Nondeterministic NC^1 computation

Hervé Caussinus* and Pierre McKenzie*

Dép. d'informatique et recherche opérationnelle
Université de Montréal
C.P. 6128, Succursale Centre-ville
Montréal (Québec), H3C 3J7 Canada

Denis Thérien*

School of Computer Science
McGill University
3480 University Street
Montréal (Québec), H3A 2A7 Canada

Heribert Vollmer†

Theoretische Informatik
Universität Würzburg
Am Exerzierplatz 3
97072 Würzburg, Germany

Abstract

We define the counting classes $\#NC^1$, $GapNC^1$, PNC^1 and $C_{=}NC^1$. We prove that boolean circuits, algebraic circuits, programs over nondeterministic finite automata, and programs over constant integer matrices yield equivalent definitions of the latter three classes. We investigate closure properties. We observe that $\#NC^1 \subseteq \#L$, that $PNC^1 \subseteq L$, and that $C_{=}NC^1 \subseteq L$. Then we exploit our finite automaton model and extend the padding techniques used to investigate leaf languages. Finally, we draw some consequences from the resulting body of leaf language characterizations of complexity classes, including the unconditional separations of ACC^0 from $MOD\text{-}PH$ and that of TC^0 from the counting hierarchy. Moreover we obtain that if dlogtime-uniformity and logspace-uniformity for AC^0 coincide then the polynomial time hierarchy equals $PSPACE$.

*Work supported by NSERC of Canada and by FCAR du Québec.

†Research performed while the author held a visiting position at the Department of Mathematics, University of California at Santa Barbara, Santa Barbara, CA 93105. Supported by the Alexander von Humboldt Foundation under a Feodor Lynen scholarship.

1 Introduction

Counting classes have been studied extensively since the introduction of $\#P$ by Valiant [35]. A number of such classes were first defined in the context of polynomial time computation [24, 17]. Then, the logspace counting class $\#L$ was investigated [5], together with many logspace variants adapted from the polynomial time case. Recently, counting classes based on finite model theory and on one-way logspace have been considered as well [31, 13].

The starting point of the present paper is the following question: how should one define $\#NC^1$? This question is natural and interesting in view of the inclusion chain $NC^1 \subseteq L \subseteq NL \subseteq P \subseteq NP$ and in view of the tight relationship between computing the permanent (resp. determinant) and $\#P$ (resp. $\#L$). Given that the basic counting classes $\#P$ and $\#L$ are defined by counting the number of accepting paths in a nondeterministic computation, what "nondeterministic NC^1" accepting paths are there to be counted[1]?

We propose a definition inspired by Vinay's circuit characterization of $\#LOGCFL$ [36]. Thus we define $\#NC^1$ as the set of functions counting the number of accepting proof trees of NC^1 circuits. (This definition has also been suggested by Jia Jiao [22].) It is common folklore that this class equals the class of functions computed by logarithmic depth arithmetic circuits over the semiring of the natural numbers.

Motivated by Barrington's characterization of NC^1 via bounded-width branching programs of polynomial size [6], we also consider a class we call $\#BP$. (A better name perhaps would be $\#BWBP$ to make clear that we are talking about bounded width, but we chose to use the shorter $\#BP$.) This class is defined by counting accepting paths in nondeterministic branching programs (these are branching programs having the capability to branch nondeterministically without consuming an input bit). The class $\#BP$ can equivalently be defined by considering ordinary (i.e., deterministic) branching programs which output a sequence of elements from a given alphabet: This sequence is then given as input to a nondeterministic finite automaton, and we count accepting paths in that NFA.

We prove that $\#BP \subseteq \#NC^1$, and that when we consider the corresponding classes of differences of $\#BP$ and $\#NC^1$ functions (we will denote those classes by $GapBP$ and $GapNC^1$), we get equality, i.e., $GapBP = GapNC^1$. This result is proved in two steps: First, we observe that $\#BP$ can be char-

[1] It is well known that adding nondeterministic input bits to an NC^1 circuit results in a computation model powerful enough to solve NP-complete problems (see e.g. [31, 37]).

acterized by what we call programs over constant matrices; and then we use a result by Ben-Or and Cleve [11] who show that evaluating an arithmetic circuit reduces to iterated product of constant-size matrices. Our result is thus a continuation of a line of research started by Barrington who showed that for decision problems and for computing boolean functions, log depth circuits are no more powerful than bounded-width branching programs, and continued by Ben-Or and Cleve who generalized Barrington's result to arithmetic circuits vs. straight-line programs. Our observation is that the phenomenon still holds when counting becomes an issue.

