Logspace and Logtime Leaf Languages*

Birgit Jenner† Universität Tübingen

Pierre McKenzie‡ Université de Montréal

Denis Thérien§ McGill University

April 12, 1996

*A preliminary version of this paper appeared in *Proc. of the 9th IEEE Structure in Complexity Theory Conference* (1994), 242-254.

†Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Sand 13, D–72076 Tübingen, Germany. Work done while at Dept. Llenguatges i Sistemes Informàtics, Universitat Politècnica Catalunya, supported by a Habilitationsstipendium of Deutsche Forschungsgemeinschaft (DFG-Je 154 2/1–2). E-mail: `jenner@informatik.uni-tuebingen.de`

‡Dép. d'informatique et recherche opérationnelle, Université de Montréal, C.P. 6128, Succursale Centre-ville, Montréal (Québec), H3C 3J7 Canada. Work supported by NSERC of Canada and by FCAR du Québec. E-mail: `mckenzie@iro.umontreal.ca`

§School of Computer Science, McGill University, 3480 University Street, Montréal (Québec), H3A 2A7 Canada. Work done while on sabbatical at the Dept. Llenguatges i Sistemes Informàtics, Universitat Politècnica Catalunya, Barcelona, Spain. Work supported by NSERC of Canada, by FCAR du Québec, and by the Spanish Ministry of Education. E-mail: `denis@cs.mcgill.ca`

Abstract

The computation tree of a nondeterministic machine M with input x gives rise to a *leaf string* formed by concatenating the outcomes of all the computations in the tree in lexicographical order. We may characterize problems by considering, for a particular "leaf language" Y, the set of all x for which the leaf string of M is contained in Y. In this way, in the context of polynomial time computation, leaf languages were shown to capture many complexity classes. In this paper, we study the expressibility of the leaf language mechanism in the contexts of logarithmic space and of logarithmic time computation. We show that logspace leaf languages yield a much finer classification scheme for complexity classes than polynomial time leaf languages, capturing also many classes within P. In contrast, logtime leaf languages basically behave like logtime reducibilities. Both cases are more subtle to handle than the polynomial time case. We also raise the issue of balanced versus non-balanced computation trees underlying the leaf language. We indicate that it is a non-trivial problem to obtain information about the leaf string of a non-balanced computation tree and present conditions under which it doesn't matter whether the computation tree is balanced or not.

1 Introduction

Fix a finite alphabet Σ and let a nondeterministic Turing machine M output a symbol from Σ when M reaches a halting configuration. With the nondeterministic choices of M ordered, and under the assumption that M never loops, let $\mathit{leafstring}^M(x)$ be the concatenation of the symbols output at the leaves of M's computation tree.

In 1991, Bovet, Crescenzi and Silvestri (1991, 1992) introduced leaf strings to define language classes (see also Vereshchagin (1994)). Given a "leaf language" $Y \subseteq \Sigma^*$, they (essentially) defined the class $\mathsf{Leaf}^P(Y)$ of all languages $A \subseteq \{0,1\}^*$ such that, for some nondeterministic polynomial time Turing machine M, $x \in A$ iff $\mathit{leafstring}^M(x) \in Y$. One of their motivations was to exhibit machine-independent conditions for separations of relativized complexity classes. Aside from showing that $\mathsf{Leaf}^P(Y)$ ranges over a wealth of natural complexity classes as Y ranges over very natural languages, they were able to formulate necessary and sufficient conditions on the leaf languages (for example, non-interreducibility by polylogtime reductions) to get oracle separations; this generalized previous work by, for example, Angluin (1980), Torán (1988), Baker, Gill, and Solovay (1975), Baker and Gill (1979), that relates separation results in low-level complexity classes and oracle separations for the exponentially higher classes.

Hertrampf *et al.* (1993) studied $\mathsf{Leaf}^P(Y)$ as a function of the complexity of Y. They proved for instance that $PSPACE$, $\oplus P$ and the levels of PH are attainable using appropriate regular languages. They also considered context-free languages Y and languages Y drawn from various time-bounded or space-bounded complexity classes.

Borchert (1994) later proved that, for any (nontrivial) regular language Y, either $\mathsf{Leaf}^P(Y) = P$, or $\mathsf{Leaf}^P(Y)$ contains one of the classes NP, co-NP, or MOD_pP for prime p. In a similar spirit, Hertrampf (1994) developed, for general languages Y and Z, a general graph-theoretic criterion for $\mathsf{Leaf}^P(Y) \subseteq \mathsf{Leaf}^P(Z)$ to hold in all relativized worlds.

In this paper we consider the classes $\mathsf{Leaf}^L(Y)$ and $\mathsf{Leaf}^{LogT}(Y)$ defined by leaf languages of nondeterministic *logspace*-bounded and *logtime*-bounded Turing machines, respectively. Our motivation here is to obtain a broader characterization of complexity classes via leaf languages than leaf languages in the polynomial setting allow, namely one that also extends below P. In such a characterization, low-level complexity classes (or single problems) within NC^1 would be related with complexity classes, for example, within NC^2, and equivalences or separations in the low-level setting might provide valuable information about equivalences or (relativized) separations in the corresponding higher classes.

One's intuition might be that a routine adaptation of known proofs will characterize the new classes. However, new subtleties arise when Y is of low complexity. In particular, the intuition that the behavior of a complete language for a class C in a sense captures the behavior of all languages in C fails. For instance, let OR be the regular language $\{0,1\}^*1\{0,1\}^*$ and let $\leq_{dlogtime}(OR)$ denote the set of languages reducible to OR in $DLOGTIME$ (a reducibility defined precisely in the next section, following Buss (1987)).We prove that

$$\mathsf{Leaf}^L(OR) = NL$$

and yet

$$\mathsf{Leaf}^L(\leq_{dlogtime}(OR)) = NP,$$

where $\mathsf{Leaf}^L(\text{class } C) = \cup_{Y \in C}\mathsf{Leaf}^L(Y)$. As another example, let REG denote the regular languages, known to contain NC^1-complete languages under $\leq_{dlogtime}$ (Barrington (1989)).We prove that

$$\mathsf{Leaf}^L(REG) = P$$

and yet

$$\mathsf{Leaf}^L(NC^1) = PSPACE.$$

In some cases furthermore we obtain characterizations which on the surface are similar to those previously known, yet which require completely different proofs. An example of this situation is Theorem 3.10 showing that $PSPACE = \mathsf{Leaf}^L(DCFL)$, a striking result nonetheless similar at first glance to the known characterization of $PSPACE$ as $\mathsf{Leaf}^P(DCFL)$ by Hertrampf *et al.* (1993).

Turning to nondeterministic *logtime* Turing machines, we observe that using a leaf language Y to recognize a language X basically amounts to $X \leq_{dlogtime} Y$. More precisely, for any class C that is closed under padding, i.e., under the operation that turns a word $w \in \Sigma^*$ into a word $w' \in (\Sigma \cup \$)^*$ that is identical to w when all \$-symbols are deleted, we show that Leaf^{LogT}(C) is the $\leq_{dlogtime}$ closure of C. We immediately derive various characterizations of $\mathsf{Leaf}^{LogT}(C)$ for classes C closed under $\leq_{dlogtime}$, e.g., $\mathsf{Leaf}^{LogT}(NC^1) = NC^1$. Moreover, we obtain characterization results for classes that are expressible as the $\leq_{dlogtime}$ closure of a class C closed under padding, e.g.

$$\mathsf{Leaf}^{LogT}(DDH) = \leq_{dlogtime}(DDH) = AC^0,$$

Class C	$\mathsf{Leaf}^P(C)$ (Hertrampf *et al.* (1993))	$\mathsf{Leaf}^L(C)$ (this paper)	$\mathsf{Leaf}^{LogT}(C)$ (this paper)
CSL	$EXPSPACE$	$EXPSPACE$	$PSPACE$
NP	$NEXPTIME$	$NEXPTIME$	NP
P	$EXPTIME$	$EXPTIME$	P
$POLYLOGSPACE$	$PSPACE$	$PSPACE$	$POLYLOGSPACE$
NC^1	$PSPACE$	$PSPACE$	NC^1
CFL	$PSPACE$	$PSPACE$	$LOGCFL$
$DCFL$	$PSPACE$	$PSPACE$	$LOGDCFL$
AC^0		PH	
Σ_k^{LogT}, any $k \geq 0$		Σ_k^p	
$SAC^{(\log\log n)}$		NP	
$POLYLOGTIME$		P	
REG	$PSPACE$	P	NC^1
$SOLVABLE$	Mod-PH	$\subseteq$ Mod-$LogSH$	ACC^0
$PERIODIC\ SOLVABLE$	Mod^*P	$\subseteq \mathrm{Mod}^*L$	$\subseteq CC^0$
$PERIODIC\ p\text{-}GROUP$		MOD_pL	
$DOT\ DEPTH\ HIERARCHY$	PH	NL	AC^0
$DOT\ DEPTH\ k$, any $k \geq 1$	$Boolean\text{-}closure(\Sigma_k^p)$	NL	

Figure 1: Our main characterizations of $\mathsf{Leaf}^L(\cdot)$ and $\mathsf{Leaf}^{LogT}(\cdot)$ classes.

where DDH is the $DOT\ DEPTH\ HIERARCHY$.

Figure 1 lists our main results, where for purposes of comparison we have included the known characterizations of the corresponding $\mathsf{Leaf}^P(\cdot)$ classes. We have defined $SAC^{(\log\log n)}$ as the class of languages accepted by $DLOGTIME$-uniform families of $(\log\log n)$-depth polynomial size circuits with unbounded fan-in OR gates and binary AND gates. Although we are not aware of a circuit characterization of the complexity class $NLOGTIME$, it is interesting to note that $\mathsf{Leaf}^L(NLOGTIME) = \mathsf{Leaf}^L(SAC^{(\log\log n)})$. See the next section for the precise definitions of these and of other classes appearing on Figure 1.

