. We look forward to further research on the *combinatorial* structure of the very low classes discussed in this paper, especially with the outward-looking motivations and ideas expressed in Section 2 and the beginning of Section 4.

Acknowledgments We thank Jianer Chen for sending us draft copies of his papers with Liming Cai and Johan Håstad, which provided a backbone for our work on the log-time hierarchy. Eric Allender and David Barrington also gave us some helpful comments.

References

[ACS87] A. Aggarwal, A. Chandra, and M. Snir. Hierarchical memory with block transfer. In *Proc. 28th Annual IEEE Symposium on Foundations of Computer Science*, pages 204–216, 1987.

[Amb85a] K. Ambos-Spies. Sublattices of the polynomial time degrees. *Inform. and Control*, 65:63–84, 1985.

[Amb85b] K. Ambos-Spies. Three theorems on polynomial degrees of NP sets. In *Proc. 26th Annual IEEE Symposium on Foundations of Computer Science*, pages 51–55, 1985.

[Amb88] K. Ambos-Spies. Polynomial time degrees of NP-sets. In E. Börger, editor, *Trends in Theoretical Computer Science*, pages 95–142. Computer Science Press, 1988.

[BDG88] J. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I.* Springer Verlag, 1988.

[BI94] D. Mix Barrington and N. Immerman. Time, hardware, and uniformity. In *Proc. 9th Annual IEEE Conference on Structure in Complexity Theory*, pages 176–185, 1994. Final version to appear in L. Hemaspaandra and A. Selman, eds., *Complexity Theory Retrospective II*, Springer Verlag, 1996.

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

[BS90] R. Boppana and M. Sipser. The complexity of finite functions. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, pages 757–804. Elsevier and MIT Press, 1990.

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

[Bus93] S. Buss. Algorithms for Boolean formula evaluation and for tree contraction. In P. Clote and J. Krajíček, editors, *Arithmetic, Proof Theory, and Computational Complexity*, pages 96–115. Oxford University Press, 1993.

[CC95] L. Cai and J. Chen. On input read-modes of alternating Turing machines. *Theor. Comp. Sci.*, 148:33–55, 1995.

[CCDF94] L. Cai, J. Chen, R. Downey, and M. Fellows. On the structure of parametrized problems in NP. In *Proc. 11th Annual Symposium on Theoretical Aspects of Computer Science*, volume 775 of *Lect. Notes in Comp. Sci.*, pages 509–520. Springer Verlag, 1994.

[CCDF95] L. Cai, J. Chen, R. Downey, and M. Fellows. On the parameterized complexity of problems in NP. *Inform. and Comp.*, 123:38–49, 1995.

[CCH95] L. Cai, J. Chen, and J. Håstad. Circuit bottom fan-in and computational power, 1995. Accepted to SIAM J. Comput.

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

[CM81] P. Chew and M. Machtey. A note on structure and looking-back applied to the relative complexity of computable functions. *J. Comp. Sys. Sci.*, 22:53–59, 1981.

[Dow86] M. Dowd. Notes on log space representation, 1986. Typewritten manuscript.

[Har85] J. Hartmanis. Independence results about context-free languages and lower bounds. *Inf. Proc. Lett.*, 20:241–248, 1985.

[Hås89] J. Håstad. Almost optimal lower bounds for small-depth circuits. In S. Micali, editor, *Randomness and Computation*, volume 5 of *Advances in Computing Research*, pages 143–170. JAI Press, Greenwich, CT, USA, 1989.

[HU79] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison–Wesley, Reading, MA, 1979.

[Imm87] N. Immerman. Languages which capture complexity classes. *SIAM J. Comput.*, 16:760–778, 1987.

[JMT94] B. Jenner, P. McKenzie, and D. Therién. Logspace and logtime leaf languages. In *Proc. 9th Annual IEEE Conference on Structure in Complexity Theory*, pages 242–254, 1994.

[KOR87] S. Kurtz, M. O'Donnell, and J. Royer. How to prove representation-independent independence results. *Inf. Proc. Lett.*, 24:5–10, 1987.

[Lad75] R. Ladner. On the structure of polynomial-time reducibility. *J. Assn. Comp. Mach.*, 22:155–171, 1975.

[Meh76] K. Mehlhorn. Polynomial and abstract subrecursive classes. *J. Comp. Sys. Sci.*, 12:147–178, 1976.

[MY85] S. Mahaney and P. Young. Reductions among polynomial isomorphism types. *Theor. Comp. Sci.*, 39:207–224, 1985.

[Reg83] K. Regan. On diagonalization methods and the structure of language classes. In *Proceedings, International Conference on Foundations of Computation Theory (FCT'83), Borgholm, Sweden, August 1983*, volume 158 of *Lect. Notes in Comp. Sci.*, pages 368–380. Springer Verlag, 1983.

[Reg86] K. Regan. On the Separation of Complexity Classes, 1986. Dissertation, Oxford University.

[Reg88] K. Regan. The topology of provability in complexity theory. *J. Comp. Sys. Sci.*, 36:384–432, 1988.

[Reg92a] K. Regan. Diagonalization, uniformity, and fixed-point theorems. *Inform. and Comp.*, 98:1–40, 1992.

[Reg92b] K. Regan. Minimum-complexity pairing functions. *J. Comp. Sys. Sci.*, 45:285–295, 1992.

[Ruz81] W. Ruzzo. On uniform circuit complexity. *J. Comp. Sys. Sci.*, 22:365–383, 1981.

[Sch82] U. Schöning. A uniform approach to obtain diagonal sets in complexity classes. *Theor. Comp. Sci.*, 18:95–103, 1982.

[Sch85] D. Schmidt. The recursion-theoretic structure of complexity classes. *Theor. Comp. Sci.*, 38:143–156, 1985.

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

[SV85] S. Skyum and L. Valiant. A complexity theory based on Boolean algebra. *J. Assn. Comp. Mach.*, 32:484–502, 1985.

[Tor88] J. Torán. An oracle characterization of the counting hierarchy. In *Proc. 3rd Annual IEEE Conference on Structure in Complexity Theory*, pages 213–223, 1988.

[Val82] L. Valiant. Reducibility by algebraic projections. *L'Enseignement mathématique*, 28:253–268, 1982.

[Vol90] H. Vollmer. The gap-language technique revisited. In *Proc. 3rd Workshop on Computer Science Logic*, pages 389–399, 1990.