We go on by examining our classes of functions as well as two related classes of sets $C_=NC^1$ and PNC^1 from a structural point of view. We exhibit some of their closure properties and we show that $GapNC^1$ has essentially all closure properties $GapL$ is known to have. This leads us to the conclusion that PNC^1 is closed under union and intersection. We also show that PNC^1 and $C_=NC^1$ are both included in logspace.

We then proceed to refine known leaf language characterizations of complexity classes, using the formal framework of leaf languages for branching programs (which we call NC^1 leaf languages). We argue that many characterizability results carry over from the polynomial time [19] and logspace [21] cases to that of NC^1. We then draw consequences from the leaf language characterizations. Some of these consequences could be made to follow from characterizations via polynomial time leaf languages known prior to the present paper, although we discovered them in the course of investigating the NC^1 leaf languages. These consequences include the unconditional separation of the circuit class ACC^0 from $MOD\text{-}PH$ (the oracle hierarchy defined using NP and all classes of the form MOD_qP as building blocks), the unconditional separation of TC^0 from the counting hierarchy (the oracle hierarchy with PP as building block), and the implication that if dlogtime uniformity and logspace uniformity coincide for AC^0, then the polynomial hierarchy equals $PSPACE$ (and hence the polynomial hierarchy collapses). Some of our separations have in the meantime been reproved by Eric Allender using "traditional" (i.e., non leaf-language) techniques [1].

Finally, we obtain two nontrivial technical improvements to the known leaf language characterizations of $PSPACE$: we prove that NC^1 leaf languages over one-counter deterministic context-free languages and NC^1 leaf languages over linear deterministic context-free languages each capture the class $PSPACE$.

Section 2 in this paper describes our computation models and defines the various complexity classes discussed above. Section 3 compares the new counting classes. Section 4 investigates their closure properties. Section 5 describes the new leaf language results, some consequences, and the tighter characterizations of *PSPACE*. Section 6 concludes with suggestions for further study.

2 Definitions

We fix a finite alphabet Σ. We write $|y|$ for the length of $y \in \Sigma^*$. We write $[n]$ for $\{1, 2, \ldots, n\}$. When Δ is a set we write Δ^Σ for $\{f : \Sigma \to \Delta\}$.

Logtime bounded Turing machines access their input via an index tape. This index tape is not erased after a read operation, and the machine detects any attempt to read past its input.

An *n-projection over* Δ is a finite sequence of pairs (i, f) where $1 \le i \le n$ and f is a function from Σ to Δ, such pairs being called *instructions*. The *length* of this sequence is denoted S_n, and its jth instruction is denoted $(B_n(j), E_n(j))$. A *projection over* Δ is a family $P = (P_n)_{n \in \mathbf{N}}$ of n-projections over Δ. We can consider P as a tuple $P = (\Sigma, \Delta, S, B, E)$, where S is a function from $\mathbf{N}$ to $\mathbf{N}$, B is a function from $\mathbf{N} \times \mathbf{N}$ to $\mathbf{N}$, and E is a function from $\mathbf{N} \times \mathbf{N}$ into Δ^Σ. We write $P(x)$ for the concatenation of the $(E_{|x|}(j))(x_{B_{|x|}(j)})$ for $1 \le j \le S_{|x|}$. P is *uniform* if the functions

$$\begin{aligned} S' : \Sigma^* &\to \mathbf{N} \\ y &\mapsto S(|y|), \end{aligned}$$

$$\begin{aligned} B' : \Sigma^* \times \mathbf{N} &\to \mathbf{N} \\ \langle y, j \rangle &\mapsto B(|y|, j), \end{aligned}$$

and

$$\begin{aligned} E' : \Sigma^* \times \mathbf{N} &\to \Delta^\Sigma \\ \langle y, j \rangle &\mapsto E(|y|, j) \end{aligned}$$

belong to *FDLOGTIME* (i.e. the complete logarithmic-length output can be computed in logarithmic time, which is stronger than requiring that each output bit *separately* be computable in logarithmic time, as is sometimes done), where integers are expressed in binary notation and $\langle \cdot, \cdot \rangle$ is a pairing function as in [7, Lemma 6.1]. A proof that uniform projections are transitive can be found in [16].