Now consider the class $Balanced$-$\mathsf{Leaf}^P(\cdot) \subseteq \mathsf{Leaf}^P(\cdot)$ defined by the restriction to nondeterministic polynomial time machines M for which, given x and i, it is easy to compute the sequence of nondeterministic choices which led M to produce the ith symbol in $leafstring^M(x)$ (see next section for the precise definition). The classes studied in Hertrampf *et al.* (1993) are in fact the $Balanced$-$\mathsf{Leaf}^P(\cdot)$ classes. In this paper we note that the Hertrampf *et al.* results indeed apply to the $\mathsf{Leaf}^P(\cdot)$ classes as well, as reported in Figure 1. On the other hand, there are classes C for which the classes *Balanced*-$\mathsf{Leaf}^P(C)$ and $\mathsf{Leaf}^P(C)$ are probably different. For example, we prove that *Balanced*-$\mathsf{Leaf}^P(\Sigma_k^{LogT}) = \Sigma_k^P \subseteq \mathsf{Leaf}^P(\Sigma_k^{LogT})$; recent work of Hertrampf, Vollmer and Wagner (1995) indicates that the inclusion is probably strict, since they show that the latter class corresponds to Σ_k^P with a PP-oracle. In fact, we state a sufficient condition, namely closure of C under padding, for $\mathsf{Leaf}(C) =$ *Balanced*-$\mathsf{Leaf}(C)$ to hold (for any of L, P or $LogT$). Note that each class C from Figure 1 is closed under padding, except for $POLYLOGTIME$, $SAC^{(\log\log n)}$, Σ_k^{LogT}, and AC^0.

Section 2 in this paper defines the terminology and addresses the issue of *Balanced*-$\mathsf{Leaf}(\cdot)$ versus $\mathsf{Leaf}(\cdot)$ classes. Section 3 forms the bulk of the paper and contains our characterizations of the $\mathsf{Leaf}^L(\cdot)$ classes. Section 4 describes our results on the $\mathsf{Leaf}^{LogT}(\cdot)$ classes. Section 5 concludes.

2 Preliminaries and definitions

The length of a string x is denoted $|x|$, and $<\cdot,\cdot>$ denotes any reasonable pairing function. For $w \in \{0,1\}^*$, $[w]$ represents the integer whose binary representation is w.

2.1 *Classes*

The context-sensitive, context-free, deterministic context-free and regular languages are denoted *CSL*, *CFL*, *DCFL* and *REG* respectively. The closure of the deterministic context-free (context-free) languages under logspace many-one reducibility is denoted by *LOGDCFL* (*LOGCFL*) (Sudborough (1978)). The syntactic monoid of a regular language is the transformation monoid of its minimal automaton. Group-free (or aperiodic) monoids are those which do not contain a nontrivial group; solvable monoids are those which do not contain a non-solvable group; dot-depth-k monoids, $k \geq 0$, provide a parametrization of all group-free monoids (Cohen and Brzozowski (1971)). The classes *SOLVABLE*, *PERIODIC SOLVABLE*, *PERIODIC* p-*GROUP* and *DOT DEPTH* k refer respectively to the classes of regular languages whose syntactic monoids are solvable monoids, solvable groups, groups of order a power of the prime p, and dot-depth-k monoids. The union over all k of *DOT DEPTH* k is the *DOT DEPTH HIERARCHY*.

As is customary in the case of Turing machines operating in sublinear time, access to the input is done via a special input index tape. Different input-read modes have appeared in the literature that play a delicate role when *DLOGTIME* and the various classes of the logtime hierarchy are considered (see Cai and Chen (1995) and Cai, Chen and Håstad (1996) for an overview of the different classes that arise and their separations). In the standard model that we refer to (see for example Buss (1987), Barrington, Immerman and Straubing (1990), Torán (1988)), the input index tape is handled as an ordinary read-write tape, and in a special state the Turing machine can access the input bit that is denoted by the current content of this tape (or will receive notice that there is no such bit, if the current content exceeds the input length). The input index tape is *not* erased after such a read operation, and in time $O(\log n)$ the machine can hence read up to $O(\log n)$ different input bits.

Define $\Pi_0^{LogT} = \Sigma_0^{LogT} = \mathit{DLOGTIME}$ and, for $k \geq 1$, let Σ_k^{LogT} denote the class of languages accepted by alternating logtime Turing machines which start in an existential configuration and which alternate at most $k-1$ times along any computation path, where we abuse notation and say that a deterministic Turing machine alternates -1 times. Let $\Pi_k^{LogT} = co\text{-}\Sigma_k^{LogT}$. We write $LH = \bigcup_k \Sigma_k^{LogT}$ for the Logtime Hierarchy (see Buss (1987), Barrington, Immerman and Straubing (1990)). The class *POLYLOGTIME* is the set of languages accepted by deterministic $O((\log n)^k)$ time-bounded Turing machines, $k \geq 1$. Following Buss (1987), Buss *et al.* (1992), for languages A and B, we say that $A \leq_{dlogtime} B$ iff for some function f increasing the length of strings only polynomially and many-one reducing A to B, the predicate "$Bit_f(c, i, x) \equiv$ the ith symbol of $f(x)$ is c" is recognized by some *DLOGTIME* Turing machine. By the class $\leq_{dlogtime}(B)$, we mean $\{A : A \leq_{dlogtime} B\}$, and for a class C, $\leq_{dlogtime}(C) = \bigcup_{B \in C} \leq_{dlogtime} (B)$.

The direct connection language of a circuit family (see Ruzzo (1981)) is a set of quadruples, where $<t, a, b, y>$ specifies that gate numbered a, of type t, is input to gate numbered b in the circuit handling inputs of length $|y|$; input gates are numbered $1, 2, \ldots, |y|$. A circuit family is *DLOGTIME*-uniform if its direct connection language is recognized in *DLOGTIME*. We define $SAC^{(\log \log n)}$ as the class of languages accepted by *DLOGTIME*-uniform families of $O(\log \log n)$-depth polynomial size circuits with unbounded fan-in *OR* gates and binary *AND* gates (with negated inputs available).

The class AC^0 is the set of languages accepted by some *DLOGTIME*-uniform family of polynomial size unbounded fan-in constant depth Boolean circuits (see Barrington, Immerman and Straubing (1990)). When MOD_q gates, computing 1 iff the sum of their inputs is divisible by q, are allowed for various q as well, the class ACC^0, or ACC, is obtained (see Barrington and Thérien (1988)). When *only* MOD_q gates are allowed for various q, the class obtained is CC^0 or "pure ACC" (see Barrington, Straubing and Thérien (1990), McKenzie, Péladeau and Thérien (1991), Yao (1990)). Finally, NC^1 is the class of languages accepted by *DLOGTIME*-uniform logarithmic-depth bounded-fan-in circuits, and we say that a language A is NC^1-reducible to another language B, denoted $A \leq_{NC^1} B$, iff there is a *DLOGTIME*-uniform logarithmic-depth bounded-fan-in circuit with oracle nodes for B that accepts A. This is the notion of NC^1-Turing reducibility introduced by Cook (1985). By the class $\leq_{NC^1}(B)$, we mean $\{A : A \leq_{NC^1} B\}$, and for a class C, $\leq_{NC^1}(C) = \bigcup_{B\in C} \leq_{NC^1}(B)$.

As usual, the classes Σ_k^P, $k \geq 0$, are the levels of the Polynomial Hierarchy PH (see Stockmeyer (1977)). Fix $q \geq 1$. The class $\mathrm{Mod}_q P$ (see Papadimitriou and Zachos (1983), Beigel and Gill (1992)) is the set of languages Y such that for some nondeterministic polynomial time Turing machine M, $x \in Y$ iff the number of accepting paths of M on x is divisible by q. If we replace NP by $\mathrm{Mod}_q P$ in the Stockmeyer construction of PH, we obtain levels which we denote $\mathrm{Mod}_q^k P$, $k \geq 0$. We define $\mathrm{Mod}_q^* P = \bigcup_{k\geq 0} \mathrm{Mod}_q^k P$. Then the class $\mathrm{Mod}^* P$ defined in Hertrampf *et al.* (1993) equals $\bigcup_{q\geq 1} \mathrm{Mod}_q^* P$. Finally, if we replace NP by $NP \cup \mathrm{Mod}_q P$ in the Stockmeyer construction, and define level $k+1$ as having access to oracles from the Boolean closure of level k, then the union over all $k \geq 0$ and all $q \geq 1$ of the levels obtained is the class Mod-PH also defined in Hertrampf *et al.* (1993).

We adapt the hierarchy definitions of the previous paragraph to the setting of logarithmic space by considering the oracle access mechanism defined by Ruzzo, Simon and Tompa (1984). In this model, the query tape has to be written on deterministically, or, equivalently, it is subject to the space bound, but the oracle answering service is given access to the machine input as well. When NP is replaced by NL, the resulting hierarchy Σ_k^L collapses to NL as a result of Immerman (1988) and Szelepcsényi (1988). Let $\mathrm{Mod}_q L$ (see Buntrock *et al.* (1992)) be defined just like $\mathrm{Mod}_q P$ except with a logarithmic space bound on the machine M. Adapting the definitions of $\mathrm{Mod}_q^k P$, $\mathrm{Mod}_q^* P$, $\mathrm{Mod}^* P$, and Mod-PH to the setting of logarithmic space and using the Ruzzo-Simon-Tompa oracle mechanism yields classes which we denote $\mathrm{Mod}_q^k L$, $\mathrm{Mod}_q^* L$, $\mathrm{Mod}^* L$, and Mod-*LogSH*. When q is prime, $\mathrm{Mod}_q^* L = \mathrm{Mod}_q L$ and in fact $\mathrm{Mod}_q^* P = \mathrm{Mod}_q P$ as a result of Beigel and Gill (1992), Buntrock *et al.* (1992).

We define *FDLOGTIME* as the class of functions $f : \Sigma^* \to \Sigma^*$ which are computable by a deterministic logtime Turing machine M in the following sense: if $f(w) = b_1 b_2 \dots b_k$, $b_i \in \Sigma$, then M on input w is able to produce the sequence $b_1, b_2, \dots, b_k$ on an output tape, all within its logarithmic time bound. Similarly, we write FP (FL) ($FPSPACE$) for the class of functions computable in deterministic polynomial time (logarithmic space) (polynomial space). We also write FNC^1 for the class of NC^1-computable functions.

2.2 *Computation trees and leaf languages*

We say that a nondeterministic Turing machine M is *adequate* if

- on any input, no sequence of nondeterministic choices causes M to enter the same configuration twice,
- the nondeterministic choices of M are ordered in a way which only depends on the finite transition relation of M,
- in any halting configuration, M outputs an element from a fixed finite alphabet.

One can always simulate a nondeterministic Turing machine by an adequate one, by using time stamps to remove configuration cycles (a trick credited to Pippenger in Borodin *et al.* (1981); time stamps being part of the new configurations, the new computation graph on any input is acyclic), by adopting an arbitrary ordering on nondeterministic choices, and, say, by outputting 1 or 0 in a halting configuration C according to whether C is accepting or rejecting.