For the following definitions we fix a semiring $(R, +, \times)$. The semirings considered in this paper are the semiring $\mathbf{N}$ and the ring $\mathbf{Z}$.

We say that a function $f : \Sigma^* \to R$ *uniformly projects* to a function $g : \Delta^* \to R$ iff there is a uniform projection P such that for each $x \in \Sigma^*$, $f(x) = g(P(x))$. We say that a language $A \subseteq \Sigma^*$ *uniformly projects* to a language $B \subseteq \Delta^*$ iff the characteristic function of A uniformly projects to that of B. Note that uniform projection reducibility implies *DLOGTIME* reducibility as defined for example in [7].

Example. A "uniform projection over Δ" is nothing but a (uniform) projection, in the sense of [34], or a (uniform) M-program viewed as a mapping from Σ^* to Δ^* with $\Delta \subseteq M$, in the sense of [6, 29]. For example, refer to the M-program described on page 337 of [29], and let $\Sigma = \{0, 1\}$ and $\Delta = \{a, b, c, e\}$. Consider the instructions ν_i, $1 \leq i \leq 2\lfloor n/2 \rfloor$, and the set $F \subset M$, defined there. Then define $S(n) = 2\lfloor n/2 \rfloor$ for $n \in \mathbf{N}$, and define $B : \mathbf{N} \times \mathbf{N} \to \mathbf{N}$ and $E : \mathbf{N} \times \mathbf{N} \to \{f : \Sigma \to \Delta\}$ as prescribed by the instructions ν_i. The result is a uniform projection over Δ, showing that the language of palindromes over Σ uniformly projects to the regular language of strings over Δ which multiply out in the monoid M to some $m \in F$.

2.1 Circuits

A *boolean n-circuit* is a directed acyclic graph with nodes of indegree 0 each labeled with an element from $[n] \times \{0, 1\}^{\Sigma}$ and called inputs. The other nodes are boolean AND or OR gates of indegree 2. There is one node of outdegree 0, called the output.

A *boolean circuit* is a family $C = (C_n)_{n \in \mathbf{N}}$ of boolean n-circuits C_n. The *direct connection language* [30, 7] of C is the set of tuples $\langle t, a, b, y \rangle$, specifying that gate numbered a, of type t, is input to gate numbered b in $C_{|y|}$ (with appropriate conventions when a is an input gate). C is *uniform* if its direct connection language is in *DLOGTIME*. C computes a function from Σ^* to $\{0, 1\}$ in the usual way. The class NC^1 is the class of languages computed by uniform logarithmic depth circuits.

Let $C = (C_n)_{n \in \mathbf{N}}$ be a boolean circuit. A *proof tree* for input x is a connected subtree of the graph that results from unwinding the circuit C_n into a tree, having the following three properties: each OR gate has indegree one, each AND gate has indegree two, and it evaluates to 1. (That we unwind the circuit into a tree before counting proof trees follows recent literature, e.g. [22, 36].) We denote by $\#accept(C, x)$ the number of proof trees for x.

A function $f : \Sigma^* \to \mathbf{N}$ is in $\#NC^1$ iff there exists a uniform boolean circuit C such that $f(x) = \#\mathit{accept}(C, x)$.

A function $f : \Sigma^* \to \mathbf{Z}$ is in $GapNC^1$ iff it is the difference of two functions in $\#NC^1$.

Let $\mathcal{T} : \mathbf{N} \times \mathbf{N} \to \{0, 1\}$. A language $X \subseteq \Sigma^*$ is in $C_{\mathcal{T}}NC^1$ iff there exist two functions $f, g \in \#NC^1$ such that $x \in X \Leftrightarrow \mathcal{T}(f(x), g(x)) = 1$. We will consider only $C_=NC^1$ and $C_{\leq}NC^1$, in analogy with similar classes considered in the literature in the conetxt of other resource bounds.

2.2 Programs over nondeterministic automata

A *nondeterministic automaton* is a tuple of the form $(Q, \Delta, q_0, \delta, F)$ where Q is the finite set of states, Δ is the input alphabet, $q_0 \in Q$ is the initial state, F is the set of accepting states, and the transition function $\delta : Q \times \Delta \times \{1, \ldots, |Q|\} \to Q$ has the property that $\delta(q, a, i)$ is defined for some i and $\delta(q, a, j)$ is defined and distinct from $\delta(q, a, i)$ for $1 \leq j < i$. Note that the $[|Q|]$ component of the domain of δ represents the possible nondeterministic choices available at a particular time step. The largest i for which $\delta(q, a, i)$ is defined is denoted $\#\delta(q, a)$.

A *program over* $A = (Q, \Delta, q_0, \delta, F)$ is a projection $P = (\Sigma, \Delta, S, B, E)$. Given P, a *sequence of choices* for $x \in \Sigma^n$ is a sequence of integers $(c_j)_{j=1}^{S_n}$ that define a sequence of states $(q_j)_{j=1}^{S_n}$ such that for $1 \leq j \leq S_n$:

$$q_j = \delta(q_{j-1}, (E_n(j))(x_{B_n(j)}), c_j)$$

for $1 \leq c_j \leq \#\delta(q_{j-1}, (E_n(j))(x_{B_n(j)}))$. An *accepting* sequence of choices is such that $q_{S_n} \in F$. The number of accepting sequences of choices for x is denoted $\#\mathit{accept}(P, x)$. P accepts the language $\{w \in \Sigma^* \mid \#\mathit{accept}(P, x) > 0\}$. The class NBP is the class of languages recognized by uniform polynomial length programs over a nondeterministic automaton.

A function $f : \Sigma^* \to \mathbf{N}$ is in $\#BP$ iff there exists a uniform polynomial length program over an automaton such that $f(x) = \#\mathit{accept}(P, x)$.

A function $f : \Sigma^* \to \mathbf{Z}$ belongs to $GapBP$ if and only if it is the difference of two functions in $\#BP$.

Let $\mathcal{T} : \mathbf{N} \times \mathbf{N} \to \{0, 1\}$. A language $X \subseteq \Sigma^*$ is in $C_{\mathcal{T}}BP$ iff there exist two functions $f, g \in \#BP$ such that $x \in X \Leftrightarrow \mathcal{T}(f(x), g(x)) = 1$. Again, in the following we only consider $C_=BP$ and $C_{\leq}BP$.

2.3 Algebraic circuits

Let Δ be a finite subset of R. An *algebraic n-circuit* is a directed acyclic graph with nodes of indegree 0 labeled with an element from $[n] \times \Delta^{\Sigma}$ and called inputs. The other nodes have indegree 2, are labeled with $\times$ or $+$, and are called algebraic gates. There is one node of outdegree 0, called the output.

An *algebraic circuit* is a family $C = (C_n)_{n \in N}$ of algebraic n-circuits. C is *uniform* if its direct connection language is in *DLOGTIME*. C computes a function from Σ^* to R in the obvious way. The class $ALG^1(R)$ is the class of functions computed by uniform logarithmic depth algebraic circuits (the exponent 1 being used to indicate depth $O(\log n)$, in analogy with the exponent in NC^1).

Let $\mathcal{T} : R \to \{0, 1\}$. A language $X \subseteq \Sigma^*$ is in $C_{\mathcal{T}}ALG^1(R)$ iff there exists a function $f \in ALG^1(R)$ such that $x \in X \Leftrightarrow \mathcal{T}(f(x)) = 1$.

2.4 Programs over constant matrices

Let k be some integer. Let Δ be a finite set of $k \times k$ matrices, q_0 be a $1 \times k$ row vector and F be a $k \times 1$ column vector, with entries in the semiring R. A *program over constant matrices* (q_0, Δ, F) is a projection $P = (\Sigma, \Delta, S, B, E)$. P on input $x \in \Sigma^n$ computes $q_0\ (\prod_{j=1}^{S_n}(E_n(j))(x_{B_n(j)}))\ F$. The class $BPM(R)$ is the class of functions computed by uniform polynomial length programs over constant matrices over R.

Let $\mathcal{T} : R \to \{0, 1\}$. A language $X \subseteq \Sigma^*$ is in $C_{\mathcal{T}}BPM(R)$ iff there exists a function $f \in BPM(R)$ such that $x \in L \Leftrightarrow \mathcal{T}(f(x)) = 1$.

3 Counting circuits vs. counting branching programs

Theorem 3.1

$$\#BP = BPM(\mathbf{N}) \subseteq ALG^1(\mathbf{N}) = \#NC^1 \subseteq \#L.$$

Proof. The inclusion $\#BP \subseteq BPM(\mathbf{N})$ is straightforward. Indeed, let $Y \in \#BP$ by means of the uniform projection $P = (\Sigma, \Delta, S, B, E)$ and let $A = (Q, \Delta, q_0, \delta, F)$ be the underlying nondeterministic automaton. Each constant matrix M_a in the $BPM(\mathbf{N})$ instance is the $|Q| \times |Q|$ transition matrix induced on A by the letter $a \in \Delta$. The projection in the $BPM(\mathbf{N})$

instance is identical to P, except for the function E which differs from that in P only in the replacement of any $a \in \Delta$ by M_a.