We denote by $\mathit{tree}(M,x)$ the unique rooted ordered tree obtained by "unravelling" in the obvious way the computation graph of an adequate machine M on input x. The rooted paths in $\mathit{tree}(M,x)$ are thus naturally ordered, and we write $\mathit{leafstring}^M(x)$ for the sequence of machine outputs corresponding to the ordered sequence of leaves of $\mathit{tree}(M,x)$. Observe that although adequate NL-machines have polynomial size computation graphs, their trees and leaf strings are generally exponentially larger, since NL-machine computation trees may contain many configurations exponentially often. With respect to tree size, NL-machines are thus like NP-machines.

Now fix a finite alphabet Σ and let $Y \subseteq \Sigma^*$. We define $\mathsf{Leaf}^P(Y)$ as the class of all languages $A \subseteq \{0,1\}^*$ such that, for some adequate nondeterministic polynomial time Turing machine M, $x \in A$ iff $\mathit{leafstring}^M(x) \in Y$. Using logarithmic space bounds instead, we define $\mathsf{Leaf}^L(Y)$, and using logarithmic time bounds we define $\mathsf{Leaf}^{LogT}(Y)$. Now $\mathsf{Leaf}(\text{class } C) = \bigcup_{Y \in C}$ $\mathsf{Leaf}(Y)$.

The following lemma will be useful later. It also motivates the "balanced" restriction to the $\mathsf{Leaf}(Y)$ classes, defined below. (For the notation see the beginning of the preliminaries.)

Lemma 2.1 *Consider an adequate (1) NP-machine M, (2) NL-machine M, (3) NLOGTIME-machine M with at most two nondeterministic choices, labelled 0 and 1, from any configuration. The function*

$$\begin{aligned} \mathsf{path}_M : \{0,1\}^* &\longrightarrow \{0,1\}^* \\ < x, w > &\longmapsto \begin{cases} 0 & \text{if } \mathit{tree}(M,x) \text{ has fewer than } [w] \text{ leaves,} \\ c_1 \ldots c_m & \text{otherwise,} \end{cases} \end{aligned}$$

belongs to (1) FPSPACE, (2) FP, (3) FNC^1, where $c_1 \ldots c_m$, $c_i \in \{0,1\}$, is the sequence of nondeterministic choices leading from M's initial configuration on input x to the $[w]$th leaf in tree(M,x).

Proof. Cases (1), (2): The key idea is that in the stated resource bound we can compute the number $l(M,C,x)$ of leaves subtended by any accessible configuration C of M on x. Indeed let a configuration C be accessible via a certain sequence σ of nondeterministic choices. Then a machine which first simulates the operation of M on x using the fixed sequence σ, and from then on simulates M, accepting along all paths, has a number of accepting paths equal to $l(M,C,x)$. Computing this number given x and σ is a *#L* computation (see Àlvarez and Jenner (1993)), hence an *FP* computation, in the case of an *NL*-machine M (resp. a *#P* computation (see Valiant (1979)), hence an *FPSPACE* computation, in the case of an *NP*-machine M).

We exploit this key idea by following a deterministic path in M's computation graph on x, starting at the root, and keeping a count of the number of leaves to the "left of the path" being formed in *tree*(M,x). Let N be a node with left child N_0 and right child N_1, and denote by n the number of leaves already determined as lying to the left of the path leading from the root down to N. To continue our path we select N_1 iff $[w]$ exceeds n plus the number of leaves subtended by N_0. This concludes the proof because path lengths are polynomial for both *NL*-machines and *NP*-machines.

For the case (3), we argue differently. Making use of the fact that there are only polynomially many different sequences $c = c_1 c_2 \ldots c_m$, we first compute for each such sequence the corresponding leaf number w_c using the function $\# : \{0,1\}^* \longrightarrow \{0,1\}^*$ defined by

$$\#(< x, c >) := \begin{cases} w & \text{if sequence } c \text{ leads to the } [w]\text{th leaf in } \mathit{tree}(M,x), \\ 0 & \text{otherwise.} \end{cases}$$

To obtain $\mathsf{path}(< x, w >)$ we compute $\#(< x, c >)$ for all sequences $c \in \{0,1\}^*$ of length at most $k \log n$ in parallel and select c_s for which $w = \#(< x, c_s >)$. Clearly, if $\# \in \mathit{FNC}^1$, then so is path. To see that $\# \in \mathit{FNC}^1$, let $\mathit{valid}(< x, c >)$ be the predicate that is true iff the sequence $c \in \{0,1\}^*$ leads precisely to a leaf in $\mathit{tree}(M,x)$. The predicate *valid* is computable in $\mathit{DLOGTIME}$ and hence in AC^0. Observe that $\#(< x, c >) = \Sigma_{c' \leq c}\ \mathit{valid}(< x, c' >)$, if $\mathit{valid}(< x, c >)$, and 0, otherwise, where $c' \leq c$ iff c' is lexicographically smaller than c. With $k \cdot \log n$ the time bound of M, $\#(< x, c >)$ is computed by checking $\mathit{valid}(< x, c' >) \wedge c' \leq c$ for all $c \in \{0,1\}^*$ of length at most $k \log n$ in parallel, and summing up all the values. This can be done with an NC^1-circuit (in fact, with a TC^0-circuit (see Barrington, Immerman and Straubing (1990)) that can be made $\mathit{DLOGTIME}$-uniform by standard methods. □

Now suppose that M is a machine such that, for each x, $\mathit{tree}(M,x)$ is a complete binary tree. Then path_M is easily computable since, if it is nontrivial, $\mathsf{path}_M(x,w)$ is w preceded with an easily computed number of leading zeroes. (In detail, the precise depth d of $\mathit{tree}(M,x)$ can be computed by simulating M on x using the all-zero sequence of nondeterministic choices; if $[10^d] < [w]$ then $\mathsf{path}_M(x,w) = 0$.) The ease with which path can be computed is crucial to some characterizations. We therefore define the following restrictions to the $\mathsf{Leaf}(Y)$ classes. We say that an adequate (1) *NP*, (2) *NL*, (3) *NLOGTIME* machine is *balanced* iff path_M belongs to (1) *FP*, (2) *FL*, (3) *FDLOGTIME*. Then we define (1) $\mathit{Balanced}\text{-}\mathsf{Leaf}^P(Y)$, (2) $\mathit{Balanced}\text{-}\mathsf{Leaf}^L(Y)$, (3) $\mathit{Balanced}\text{-}\mathsf{Leaf}^{LogT}(Y)$, to be the class of all languages $A \subseteq \{0,1\}^*$ such that, for some adequate *balanced* nondeterministic (1) polynomial time, (2) logspace, (3) logtime Turing machine M, $x \in A$ iff $\mathit{leafstring}^M(x) \in Y$.

Note that $\mathit{Balanced}\text{-}\mathsf{Leaf}^P(\mathrm{Y}) = \mathcal{R}_m^{\mathrm{p,bit}}(Y)$ of Hertrampf *et al.* (1993) The following observations show that all the Hertrampf *et al.* results reported in Figure 1 indeed hold as well in the $\mathsf{Leaf}^P(\cdot)$ setting.

Let $Y \subseteq \Sigma^*$ with $\$ \notin \Sigma$. For a string $w = a_1 a_2 \ldots a_k \in \Sigma^k$, define $\$(w)$ to be the language represented by the regular expression $\$^* a_1 \$^* a_2 \$^* \ldots \$^* a_k \$^*$. Then define $\mathit{pad}(Y) = \bigcup_{w \in Y} \$(w)$. Say a class of languages C is closed under padding iff $(\forall Y \in C)[Pad(Y) \in C]$.

Proposition 2.2 *Except for POLYLOGTIME, $SAC^{(\log\log n)}$, Σ_k^{LogT} and AC^0, each class C listed on Figure 1 is closed under padding.*

Proof. Any class that is characterized by a machine model that can fully read its input and distinguish between padding and non-padding symbols is closed under padding. Non-closure of the named classes under padding is an observation of an anonymous referee. Indeed, over the singleton alphabet $\{0\}$, $(00)^* \in AC^0 \cap SAC^{(\log\log n)} \cap DLOGTIME$. But $Pad((00)^*)$ is essentially $PARITY$, which is not in $AC^0 \cup \Sigma_k^{LogT}$ by Atjai (1983) and Furst, Saxe and Sipser (1984). Now $SAC^{(\log\log n)}$ has depth-two unbounded fan-in circuits of size $n^{O(\log n)}$ since one bottom existential gate may receive as input universal gates, each one checking the $O(2^{c\cdot\log\log n}) = O((\log n)^c)$ leaves of one of the $O(n^{c\cdot\log n})$ possible proof trees for an input of length n, for c a constant. (A proof tree for an input w in a circuit C is a minimal subcircuit of C which includes the output gate of C and which has one wire into each OR gate and two wires into each AND gate.) But Lupanov (1961) proves that $PARITY$ cannot be computed by depth-two unbounded fan-in circuits of size $n^{O(\log n)}$, showing that $Pad((00)^*)$ does not belong to $SAC^{(\log\log n)}$ either.

Similarly, $0^{\log n}\{0,1\}^*$ is contained in *DLOGTIME* and thus in *POLYLOGTIME*, because a logtime machine can query its first $\log n$ input bits. But $pad(0^{\log n}\{0,1\}^*)$ is not contained in *POLYLOGTIME*, because any of the first polylogarithmic many input bits queried by a polylogtime machine might all be \$'s and the machine would run out of time before it could even begin counting 0's. $\square$

Proposition 2.3 *Let $*$ denote either the superscript L, P, or LogT. For any language Y, $\mathsf{Leaf}^*(Y) \subseteq$ Balanced-$\mathsf{Leaf}^*(pad(Y))$. For any class C that is closed under padding, $\mathsf{Leaf}^*(C) =$ Balanced-$\mathsf{Leaf}^*(C)$.*

Proof. Let $A \in \mathsf{Leaf}^*(Y)$ via a machine M, in either the polytime, logspace, or logtime setting. On input x, an upper bound t on the computation time of M can be computed in polytime, logspace, or logtime respectively. Hence M can be simulated by a machine M' whose computation tree is a complete binary tree of depth t. In this tree, all leaves output \$ except leftmost branches of a subtree originating from a leaf in $tree(M,x)$. Thus M' accepts A, using the leaf language $pad(Y)$, since $leafstring^M(x) \in Y \iff leafstring^{M'}(x) \in Pad(Y)$. Hence $\mathsf{Leaf}^*(Y)$

$\subseteq$ *Balanced*-$\mathsf{Leaf}^*(Pad(Y))$. Thus if Y belongs to a class C closed under padding, $\mathsf{Leaf}^*(Y) \subseteq$ *Balanced*-$\mathsf{Leaf}^*(C)$, proving that $\mathsf{Leaf}^*(C) \subseteq$ *Balanced*-$\mathsf{Leaf}^*(C)$. $\square$

So we only need to worry about *Balanced*-$\mathsf{Leaf}(C)$ classes separately from the $\mathsf{Leaf}(C)$ classes when C is unknown to be (or known not to be) closed under padding.