To show $BPM(\mathbf{N}) \subseteq \#BP$, consider a $BPM(\mathbf{N})$ instance involving $k \times k$ matrices over $\{0, 1, \ldots, m\}$, noting that m, and more importantly k, are constants. For each such matrix M, one can construct a graph G_M composed of $l = 3 + \lfloor \log_2 m \rfloor$ layers of $3k^2$ nodes, such that the k^2 entries in M are the respective numbers of paths between the nodes labelled $1, 2, \ldots, k$ in the first layer of G_M and the nodes labelled $1, 2, \ldots, k$ in the last layer of G_M (see for instance [33] or [32, p. 42] for the crucial step in this construction). The automaton in the $\#BP$ instance will thus have $3k^2$ states and its alphabet will be $\{t : [3k^2] \to [3k^2]\}$. The uniformity machines must then simply "replace" any occurrence of a $k \times k$ matrix M by a string of $l-1$ mappings t corresponding to the l layers of G_M. In particular, if S_n (available in log time by hypothesis) is the length of the $BPM(\mathbf{N})$ instance, then the length of the $\#BP$ program will be $(l-1)S_n$. (Technically, it is convenient to assume with no loss of generality that $\lfloor \log_2 m \rfloor$ is a power of 2 in order for the uniformity machines to easily multiply and divide by $\lfloor \log_2 m \rfloor$, and to pad out the program to length $2\lfloor \log_2 m \rfloor S_n$ in order for the uniformity machines to easily extract from an instruction number the number of the corresponding graph layer). Note that we have neglected the line and column vectors involved in the $BPM(\mathbf{N})$ instance: these can be extended to $k \times k$ matrices themselves and handled by a simple adjustment to the above argument.

The inclusion $BPM(\mathbf{N}) \subseteq ALG^1(\mathbf{N})$ follows by the obvious divide and conquer strategy. The resulting $ALG^1(\mathbf{N})$ circuit is clearly uniform in an intuitive sense, being composed of identical matrix product layers, each layer being built from identical scalar product subcircuits, with each subcircuit essentially a binary tree. The uniformity in the technical sense follows by labelling the circuit nodes in such a way as to permit deducing from a label, in log time, the layer number, the subcircuit number, and the binary tree node number (encoded to easily provide immediate predecessor information) of the corresponding circuit node.

Equality of $\#NC^1$ and $ALG^1(\mathbf{N})$ is folklore, see e.g. [22, 36].

Finally, to show $\#NC^1 \subseteq \#L$, consider a $\#NC^1$ circuit. A nondeterministic logspace machine can perform a modified depth-first search of this circuit, adapting Borodin's $NC^1 \subseteq L$ proof [Bo77], as follows. The search starts at the output gate and proceeds towards the inputs, choosing a wire nondeterministically when it hits an OR gate, and proceeding normally when it hits an AND gate. The computation accepts when a proof that the output gate evaluates to one is found. An induction shows that the number of accepting paths in this procedure equals the number of proof trees in the original circuit. ■

Corollary 3.2 *Properly encoded, the following functions are #BP-complete under uniform projections:*

(a) *Given a sequence of constant dimension matrices over the natural numbers $\{0, 1\}$, compute a specific entry in the matrix product.*

(b) *Given a sequence of constant dimension matrices over natural numbers expressed in binary notation, compute a specific entry in the matrix product.*

Proof. By "proper", we mean that the matrix dimension k is a power of 2, and furthermore, in case (b), that we multiply η matrices whose entries are η-bit numbers, with η a power of 2. The need for these restrictions results from the very tight uniformity conditions implicit in the definition of the class *#BP*, as can be seen below.

We first argue that (a) uniformly projects to (b). Indeed, consider an instance of (a) having length n, involving matrices of constant dimension k, with k a power of 2. A log time machine can compute $\alpha = \lceil \log_2(n) \rceil$. The projection will consist of expanding each entry of each input matrix into a binary integer of length 2^α, and padding out the sequence of matrices with identity matrices, to length 2^α. Computing the output length $k^2 2^{2\alpha}$ of the projection can be done in *FDLOGTIME*, and similarly for the other two functions involved in defining a uniform projection.

Now the proof in Theorem 3.1 that $\#BP \subseteq BPM(\mathbf{N})$ easily extends to yield a uniform projection from an arbitrary function in *#BP* to the function (a); the only issue worth pointing out is the need to expand any occurrence of M_a (which was a constant symbol in Theorem 3.1) into the k^2 relevant bits. Hence the function (a) is *#BP*-hard under uniform projections.

We now show that the function (b) belongs to *#BP*. Indeed, given a string of length n, a log time machine can compute n. Because k is a power of 2, the machine can check that n is a multiple of k^2, and it can compute n/k^2. Because the purported number η of matrices equals the length of each matrix entry and because η is assumed to be a power of 2, the machine can further check that $n/k^2 = \eta^2$ for some $\eta = 2^\beta$, and it can compute β. The proof now mimics the proof in Theorem 3.1 that $BPM(\mathbf{N}) \subseteq \#BP$, with one new subtlety: To ensure that each layer in a graph G_M depends on a single bit of the representation of the matrix M, the construction of G_M requires more layers, i.e. approximately $k^2 \lfloor \log_2 m \rfloor$ layers, instead of the $\lfloor \log_2 m \rfloor$ layers used in Theorem 3.1. This proves that the function (b) belongs to $\#BP$. The function (a) belongs to $\#BP$ by the transitivity of uniform projections. ■

Theorem 3.3

$$GapBP = BPM(\mathbf{Z}) = ALG^1(\mathbf{Z}) = GapNC^1 \ .$$

Proof. The inclusion $GapNC^1 \subseteq ALG^1(\mathbf{Z})$ follows from $\#NC^1 \subseteq ALG^1(\mathbf{N})$ (Theorem 3.1). Now Ben-Or and Cleve [11] reduce evaluating an algebraic formula over a ring R to multiplying 3×3 matrices whose entries other than 0 and 1 are chosen from the inputs to the formula. When applied to a uniform logarithmic depth circuit family, their reduction yields as an intermediate step a projection to the matrix product problem. As argued by the authors of [11], this projection can be made *DLOGTIME*-uniform[2], given the close analogy with the reductions in [6, 7]. Thus $ALG^1(\mathbf{Z}) \subseteq BPM(\mathbf{Z})$ since the finite set of ring elements used as inputs to the algebraic circuit give rise to a finite set of possible matrices. Now to prove $BPM(\mathbf{Z}) \subseteq GapBP$, we expand each matrix entry $x \in \mathbf{Z}$, turning a $k \times k$ matrix into a $2k \times 2k$ matrix: x becomes $\begin{bmatrix} x & 0 \\ 0 & x \end{bmatrix}$ if $x \geq 0$, and x becomes $\begin{bmatrix} 0 & -x \\ -x & 0 \end{bmatrix}$ otherwise. In this way, we can set up an expanded vector-matrices-vector product over $\mathbf{N}$ to yield a vector $[a \ \ b] \in \mathbf{N}^2$ with the property that $a-b$ is the scalar result of the original vector-matrices-vector product over $\mathbf{Z}$. The rest of the argument follows as in the proof of $BPM(\mathbf{N}) \subseteq \#BP$. Finally, $GapBP \subseteq GapNC^1$ follows from $\#BP \subseteq \#NC^1$. ■

Corollary 3.4 *Properly encoded, the following functions are complete for $GapNC^1$ under uniform projections:*

(a) *Given a sequence of constant dimension matrices over the integers $\{-1, 0, 1\}$, compute a specific entry in the matrix product.*

(b) *Given a sequence of constant dimension matrices over integers expressed in binary notation, compute a specific entry in the matrix product.*

---

[2] *DLOGTIME* uniformity of programs in the sense of [6, 7, 9] differs from our definition of program uniformity in that the former requires *DLOGTIME* recognition of a description language for programs akin to the direct connection language for circuits. However, the two definitions are equivalent in all interesting situations, because, for example, although [6, 7, 9] do not formally require the ability to compute the length of a program in log time, any program of interest can be padded to length 2^l, where l is the length of the field of bits allocated to instruction numbers in the description language, and 2^l is computable in log time.