3 Classes $\mathsf{Leaf}^L(\cdot)$

In this section it is shown that a variety of well-known complexity classes are expressible in terms of leaf languages of nondeterministic logspace machines. This is true for resource-bounded as well as context-free and regular leaf languages.

3.1 Resource-bounded leaf languages

Our first theorem shows that any class in the Polynomial Hierarchy is related to its corresponding class in the Logtime Hierarchy via leaf languages of logspace machines or balanced polynomial-time machines. The proof is by padding techniques similar to those in Hertrampf *et al.* (1993) (Section 7) for polynomial time bounds.

Theorem 3.1 *For each $k \geq 0$, $\mathsf{Leaf}^L(\Sigma_k^{LogT}) =$ Balanced-$\mathsf{Leaf}^L(\Sigma_k^{LogT}) =$ Balanced-$\mathsf{Leaf}^P(\Sigma_k^{LogT}) = \Sigma_k^P$.*

Proof. *Balanced*-$\mathsf{Leaf}^L(\Sigma_k^{LogT}) \subseteq \mathsf{Leaf}^L(\Sigma_k^{LogT}) \cap$ *Balanced*-$\mathsf{Leaf}^P(\Sigma_k^{LogT})$ is a trivial inclusion. We argue that $\mathsf{Leaf}^L(\Sigma_k^{LogT}) \cup$ *Balanced*-$\mathsf{Leaf}^P(\Sigma_k^{LogT}) \subseteq \Sigma_k^P \subseteq$ *Balanced*-$\mathsf{Leaf}^L(\Sigma_k^{LogT})$. For the first inclusion let $A \in \mathsf{Leaf}^L(\Sigma_k^{LogT}) \cup$ *Balanced*-$\mathsf{Leaf}^P(\Sigma_k^{LogT})$ via a $p(n)$ time-bounded NL-machine or a balanced NP-machine M with leaf language $Y \in \Sigma_k^{LogT}$. Let M_Y be the alternating logtime machine that accepts Y with $k-1$ alternations. Construct another polynomial time machine M' that on input x simulates M_Y on $\mathit{leafstring}^M(x)$. This machine M' alternates whenever M_Y alternates. Whenever M_Y queries the input with a string w on its index tape, M' starts a deterministic simulation of M that computes the output of M along the path leading M to the $[w]$th leaf of $\mathit{tree}(M,x)$. In the case of $A \in \mathsf{Leaf}^L(\Sigma_k^{LogT})$, $\mathsf{path}_M(x,w) \in FP$ by Lemma 2.1. In the case of $A \in$ *Balanced*-$\mathsf{Leaf}^P(\Sigma_k^{LogT})$,

$\mathsf{path}_M(x,w) \in FP$ by assumption. Clearly, M' is a $(k-1)$-alternation machine which accepts iff $\mathit{leafstring}^M(x) \in Y$. Now observe that the logtime machine M_Y running on $\mathit{leafstring}^M(x)$ takes $O(\log(2^{p(|x|)}))$ steps. Hence the time bound of M' is polynomial, since there are $O(\log(2^{p(|x|)}))$ simulations of M that together consume $O(p(|x|) \cdot q(|x|))$ steps, where $q(n)$ is the polynomial that bounds the time of a simulation. This proves that $A \in \Sigma_k^P$.

Now, for the second inclusion, let $A \in \Sigma_k^P$ via an alternating Turing machine M which has at most two choices from any configuration, which has a read-only input tape, and which operates in polynomial time $p(n)$. Let $x = b_1 b_2 \dots b_n$, $b_i \in \{0,1\}$, and design an *NL*-machine M' in such a way that $\mathit{leafstring}^{M'}(x) = \$b_1\$b_2\$\$\$b_3\$^7 b_4 \dots \$^{2^{n-1}-1} b_n \$^{2^{p(n)}-2^n}$, that is, the ith bit of x is the 2^ith bit of $\mathit{leafstring}^{M'}(x)$. To do this, M' counts to $p(n)$, nondeterministically branching two ways, denoted the 0th and 1st choices respectively, at each step. At a leaf, M' outputs b_i for some i, $1 \le i \le n$, iff this leaf was obtained by the sequence of choices $0^{p(n)-i}1^i$; otherwise M' outputs \$. This yields the desired *NL* machine M'. This machine is balanced because its computation tree is a complete binary tree (the computation of $p(n)$ can be done as the branching is carried out). Now, consider a machine M'' with an index tape for accessing its input. The new machine M'' operates like M except when M reads b_i ; then M'' reads the 2^ith bit of $\mathit{leafstring}^{M'}(x)$ by querying it via its input index tape. (Observe that any such read operation of M'' requires at most a constant number of steps, since the contents of the index tape are not deleted.) Then M'' accepts $\mathit{leafstring}^{M'}(x)$ iff M accepts x. Now in terms of $n'' = |\mathit{leafstring}^{M'}(x)| = 2^{p(n)}$, the Σ_k^P-machine M'' operates in time $O(p(n)) = O(\log n'')$, thus becoming a Σ_k^{LogT}-machine. Hence $A \in \mathit{Balanced}\text{-}\mathsf{Leaf}^L(\Sigma_k^{LogT})$. □

Theorem 3.1 implies for unbalanced polynomial-time machines that $\Sigma_k^P \subseteq \mathsf{Leaf}^P(\Sigma_k^{LogT})$. However, this inclusion is probably strict, since Hertrampf, Vollmer, and Wagner (1994) show that $\mathsf{Leaf}^P(\Sigma_k^{LogT}) = {\Sigma_k^P}^{PP}$ for each $k \ge 0$, that is, a *PP* or *#P* oracle is needed to compute the function path of Lemma 2.1.

Corollary 3.2 *(i)* $\mathsf{Leaf}^L(\mathit{DLOGTIME}) = P$;
(ii) $\mathsf{Leaf}^L(\mathit{NLOGTIME}) = \mathsf{Leaf}^L(\le_{dlogtime}(OR)) = NP$;
(iii) $\mathsf{Leaf}^L(AC^0) = PH$.

Proof. The last statement follows from $AC^0 = LH = \bigcup_{k \ge 0} \Sigma_k^{LogT}$ (see Barrington, Immerman and Straubing (1990)). The first statement and part of the second statement are the cases $k = 0$ and $k = 1$ of Theorem 3.1. Now it is seen that $Y \leq_{dlogtime} OR$ iff $Y \in NLOGTIME$, as follows. Let a language Y be accepted by a $c \log n$ time-bounded $NLOGTIME$-machine M with at most two nondeterministic choices, the 0th and the 1st, from any configuration. For any input x and for any string $w \in \{0,1\}^{\lceil c \log |x| \rceil}$, let $s_{x,w}$ equal 1 if w has a prefix which is an accepting path of M on input x, and 0 otherwise. Observe that $s_{x,w}$ is computable in $DLOGTIME$ from $<x, w>$. Then $Y \leq_{dlogtime} OR$ by means of the function f which maps x to $s_{x,w_1} s_{x,w_2} s_{x,w_3} \cdots s_{x,w_{2^{\lceil c \log |x| \rceil}}}$, where w_i is the length-$\lceil c \log |x| \rceil$ binary representation of $i - 1$. Conversely, let $Y \leq_{dlogtime} OR$ by means of a function f. An $NLOGTIME$-machine on input x can guess a position i in the string $f(x)$ and verify that the corresponding bit $f_i(x)$ is 1 by simulating the $DLOGTIME$-machine that computes the predicate associated with f. □

The next corollary follows from the *proof* of Theorem 3.1, which can be adapted to the situation in which the time bound to accept the leaf languages is polylogarithmic, i.e. $O((\log n)^k)$, or to the situation in which the number of alternations is not constant.

Corollary 3.3 *(i)* $\mathsf{Leaf}^L(POLYLOGTIME) = P$;
(ii) $\mathsf{Leaf}^L(NC^1) = \mathsf{Leaf}^L(ALOGTIME) = PSPACE$.

Proof. It suffices to argue that $\mathsf{Leaf}^L(POLYLOGTIME) \subseteq P$ to prove the first statement, and this is immediate from the proof of Theorem 3.1. The second statement follows since $NC^1 = ALOGTIME$ (Ruzzo (1981)) and $APTIME = PSPACE$ (Chandra, Kozen, and Stockmeyer (1981)). □

To our knowledge no circuit characterization for $NLOGTIME$ is known. Nevertheless, the next theorem shows that for a characterization of NP in terms of leaf languages, instead of $NLOGTIME$ we may take the class of languages accepted by semi-unbounded $(\log \log n)$ depth-bounded polynomial-size circuits.

Theorem 3.4 $\mathsf{Leaf}^L(SAC^{(\log \log n)}) = \textit{Balanced}\text{-}\mathsf{Leaf}^P(SAC^{(\log \log n)}) = NP$.

Proof. We begin with $\mathsf{Leaf}^L(SAC^{(\log \log n)}) \subseteq NP$. Via the NL-machine M with leaf language $Y \in SAC^{(\log \log n)}$, let $A \in \mathsf{Leaf}^L(SAC^{(\log \log n)})$. To determine whether $x \in A$, an NP-machine M' guesses a "proof tree" of the semi-unbounded fan-in circuit C deciding whether $\textit{leafstring}^M(x) \in Y$. (A proof tree for $\textit{leafstring}^M(x)$ in C is a minimal subcircuit of C which includes the output gate of C and which has one wire into each OR gate and two wires into each AND gate, and which evaluates to 1.) Now C is an exponential size circuit in terms of $|x|$, but if $\mathit{leafstring}^M(x) \in Y$, then some proof tree for C, expressible using at most

$$O(2^{k\cdot\log\log(|\mathit{leafstring}^M(x)|)}) \cdot O(\log(|\mathit{leafstring}^M(x)|)) \text{ bits},$$

which is $O(2^{k\cdot\log\log(2^{cp(|x|)})} \cdot \log(2^{cp(|x|)})) = O((p(|x|))^{k+1})$ bits, evaluates to 1 on $\mathit{leafstring}^M(x)$, where $p(n)$ bounds the running time of M and k and c are constants. Hence M' can guess and then evaluate such a proof tree. Note that M' can compute the length of $\mathit{leafstring}^M(x)$ because this is a $\#L$ computation (Àlvarez and Jenner (1993)). M' can thus identify which circuit in the uniform family is relevant. M' can also verify connections in the proof tree it has guessed; each verification takes time $O(\log(2^{cp|x|})) = O(p(|x|))$ by the $DLOGTIME$ uniformity criterion for C. Care is needed at the inputs to the proof tree. When M' needs to know the value of the symbol at the ith position in $\mathit{leafstring}^M(x)$, it makes use of Lemma 2.1, using polynomial time to compute the path leading M to its ith leaf, and using polynomial time again to simulate M and to determine the value output by M at its ith leaf. This proves that $A \in NP$.

Essentially the same proof shows that $\mathit{Balanced}\text{-}\mathsf{Leaf}^P(SAC^{(\log\log n)}) \subseteq NP$. Then M is a balanced NP-machine, and M' can determine the length of $\mathit{leafstring}^M(x)$ by binary search, using the assumption that $\mathsf{path}_M \in FP$. Similarly, M' obtains the $[w]$th symbol of $\mathit{leafstring}^M(x)$ by simulating M on the nondeterministic choices $\mathsf{path}_M(x, w)$, in polynomial time.

For the reverse inclusions, we argue that $NLOGTIME \subseteq SAC^{(\log\log n)}$. The result then follows from Corollary 3.2, noting that the latter corollary also holds in the $\mathit{Balanced}\text{-}\mathsf{Leaf}^P(\cdot)$ case. Now to simulate a nondeterministic logtime machine, we use a Savitch (1970)-like argument, evaluating a predicate $P(i, j, m)$ recursively, where $P(i, j, m)$ holds iff configuration j of M is reachable from configuration i in m steps. Noting that $P(i, j, m)$ holds iff there exists k such that $P(i, k, \lceil m/2 \rceil)$ and $P(k, j, \lfloor m/2 \rfloor)$ hold, we use the unbounded fan-in OR gates to handle the existential quantifier. Since the maximum relevant m is $O(\log n)$, the recursion depth and thus the semi-unbounded fan-in circuit depth is $O(\log\log n)$. The resulting circuit can routinely be made $DLOGTIME$-uniform. $\square$

So far in this section we have shown that for various leaf language classes C, the classes $\mathsf{Leaf}^L(C)$ and $\mathit{Balanced}\text{-}\mathsf{Leaf}^P(C)$ coincide. The following general theorem states that closure under $\leq_{dlogtime}$ reductions is a sufficient condition for this to hold. Note that the condition may not be necessary since, for example, such a closure property is not known about the classes in Theorem 3.1.

First observe that unbalanced logspace machines can be simulated by balanced polynomial-time machines without changing the leaf language.

Proposition 3.5 *$\mathsf{Leaf}^L(Y) \subseteq$ Balanced-$\mathsf{Leaf}^P(Y)$ for any language Y.*

Proof. By Lemma 2.1, $\mathsf{path}_M \in FP$ for any adequate *NL*-machine M. Then it suffices to view M accepting $A \in \mathsf{Leaf}^L(Y)$ as an *NP*-machine. □

Bovet, Crescenzi and Silvestri (1992) and Hertrampf *et al.* (1993) observed that

Proposition 3.6 *Balanced-$\mathsf{Leaf}^P(\leq_{polylogtime}(Y))$ = Balanced-$\mathsf{Leaf}^P(Y)$ for any language Y.*

(In Bovet, Crescenzi and Silvestri (1992) the following equivalence was even shown: $Y \leq_{polylogtime} Z$ iff *Balanced*-$\mathsf{Leaf}^P(Y) \subseteq$ *Balanced*-$\mathsf{Leaf}^P(Z)$ holds in all relativized worlds.)

We now show that closure under $\leq_{dlogtime}$ is sufficient to relate polynomial time and logspace leaf languages.

Theorem 3.7 *Balanced-$\mathsf{Leaf}^L(\leq_{dlogtime}(Y))$ = Balanced-$\mathsf{Leaf}^P(Y)$ for any language Y.*

Proof. The left to right inclusion follows from Proposition 3.6. For the other inclusion, let $Y \subseteq \Sigma^*$ be the leaf language of a $p(n)$ time-bounded balanced *NP* machine M, and let M_p be the $q(n)$ time-bounded deterministic machine that computes the function path_M. We design, as in the proof of Theorem 3.1, an *NL*-machine M' that on input $x = b_1 b_2 \dots b_n$, $b_i \in \Sigma$ produces $\mathit{leafstring}^{M'}(x) = \$b_1\$b_2\$\$\$b_3\$^7 b_4 \dots \$^{2^{n-1}-1} b_n \$^{2^{p(n)+q(n)}-2^n}$ of length exactly $2^{p(n)+q(n)}$. This machine M' is balanced because it has complete binary computation trees.

Now it suffices to show that a direct access *DLOGTIME* machine N can "reduce" $\mathit{leafstring}^{M'}(x)$ to $\mathit{leafstring}^{M}(x)$. In other words, N must determine whether the ith symbol of $\mathit{leafstring}^{M}(x)$ is c, given as input $< c, i, \mathit{leafstring}^{M'}(x) >$. Note that the time available to N is $O(p(n)+q(n))$. Now N can extract $b_1 b_2 \dots b_n$ from $\mathit{leafstring}^{M'}(x)$ by using its direct access mechanism in order to read the input symbols at positions $2^0, 2^1, \dots, 2^{n-1}$ and copy them onto some storage tape. From then on, N can operate as an ordinary machine when reading x. N then computes $\mathsf{path}_M(x, i)$ in time

$q(n)$. If $\mathsf{path}_M(x,i) = 0$, N rejects. Otherwise, in time $O(p(n))$, N then simulates M on the sequence of the nondeterministic choices $\mathsf{path}_M(x,i)$ and accepts iff M outputs c. $\square$

Corollary 3.8 *For an arbitrary language Y the following classes coincide: Balanced-$\mathsf{Leaf}^P(Y)$, and Balanced-$\mathsf{Leaf}^L(\leq_{dlogtime}(Y))$, $\mathsf{Leaf}^L(\leq_{dlogtime}(Y))$, Balanced-$\mathsf{Leaf}^L(\leq_{polylogtime}(Y))$, $\mathsf{Leaf}^L(\leq_{polylogtime}(Y))$.*

Corollary 3.8 allows us to sharpen various results obtained for polynomial-time leaf languages by Hertrampf *et al.* (1993) (see Figure 1).

Corollary 3.9 *(i) $\mathsf{Leaf}^L(POLYLOGSPACE) = PSPACE$;*
(ii) $\mathsf{Leaf}^L(P) = EXPTIME$;
(iii) $\mathsf{Leaf}^L(NP) = NEXPTIME$;
(iv) $\mathsf{Leaf}^L(CSL) = EXPSPACE$.

Proof. (i)-(iii) and the left to right inclusion of (iv) follow from Corollary 3.8 and Figure 1. For the remaining inclusion $EXPSPACE \subseteq \mathsf{Leaf}^L(CSL)$, we apply the padding technique of Theorem 3.1. $\square$

These results extend Corollary 3.3, where it was shown that there are logspace leaf languages in NC^1 that capture $PSPACE$, to classes of leaf languages that include NC^1.

3.2 Context-free leaf languages and regular leaf languages

Turning to the expressive power of context-free languages, our first result is somewhat surprising. It states that NL-machines are powerful enough to accept any $PSPACE$ language by submitting carefully constructed leaf strings to a deterministic pushdown automaton.

Theorem 3.10 *$\mathsf{Leaf}^L(DCFL) = \mathsf{Leaf}^L(CFL) = PSPACE$.*

Proof. Clearly $\mathsf{Leaf}^L(DCFL) \subseteq \mathsf{Leaf}^L(CFL) \subseteq \mathsf{Leaf}^P(CFL) \subseteq PSPACE$, where the last inclusion is from Hertrampf *et al.* (1993). To prove $PSPACE \subseteq \mathsf{Leaf}^L(DCFL)$, we show how to construct an NL-machine and a $DCFL$ to recognize the set of true Boolean sentences of the form $\exists x_1 \forall x_2 \ldots \exists x_{n-1} \forall x_n [C_1 \vee \ldots \vee C_n]$, where each C_i is a conjunction of three litterals.

We consider an NL-machine M that behaves as follows:

Stage 1: M makes n nondeterministic binary choices, each of the 2^n lexicographically ordered paths corresponding to an n-bit string. Along the path $i_{11}i_{12}\ldots i_{1n} \in \{0,1\}^*$ M determines the Boolean value of C_1 when variable $x_1 = i_{11}$, variable $x_2 = i_{12}$, and so on, which we denote $C_1(x_{i_{11}}, x_{i_{12}}, \ldots, x_{i_{1n}})$. When this stage is over, *tree*(M) has 2^n leaves respectively containing the 1-bit information $B_{i_{11}i_{12}\ldots i_{1n}} = C_1(x_{i_{11}}, x_{i_{12}}, \ldots, x_{i_{1n}})$.

Stage 2: From each leaf of the previous stage, M again makes n choices. Along path $i_{21}i_{22}\ldots i_{2n}$ issued from leaf $i_{11}i_{12}\ldots i_{1n}$, M determines the bit $B_{i_{11}\ldots i_{1n}i_{21}\ldots i_{2n}} = C_1(x_{i_{11}}, \ldots, x_{i_{1n}}) \vee C_2(x_{i_{21}}, \ldots, x_{i_{2n}})$. After this stage, the computation tree has 2^{2n} leaves.

Stages 3 to n**:** M continues in this way for n stages. At this point, M has 2^{n^2} leaves respectively having the 1-bit value $B_{i_{11}\ldots i_{1n}\ldots\ldots i_{n1}\ldots i_{nn}} = C_1(x_{i_{11}}, \ldots, x_{i_{1n}}) \vee \ldots \vee C_n(x_{i_{n1}}, \ldots, x_{i_{nn}})$.

We are only interested in the 2^n leaves holding the values

$$C_1(x_{i_1}, \ldots, x_{i_n}) \vee \ldots \vee C_n(x_{i_1}, \ldots, x_{i_n}),$$

i.e. we want to check if

$$\exists x_1 \forall x_2 \ldots \exists x_{n-1} \forall x_n [B_{i_1\ldots i_n i_1 \ldots i_n \ldots\ldots i_1 \ldots i_n}].$$

The leftmost leaf is relevant because it is $B_{0^{n^2}}$ which has the correct form. The next relevant leaf $B_{0^{n-1}10^{n-1}1\ldots 0^{n-1}1}$ is at a distance

$$2^0 + 2^n + 2^{2n} + \ldots + 2^{n(n-1)} = (2^{n^2} - 1)/(2^n - 1).$$

Similarly, all subsequent relevant leaves are separated by that same distance.