Proof. To prove that any function $h = f - g \in GapBP$ uniformly projects to the function (a), we simply arrange to compute $f \in \#BP$ and $g \in \#BP$ in two independent diagonal blocks of a matrix product over $\mathbf{N}$, and then we obtain $f - g$ by tacking on a final matrix with entries from $\{-1, 0, 1\}$. The details essentially repeat the hardness proof from Corollary 3.2. To prove that the function (b) is in $GapBP$, we build on the proof in Theorem 3.3 that $BPM(\mathbf{Z}) \subseteq GapBP$ and we appeal to the upper bound arguments of Corollary 3.2. ∎

Corollary 3.5

$$\begin{aligned} C_{=}BP = C_{=0}BPM(\mathbf{Z}) &= C_{=0}ALG^1(\mathbf{Z}) \\ &= C_{=}NC^1 \subseteq L. \\ C_{\leq}BP = C_{\leq 0}BPM(\mathbf{Z}) &= C_{\leq 0}ALG^1(\mathbf{Z}) \\ &= C_{\leq}NC^1 \subseteq L. \end{aligned}$$

Proof. All equalities are immediate consequences of Theorem 3.3. Now Lipton and Zalcstein show that a logspace machine can check $BPM(\mathbf{Z})$ calculations for equality with zero by carrying it out modulo enough small primes, proving $C_{=0}BPM(\mathbf{Z}) \subseteq L$ [27]. Ioan Macarie ([28], see also [15]) shows that two natural numbers represented by their residues modulo appropriately many small primes can be compared in log space, implying that determining if a given entry in a product of constant size matrices over the integers can be done in log space, which proves $C_{\leq}BP \subseteq L$. Manindra Agrawal, Eric Allender, and Samir Datta (personal communication) obtained another proof of the latter inclusion making use of results from [26]. ∎

A natural question now is of course about the relation between $C_{\leq}NC^1$ and $C_{=}NC^1$. We come back to this issue in the next section.

Corollary 3.5 motivates defining PNC^1 as $C_{\leq}NC^1$. Observe that PP is defined analogously in terms of $GapP$ functions in [17].

Corollary 3.6 *The following languages are $C_{=}NC^1$-complete (resp. PNC^1-complete) under uniform projections:*

(a) *Given a sequence of constant dimension matrices over the integers $\{-1, 0, 1\}$, determine whether a specific entry in the matrix product is zero (resp. nonnegative).*

(b) *Given a sequence of constant dimension matrices with integer entries expressed in binary notation, determine whether a specific entry in the matrix product is zero (resp. nonnegative).*

Theorem 3.1 and the definition of NC^1 imply that $C_{>0}ALG^1(\mathbf{N}) = NC^1$. Now we say that a circuit C is *unambiguous* if, for each input x, $\#accept(C,x) \in \{0,1\}$. For later use, we mention that the class NC^1 is unchanged if unambiguous circuits alone are used to define it, as can be seen using techniques from [25], or by simply observing that the NC^1 circuit evaluating a permutation branching program from [6] is unambiguous.

4 Closure properties

Clearly all the function classes examined in the previous sections are closed under addition and multiplication. In this section, we will generalize this and observe that $GapNC^1$ essentially has all closure properties $GapL$ is known to have, but due to the results on unambiguous circuits mentioned in the previous section we will see that some other interesting properties hold, that probably do not hold in the logspace context. Once we identify those properties it is straightforward to use known techniques to obtain structural results for the classes $C_=NC^1$ and PNC^1.

We start with some simple consequences of the obvious closure of $\#NC^1$ under addition and multiplication:

Lemma 4.1 *PNC^1 is closed under taking complements.*

Proof. Follows from

$$a \not\geq b \text{ iff } b \geq a+1$$

and closure of $\#NC^1$ under addition. ∎

Lemma 4.2 *$C_=NC^1$ is closed under union and intersection.*

Proof. Follows from $C_=NC^1 = C_{=0}GapNC^1$, $f(x) = 0$ or $g(x) = 0$ iff $f(x) \cdot g(x) = 0$, $f(x) = 0$ and $g(x) = 0$ iff $f(x)^2 + g(x)^2 = 0$, and closure of $GapNC^1$ under addition and multiplication. ∎

Lemma 4.3 *$FNC^1 \subseteq \#NC^1$.*

Proof. Let $f \in FNC^1$.