M now enters another stage. From each leaf l, M makes $n-1$ nondeterministic choices. Then on each but the leftmost path from l it makes one more nondeterministic choice and produces a marker \$; on the leftmost path from l, it does not move and keeps the bit from l. Thus a former leaf labelled b gives rise to $2^n - 1$ new leaves spelling out $b\$^{2^n-2}$. This has the consequence that in the new leaf string, the relevant leaves are separated by $2^{n^2} - 1$ positions, with the leftmost leaf being relevant.

In the final stage, M expands each node labelled b, for $b \neq \$$, into a subtree with leaf substring $b\#^{2^{n^2}-1}$. Hence, if after the first n stages the leaf string was $b_1 b_2 \ldots b_s \in \{0,1\}^*$, it now has the form $b_1\#^\alpha\$^\beta b_2\#^\alpha\$^\beta \ldots b_s\#^\alpha\$^\beta$, where $\alpha = 2^{n^2} - 1$, $\beta = 2^n - 2$.

Note that a DPDA can easily access the 2^n relevant positions. Indeed, the 1st position is relevant. Then push all # onto the stack. Next, for each symbol in $\{0, 1, \$\}$ that is read, pop a # from the stack. When the stack is empty the next bit is relevant.

To conclude the proof we observe that the truth of the quantified sentence can be decided by evaluating a perfect binary tree of depth n, with OR at the root, with levels alternating between AND and OR and with the leaves labelled by $\phi(x_1 \dots x_n) = C_1(x_1 \dots x_n) \vee \dots \vee C_n(x_1 \dots x_n)$ in lexicographical order. Thus if our leaf string had ANDs and ORs appropriately placed so that these operands together with the relevant bits would correspond to the postfix normal form of the formula, we would be done: a DPDA could simply traverse the string, taking into account only the relevant bits, and evaluating the postfix sentence as it goes along.

But this is easily done: consider the 2^n leaves at the end of the first stage and note that in the final leafstring, there will be exactly one relevant bit in each of the 2^n corresponding subtrees. The number of logical operators to insert after the k^{th} relevant bit is i iff the binary expansion of k ends with 01^i. The NL-machine can compute the value i on a special work tape as it performs the first stage and keep this value throughout the subsequent steps. The final stage can be modified as follows; after all markers have been inserted, each leaf containing a value b in $\{0, 1\}$ is expanded into a subtree with leaf substring consisting of b followed by i logical operators, beginning with AND and alternating between AND and OR (noting that n is always even), where i is the value kept on the special tape. When the DPDA will traverse the leafstring, each time a relevant bit is found, the operators following it will be processed; operators placed after non-relevant bits are simply to be skipped. $\square$

We now consider regular leaf languages. Determining membership of the leaf string of an *NL*-machine M in a regular language Y amounts to evaluating a circuit over the transformation monoid of the DFA of Y. This observation and known results on circuit evaluation (see Beaudry *et al.* (1993)) form the basis of the proofs of the next three theorems.

Contrasting with Theorem 3.10 and with the polynomial time setting, where $\mathsf{Leaf}^P(REG)$=$\mathsf{Leaf}^P(DCFL)$=$\mathsf{Leaf}^P(CFL)$=$PSPACE$, our next theorem shows that in terms of logspace leaf languages the gap between regular and deterministic context-free languages is significant.

Theorem 3.11 $\mathsf{Leaf}^L(REG) = P$.

Proof. Let $A \in \mathsf{Leaf}^L(REG)$, via an adequate *NL*-machine M and via the DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ accepting M's leaf language. Associate with each $a \in \Sigma$ the map $\delta_a : Q \to Q$ and in the usual way extend the definition to δ_w for any $w \in \Sigma^*$. Clearly $x \in A$ iff $\delta_{leafstring^M(x)}$ maps q_0 to a state in F.

Now consider the polynomial size computation *graph* of M on x. This graph can be viewed as a circuit over the finite semigroup $\{f : Q \to Q\}$ in the sense of Beaudry *et al.* (1993). The (ordered) inputs to a circuit gate N are the (ordered) successors of N in the graph. The input gates to the circuit are the halting configurations of M. When viewed as a circuit input gate, a halting configuration which outputs $a \in \Sigma$ produces the input value δ_a. Finally, the task of each internal gate in the circuit is to perform the composition of the functions $f : Q \to Q$ it receives (in the obvious order). It should be clear that $\delta_{leafstring^M(x)}$ is the value computed at the root of this circuit. Now each composition operation takes constant time, hence a straightforward bottom-up evaluation of the circuit shows that $A \in P$.

To show $P \subseteq \mathsf{Leaf}^L(REG)$, we use the fact that evaluating a circuit over any fixed non-solvable group G is P-complete Beaudry *et al.* (1993). Let it be required to determine whether such a circuit C over G evaluates to a prescribed element $g \in G$. We design an *NL* machine M which guesses an ordered path from the output gate of C to an input gate i of C, and which outputs the element of G attached to input gate i. Then C evaluates to g iff $leafstring^M(< C, g >)$ belongs to the regular language $\{w \in G^* \mid w \text{ multiplies out to } g\}$. Hence a P-complete problem is in $\mathsf{Leaf}^L(REG)$, and clearly the latter class is closed under reductions as coarse as logspace. □

We obtain characterizations of some important subclasses of NC^2 by considering subclasses of the regular languages.

Theorem 3.12 *(i) For each $k \geq 1$, $\mathsf{Leaf}^L(DOT\ DEPTH\ k) = NL$; hence $\mathsf{Leaf}^L(DOT\ DEPTH\ HIERARCHY) = NL$;*
(ii) $\mathsf{Leaf}^L(PERIODIC\ p\text{-}GROUP) = MOD_pL$.

Proof. (i) Obviously $NL = \mathsf{Leaf}^L(\{0,1\}^*1\{0,1\}^*)$, so the inclusion $NL \subseteq \mathsf{Leaf}^L(DOT\ DEPTH\ 1)$ is trivial. For the converse, let $A \in \mathsf{Leaf}^L(K)$ via an *NL*-machine M, where the transformation monoid T of the minimal DFA $\mathcal{A}$ of K is group-free, i.e., in the *DOT DEPTH HIERARCHY*. As in the proof of Theorem 3.11, determining whether $x \in A$ reduces to the problem of evaluating a circuit over T. As shown by Beaudry *et al.* (1993), this problem is in the logspace hierarchy and hence in NL, since NL is closed under complementation (Immerman (1988), Szelepcsényi (1988)).

(ii) The proof proceeds exactly as in the NL case. This time, we appeal to a theorem of Beaudry *et al.* (1993) stating that evaluating a circuit over a p-group can be done in MOD_pL. $\square$

Theorem 3.13 *(i)* $\mathsf{Leaf}^L(\mathit{PERIODIC\ SOLVABLE}) \subseteq \mathit{Mod}^*L$; *(ii)* $\mathsf{Leaf}^L(\mathit{SOLVABLE}) \subseteq \mathit{Mod\text{-}LogSH} \subseteq \mathit{DET} \subseteq NC^2$.

Proof. As in the previous argument, noting that the proof that circuit evaluation over a solvable group of exponent q (resp. solvable monoid) can be done in the complexity class DET as shown by Beaudry *et al.* (1993), in fact puts the problem in Mod_q^*L (resp. Mod-$LogSH$). $\square$

4 Classes $\mathsf{Leaf}^{LogT}(\cdot)$

Logtime leaf languages and $DLOGTIME$ reductions are closely related. The following theorem makes the relationship precise.

Theorem 4.1 *For any language Y we have $Y \subseteq$ Balanced-$\mathsf{Leaf}^{LogT}(Y) \subseteq \leq_{dlogtime}(Y) \subseteq$ Balanced-$\mathsf{Leaf}^{LogT}(Y\$^*) \subseteq \leq_{dlogtime}(Y\$^*) \subseteq \mathsf{Leaf}^{LogT}(Y\$^*) \subseteq \leq_{NC^1}(Y)$.*
In particular, for any class C that is closed under (i) padding on the right, (ii) NC^1-reductions, we have (i) Balanced-$\mathsf{Leaf}^{LogT}(C) = \leq_{dlogtime}(C)$; (ii) $\mathsf{Leaf}^{LogT}(C) = C$.

Proof. For the first inclusion, we argue that a balanced $NLOGTIME$ machine M can reproduce its input x as $\mathit{leafstring}^M(x)$. To do this, M first deterministically computes $|x|$ in $DLOGTIME$ using the slick binary search technique described by Buss (1987). With the aid of $|x|-1$ written out in binary on a tape, M can then proceed in nondeterministic logtime to produce a $\mathit{tree}(M,x)$ having exactly $|x|$ leaves. (This requires a bit of care. M begins by reading the most significant bit of $|x|-1$ in "seeing" mode. When a 1 is read in seeing mode, M branches two ways, entering "blind" mode along the 0th branch and remaining in seeing mode along the 1st branch. When a 0 is read in seeing mode, only the 0th branch is taken and the mode is unchanged. When either a 0 or a 1 is read in blind mode, M branches two ways and remains in blind mode. The result of successively reading each bit of $|x|-1$ according to the above instructions will yield a tree with $|x|$ leaves.) While doing this, M can copy the binary choices made along a path onto its index tape to form an integer i. When the bottom of the path is reached, M reads the ith symbol of x and immediately outputs it. Observe that

$$\mathsf{path}_M(x,w) = \begin{cases} 0 & \text{if } |x| < [w]+1, \\ 0^t w & \text{otherwise,} \end{cases}$$

where $t + |w|$ is the time taken by M along any computation path. Hence $\mathsf{path}_M \in \mathit{FDLOGTIME}$, because $|x| < [w]+1$ can be tested, and t can be computed by simulating M on the all-zero sequence of nondeterministic choices.

For the proof of the second inclusion, let $A \in \mathit{Balanced}\text{-}\mathsf{Leaf}^{\mathit{LogT}}(\mathrm{Y})$ via an $\mathit{NLOGTIME}$-machine M. Define the function f which maps x to $\mathit{leafstring}^M(x)$. Then f many-one reduces A to Y, since $x \in A \iff \mathit{leafstring}^M(x) = f(x) \in Y$. Furthermore, the predicate $\mathit{Bit}_f(c,i,x)$, stating that the ith symbol $f_i(x)$ of $f(x)$ is c, is computable in $\mathit{DLOGTIME}$ by simulating M on x following the ith path, since $\mathsf{path}_M \in \mathit{FDLOGTIME}$ by assumption. This proves that $A \leq_{\mathit{dlogtime}} Y$.

Turning to the third inclusion, $\leq_{\mathit{dlogtime}}(Y) \subseteq \mathit{Balanced}\text{-}\mathsf{Leaf}^{\mathit{LogT}}(Y\$^*)$. let $A \leq_{\mathit{dlogtime}} Y$ via a function f bounded by the polynomial p whose associated predicate $\mathit{Bit}_f(c,i,x)$ is $\mathit{DLOGTIME}$-computable by a machine M. Then $|f(x)| \leq p(|x|)$ holds for all x. Let p be bounded by the function h defined by $h(x) = 2^{k||x||}$, where $||x||$ denotes the length of the binary representation of $|x|$ and k is a constant. h is computable in $\mathit{FDLOGTIME}$, since on input x with binary search over the input tape positions we can compute the binary representation of $n = |x|$ and using time $O(\log n)$ we can furthermore compute the product $k|n|$ in unary notation, which gives us the number of zeros in the binary representation of $h(x)$.

Now we design an $\mathit{NLOGTIME}$ machine M' which, on input x, has exactly $h(x)$ paths and outputs the ith bit of $f(x)$ along each path i such that $0 \leq i < |f(x)| - 1$ and \$ on each path i such that $|f(x)| \leq i \leq h(x) - 1$. Then $x \in A \iff f(x) \in Y \iff \mathit{leafstring}^{M'}(x) \in Y\$^*$, proving that $A \in \mathsf{Leaf}^{\mathit{LogT}}(Y\$^*)$. M' operates as follows. It first computes $h(x)$ in $\mathit{DLOGTIME}$. It then applies the technique discussed in the proof of the first inclusion to guess along its ith path an integer i, $0 \leq i < h(x)$. Finally, M' computes its output along the ith path by simulating M on $<c,i,x>$ for each c. If M always rejects, M' outputs \$.

The fourth inclusion follows from the second, and the fifth from the third.

Now for the last inclusion, let $A \in \mathsf{Leaf}^{LogT}(Y\$^*)$ by an $NLOGTIME$ machine M. Since $\mathsf{path}_M \in FNC^1$ by Lemma 2.1 and since M can be simulated in NC^1 on a given input x along a given path, an FNC^1 circuit can compute $leafstring^M(x)$. Then if $leafstring^M(x) \neq y\$^j$ for some y not containing \$, the circuit rejects. Otherwise, the length of y is computed and y is fed into an oracle node for Y of appropriate length (the circuit contains a layer of oracle nodes for Y, one for each possible length of y). The outcome of this oracle node determines acceptance or rejection of x. This shows $A \leq_{NC^1} Y$.

Finally, if (i) C is closed under padding on the right, the first four inclusions imply that $Balanced\text{-}\mathsf{Leaf}^{LogT}(C) = \leq_{dlogtime}(C)$; if (ii) C is closed under NC^1-reducibility, then the last inclusion implies $\mathsf{Leaf}^{LogT}(\mathrm{C}) \subseteq C$, and the first implies $C \subseteq \mathsf{Leaf}^{LogT}(\mathrm{C})$. □

Let us remark that $Balanced\text{-}\mathsf{Leaf}^{LogT}(Y) = \leq_{dlogtime}(Y)$ holds for any language Y, if we require the $\leq_{dlogtime}$-reductions f to be bounded in length by a function h in *FDLOGTIME*; i.e., satisfying that $|f(x)| = h(x)$ for all x. Then an $NLOGTIME$ machine can compute $h(x)$ on input x, branch into $h(x)$ paths computing on path i the ith bit of $f(x)$ by simulating the $DLOGTIME$ machine for $Bit_f(c, i, x)$ on all (of the constant many) c. To be sure, having such a function h (which acts as an additional "uniformity condition") seems to be a stronger condition than having only a polynomial bound on the length of f.

Corollary 4.2 *(i) For any* $k \geq 0$, $Balanced\text{-}\mathsf{Leaf}^{LogT}(\Sigma_k^{LogT}) \subseteq \Sigma_{k+1}^{LogT}$*; hence* $AC^0 = LH = Balanced\text{-}\mathsf{Leaf}^{LogT}(LH)$*;*
(ii) $Balanced\text{-}\mathsf{Leaf}^{LogT}(POLYLOGTIME) = POLYLOGTIME$.

Proof. (i) Buss (1987) shows that $DLOGTIME$ reducibility leads at most one level higher up in the logtime hierarchy. Then we apply the first and second inclusions of Theorem 4.1. (ii) follows from Theorem 4.1 (i). □

Corollary 4.3 *If C is either NC^1, POLYLOGSPACE, P, or NP, then* $C = \mathsf{Leaf}^{LogT}(C)$. □

Let us now again take C to be a class in the Chomsky Hierarchy, as we have done for leaf languages in the logspace setting in the previous section, revealing a significant gap in complexity between regular and deterministic context-free languages. The following theorem shows that leaf languages in the logtime setting characterize exactly those complexity classes for which the corresponding membership problem is complete with respect to logtime reducibility.

Theorem 4.4 *(i)* $\mathsf{Leaf}^{LogT}(REG) = NC^1$,
(ii) $\mathsf{Leaf}^{LogT}(DCFL) = LOGDCFL$,
(iii) $\mathsf{Leaf}^{LogT}(CFL) = LOGCFL$,
(iv) $\mathsf{Leaf}^{LogT}(CSL) = PSPACE$.

Proof. (i) Since REG is closed under padding, we obtain from Proposition 2.3 and Theorem 4.1(i) that $\mathsf{Leaf}^{LogT}(REG) = \leq_{dlogtime}(REG)$. But there are regular languages that are complete for NC^1 with respect to $\leq_{dlogtime}$ reducibility, and hence $\leq_{dlogtime}(REG) = NC^1$ (Barrington (1989)).

(ii) to (iv) are proved analogously. The completeness properties are folklore for (iv), they can be found in Bédard, Lemieux and McKenzie (1993) for (iii), and in Muscholl (1992) for (ii). □

Our last theorem deals with subclasses of the regular languages.

Theorem 4.5 *(i)* $\mathsf{Leaf}^{LogT}(SOLVABLE) = ACC^0$;
(ii) $\mathsf{Leaf}^{LogT}(PERIODIC\ SOLVABLE) \subseteq CC^0$;
(iii) $\mathsf{Leaf}^{LogT}(DOT\ DEPTH\ HIERARCHY) = AC^0$.

Proof. We only sketch the proof of (i). Let $A \in \mathsf{Leaf}^{LogT}(SOLVABLE)$. Since $SOLVABLE$ is closed under padding, we can assume by Proposition 2.3 that $A \in Balanced\text{-}\mathsf{Leaf}^{LogT}(SOLVABLE)$ via a balanced logtime machine M. Since $DLOGTIME \subseteq AC^0$ (Barrington, Immerman and Straubing (1990)), an AC^0 circuit on input x can compute $leafstring^M(x)$ by appropriate parallel simulations of M. But then deciding whether $leafstring^M(x)$ belongs to a solvable regular language can be done in ACC^0 as shown by Barrington and Thérien (1988). Hence $A \in ACC^0$.

For the converse, let $A \in ACC^0$. By the results of Barrington and Thérien (1988), there exists $Y \in SOLVABLE$ such that $A \leq_{dlogtime} Y$. Now by Theorem 4.1, $\leq_{dlogtime}(Y) \subseteq \mathsf{Leaf}^{LogT}(Y\$^*) \subseteq \mathsf{Leaf}^{LogT}(SOLVABLE)$, where the last inclusion holds because $SOLVABLE$ is closed under padding on the right.

The other cases are similar. $DLOGTIME \subseteq CC^0$ is an observation of Alexis Maciel (personal communication). We do not obtain $CC^0 \subseteq \mathsf{Leaf}^{LogT}(PERIODIC\ SOLVABLE)$ because $PERIODIC\ SOLVABLE$ is not closed under padding on the right. □

5 Conclusion

We have shown that a large variety of complexity classes between NL and $EXPSPACE$ may be characterized by leaf languages of logspace machines, that is, as $\mathsf{Leaf}^L(C)$ for appropriate C of significantly smaller complexity. These characterizations nicely extend the characterizations previously obtained via leaf languages of polynomial-time machines whose characterization capabilities range only above P (see Figure 1). Hence, logspace leaf languages could provide means of using the interreducibility properties of concrete low-level problems within NC^1, say for example of specific regular languages C, to draw information about the interreducibility of the corresponding high-level problems $\mathsf{Leaf}^L(C)$ within NC^2, or alternatively to draw information about possible (relativized) separations. Concerning the latter, work remains to be done in order to determine whether the results of Bovet, Crescenzi and Silvestri (1991, 1992) about machine-independent relativizations in the polynomial time setting can be applied in the logspace setting. This is challenging in particular within NC^2, because here we are confronted with a multitude of computational reducibility models (deterministic and nondeterministic logspace machines, unbounded and bounded fan-in circuits, etc.).

Our results about logspace leaf languages have shown that unlike in the case of the classes $\mathsf{Leaf}^P(C)$ studied by previous authors (see Bovet, Crescenzi and Silvestri (1991, 1992), Hertrampf *et al.* (1993), Borchert (1994), Hertrampf (1994)), applying a $DLOGTIME$ reduction to C can affect the class $\mathsf{Leaf}^L(C)$ considerably. In fact, we have shown that $DLOGTIME$ reductions are sufficient to push logspace leaf language classes up to the level of polytime leaf languages classes. One point of view explaining this sensitivity of $\mathsf{Leaf}^L(Y)$ to $DLOGTIME$ reductions is the following. Although NL-machines behave like NP-machines with respect to computation tree size, in general each configuration of an NL-machine gives rise to exponentially many nodes in the machine's computation tree. Hence the tree of an NL-machine is built from many identical subtrees. If the leaf language Y is such that the "overall effect" of each subtree arising from a given configuration C of the machine is identical to the effect of any other subtree arising from C, then this effect can be recovered by referring to C alone. Now, when a $DLOGTIME$-reduction shuffles the leaf bits around, such a recovery is no longer possible and we need the paths themselves to figure out the contribution of each leaf. In this case, we might as well have used an NP-machine.

Leaf languages for nondeterministic logtime machines, in contrast, seem less interesting for further research. We have shown that, minor technical differences aside, they behave like *DLOGTIME* reductions and hence are not able to relate problems differing widely in complexity.

For leaf languages defined from syntactic monoids, we have seen the central role played by the circuits over algebraic structures studied in Beaudry *et al.* (1993). In some cases however, the exact relationship between such leaf languages and circuit evaluation remains to be worked out precisely.

Recently, some new work on the topic of leaf languages has been done. It was shown independently by Borchert and Lozano (1995) and by Veith (1996) that characterizing a problem via a leaf language A of a polynomial time machine is basically equivalent to characterizing the problem via a succinct representation of A, thereby relating the topic of leaf languages to another well-studied research topic. (Here the succinct respresentation $S(A)$ of a language A is the set of coded pairs (c, m), where c is a circuit and m a positive integer, such that the length-m prefix of the word $w(c)$ is contained in A, where $w(c)$ is the concatenation of the value of all of the 2^n assignments to the n input variables for c in lexicographical order (see Borchert and Lozano (1995)). This equivalence does not carry over to the case of logspace leaf languages unless P=L, but it would be interesting to see whether leaf languages in the logspace setting turn out to have a similar characterization, for example by considering finite automata instead of circuits, and by letting the word w represent the behavior of the automata of n states on the first 2^n strings, or by some other concept.

In another recent paper, Caussinus *et al.* (1996) study the leaf language mechanism for finite automata and show that the results of Section 3 carry over to this model. They furthermore extend Theorem 3.10, showing that deterministic linear and, respectively, one-counter context-free leaf languages suffice to yield *PSPACE*. Their paper also exploits leaf language characterizations in order to deduce unconditional separations between low-level and high-level complexity classes, obtaining for example that $ACC^0 \neq \text{Mod-}PH$.

Acknowledgement

We thank Hervé Caussinus for helpful comments. Thanks also go to the two anonymous referees for their very detailed and valuable comments. One of the referees provided the proof of the non-closure-under-padding results of Proposition 2.2. The first and the third author gratefully acknowledge the hospitality and support of the Secció Informàtica Teòrica of the Dept. Llenguatges i Sistemes Informàtics, Universitat Politècnica Catalunya, Barcelona.

References

ÀLVAREZ, C. AND JENNER, B. (1993), A very hard log-space counting class, *Theoretical Computer Science* **107** (1993), 3-30.

ANGLUIN, D. (1980), On counting problems and the polynomial-time hierarchy, *Theoretical Computer Science* **12** (1980), 161–173.

AJTAI, M. (1983), Σ_1^1 formulae on finite structures, *Ann. Pure Appl Logic,* **24** (1983), 1–48.

BAKER, T. GILL, J., AND SOLOVAY, R. (1975), Relativizations of the P=?NP question, *SIAM J. on Computing* **4**, (1975), 431–442.

BAKER, T. AND SELMAN, A. (1979), A second step toward the polynomial hierarchy, *Theoretical Computer Science* **8** (1979), 177–187.

BARRINGTON, D. M. (1989), Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1, *J. Computer and Systems Science* **38** (1989), 150-164.

BARRINGTON, D. M., IMMERMAN, N., STRAUBING, H. (1990), On uniformity within NC^1, *J. Computer and Systems Science* **41** (1990), 274-306.

BARRINGTON, D. M., STRAUBING, H., AND THÉRIEN, D. (1990), Non-uniform automata over groups, *Information and Computation* **89, 2** (1990), 109-132.

BARRINGTON, D. M. AND THÉRIEN, D. (1988), Finite monoids and the fine structure of NC^1, *J. of the Association for Computing Machinery* **35** (1988), 941-952.

BEAUDRY, M., MCKENZIE, P., PÉLADEAU, P., AND THÉRIEN, D. (1993), Finite monoids: from word to cirucit evaluation, To appear in *SIAM J. on Comput.* (Prelim. version appeared as: Circuits with monoidal gates, Proc. of the 10th Symp. on Theoretical Aspects of Computer Science, *Springer Lecture Notes in Comp. Sci. 665* (1993), 555-565.)

BÉDARD, F., LEMIEUX, F. AND MCKENZIE, P. (1993), Extensions to Barrington's M-program model, *Theoretical Computer Science A* **107**, 1 (1993), 31-61.

BEIGEL, R. AND GILL, J. (1992), Counting classes: thresholds, parity, mods, and fewness, *Theoretical Computer Science* **103** (1992), 3-24.

BORCHERT, B. (1994), On the acceptance power of regular languages, Proc. of the 11th Symp. on Theoretical Aspects of Computer Science, *Springer Lecture Notes in Comp. Sci. 775* (1994), 533,541.

BORCHERT, B. AND LOZANO, A. (1995) Succinct circuit representation and leaf languages are basically the same concept, Technical Report Nr. 17, Mathematisches Institut, Universität Heidelberg, July 1995.

BORODIN, A., FISHER, M. J., KIRKPATRICK, D. G., LYNCH, N. A. AND TOMPA, M. (1981), A time-space tradeoff for sorting on non-oblivious machines, *J. Computer and Systems Science* **22** (1981), 351-364.

BOVET, D., CRESCENZI, P. AND SILVESTRI, R. (1991), Complexity classes and sparse oracles, *Proc. of the 6th IEEE Structure in Complexity Theory Conference* (1991), 102-108.

BOVET, D., CRESCENZI, P. AND SILVESTRI, R. (1992) A uniform approach to define complexity classes, *Theoretical Computer Science* **104** (1992), 263-283.

BUNTROCK, G., DAMM, C., HERTRAMPF, U. AND MEINEL, C. (1992), Structure and importance of Logspace-MOD-classes, *Math. Syst. Theory* **25** (1992), 223-237.

BUSS, S. R. (1987), The Boolean formula value problem is in ALOGTIME, *Proc. of the 19th ACM Symp. on the Theory of Computing* (1987), 123-131.

BUSS, S. R., COOK, S., GUPTA, A. AND RAMACHANDRAN, V. (1992), An optimal parallel algorithm for formula evaluation, *SIAM J. on Computing* **21**, 4 (1992), 755-780.

CAI, L. AND CHEN, J. (1995), On input read-modes of alternating Turing machines, *Theoretical Computer Science* **148** (1995), 33–55.

CAI, L., CHEN, J. AND HÅSTAD, J. (1996), Circuit bottom fan-in and computational power, to appear in *SIAM J. on Comput.*

CAUSSINUS, H., MCKENZIE, P., THÉRIEN, D. AND VOLLMER, H. (1996), Nondeterministic NC^1computation, to appear in: *11th Proc. Computational Complexity Conf.*, IEEE, 1996,

CHANDRA, A., KOZEN, D. AND STOCKMEYER, L. (1981). Alternation, *J. of the Association for Computing Machinery* **28** (1981), 114-133.

COHEN, R. AND BRZOZOWSKI, J. (1971), Dot-depth of star-free events, *J. Computer and Systems Science* **5** (1971), 1-16.

COOK, S. A. (1985), A taxonomy of problems with fast parallel solutions, *Information and Computation* **64** (1985), 2-22.

FURST, M. L, SAXE, J. B., AND SIPSER, M. (1984), Parity, circuits, and the polynomial-time hierarchy, *Math. Syst. Theory* **17** (1984), 13-27.

HÅSTAD, J. (1986), Almost optimal lower bounds for small-depth circuits, *Proc. 18th ACM STOC,* (1986), pp. 6–20.

HERTRAMPF, U. (1994), Complexity classes with finite acceptance types, Proc. of the 11th Symp. on Theoretical Aspects of Computer Science, *Springer Lecture Notes in Comp. Sci. 775* (1994), 543-553.

HERTRAMPF, U., LAUTEMANN, C., SCHWENTICK, T., VOLLMER, H. AND WAGNER, K. (1993), On the power of polynomial time bit-reductions, *Proc. of the 8th IEEE Structure in Complexity Theory Conference* (1993), 200-207.

HERTRAMPF, U., VOLLMER, H. AND WAGNER, K. (1994), On balanced vs. unbalanced computation trees, Technical Report No. 82, Institut für Informatik, Universität Würzburg, 1994, to appear in *Math. Syst. Theory*, 1995.

IMMERMAN, N. (1988), Nondeterministic space is closed under complementation, *SIAM J. on Computing* **17**, 5 (1988), 935-938.

LUPANOV, O. (1961), Implementing the algebra of logic functions in terms of constant-depth formulas in the basis $+$, $*$, $-$. English translation in *Sov. Phys.-Dokl.* **6**, 2 (1961); orig. in *Dokla. Akad. Nauk SSSR* **136**, 5.

MCKENZIE, P., PÉLADEAU, P. AND THÉRIEN, D. (1991) NC^1: The Automata-Theoretic Viewpoint, *Computational Complexity* **1** (1991), 330-359.

MUSCHOLL, A. (1992), Characterizations of LOG, LOGDCFL and NP based on groupoid programs, Extended abstract, Institut für Informatik, Universität Stuttgart, (1992) 13 pages.

PAPADIMITRIOU, C. AND ZACHOS, S. (1983), Two remarks on the power of counting, *Proc. 6th GI conf. on Theoretical Computer Science*, Springer LNCS vol. 145 (1983), 269-276.

RUZZO, W. (1981), On uniform circuit complexity, *J. Computer and Systems Science* **22** (1981), 365-383.

RUZZO, W., SIMON, J. AND TOMPA, M. (1984), Space-bounded hierarchies and probabilistic computations *J. Computer and Systems Science* **28** (1984), 216-230.

SAVITCH, W. J. (1970), Relationships between nondeterministic and deterministic tape complexities, *J. Computer and Systems Science* **4** (1970), 177-192.

STOCKMEYER, L. (1977), The polynomial-time hierarchy, *Theoretical Computer Science* **3** (1977), 1-22.

SUDBOROUGH, I. H. (1978), On the tape complexity of deterministic context-free languages, *J. of the Association for Computing Machinery* **25** (1978), 405-414.

SZELEPCSÉNYI, R. (1988), The method of forced enumeration for nondeterministic automata, *Acta Informatica* **26** (1988), 279-284..

TORÁN, J. (1988), *Structural properties of the counting hierarchies*, Ph.D. Thesis, Tech. Rep. LSI-88-2, Universitat Polytècnica de Catalunya, 1988.

VALIANT, L. G. (1979), The complexity of computing the permanent, *Theoretical Computer Science* **8** (1979), 189–201.

VEITH, H. (1996), Succinct representation, leaf languages, to appear in *11th Proc. Computational Complexity Conf.*, IEEE, 1996.

VERSHCHAGIN, N. K. (1994) Relativizable and nonrelativizable theorems in the polynomial theory of algorithms, *Russian Acad.Sci.Izv.Math.*, **42** (1994), 261–298.

YAO, A. (1985), Separating the polynomial-time hierarchy by oracles, *Proc. 26th IEEE FOCS,* (1985), pp. 1–10.

YAO, A. (1990). On ACC and threshold circuits, *Proc. of the 31st IEEE Symp. on the Foundations of Computer Science* (1990), 619-627.