For every i, the language $A_i = \{x \mid \text{the } i\text{-th bit in } f(x) \text{ is } 1\}$ is in NC^1 and can thus be computed unambiguously. Moreover, for every i there is a uniform circuit C_i having exactly 2^i accepting proof trees (C_i is a complete binary tree of OR gates of depth i).

Now the circuit consisting of an OR (over all bit positions i of $f(x)$) of an AND of a circuit for A_i and C_i will have exactly $f(x)$ proof trees. The uniformity of this construction is immediate from the uniformity of circuits for A_i and the uniformity of the C_i. ■

For classes of functions $\mathcal{F}, \mathcal{G}$, let $\mathcal{F} - \mathcal{G} = \{f - g \mid f \in \mathcal{F}, g \in \mathcal{G}\}$.

Lemma 4.4 *$\#BP \subseteq FNC^1 - \#BP$.*

Proof. Let P be a uniform program over the automaton A. Manipulate A in the following way: Introduce new states and transitions such that the number of accepting paths does not increase for any input, but every state has the same number of outgoing edges.

Let now $f \in \#BP$ be computed by a program over an automaton normalized in the above way. Then $h(x) =$ the total number of paths (accepting or rejecting) on an input of length $|x|$ can be computed in FNC^1 (it is simply the out-degree of the automaton to the power of the length of the output of the projection). Defining $g(x)$ to be the number of accepting paths of the same program but swapping accepting and rejecting states in the automaton, we get $f(x) = h(x) - g(x)$, thus $f \in FNC^1 - \#BP$. ■

Corollary 4.5 *$GapNC^1 = \#NC^1 - FNC^1 = FNC^1 - \#NC^1$.*

This gives us the following normal forms for PNC^1 and $C_=NC^1$:

Corollary 4.6
1. *For every language A, $A \in PNC^1$ if and only if there exist $f \in \#NC^1$ and $g \in FNC^1$ such that for every x, $x \in A \leftrightarrow f(x) \geq g(x)$.*
2. *For every language A, $A \in C_=NC^1$ if and only if there exist $f \in \#NC^1$ and $g \in FNC^1$ such that for every x, $x \in A \rightarrow f(x) = g(x)$ and $x \notin A \rightarrow f(x) < g(x)$.*

Corollary 4.7 *$C_=NC^1 \subseteq PNC^1$.*

To establish more closure properties of PNC^1, we first address some nontrivial closure properties of $\#NC^1$ and $GapNC^1$. The following terminology is from [37]: Let $\mathcal{F}$ be a class of functions. We say that $\mathcal{F}$ is closed under *weak summation* if for every $f \in \mathcal{F}$ and every $c \in \mathbf{N}$ we have that the function h defined as $h(x) = \sum_{y=0}^{|x|^c} f(x, y)$ is also in $\mathcal{F}$. We say that $\mathcal{F}$ is closed under *weak product* if for every $f \in \mathcal{F}$ and every $c \in \mathbf{N}$ we have that the function h defined as $h(x) = \prod_{y=0}^{|x|^c} f(x, y)$ is also in $\mathcal{F}$. We say that $\mathcal{F}$ is closed under *taking binomial coefficients* if for every $f \in \mathcal{F}$ and every $c \in \mathbf{N}$ we have that the function h defined as $h(x) = \binom{f(x)}{c}$ is also in $\mathcal{F}$.

Lemma 4.8 *$\#NC^1$ is closed under weak summation and weak product.*

Proof. Given x, to compute $m = |x|^c$ is possible within NC^1. Our circuit will have subcircuits for every (x, y) for y in a range from 0 to the smallest power of 2 greater than or equal to m. This makes the range easily (in logtime) computable. To ensure that only relevant subcircuits contribute to the result, we will have subcircuits that unambiguously check $y \leq m$. To compute the polynomial sum or product can finally be achieved by a binary tree of logarithmic depth. The uniformity of the construction is immediate from the uniformity of the relevant FNC^1 computation, the uniformity of complete binary trees, and the uniformity of the circuits for the given function. ∎

Lemma 4.9 *$\#BP$ is closed under binomial coefficients.*

Proof. Let $f \in \#BP$ be witnessed by the NFA A. Then $\binom{f}{c} \in \#BP$ is witnessed by the NFA A' which works as follows: A' follows c paths of A simultaneously. To ensure that all paths are different, we use flags in A's state set. That is, the new set of states is $Q^c \times \{0, 1\}^{c-1}$, if Q is the set of states of A. To ensure that only one element out of all $c!$ permutations of c paths contributes in the end, we design A''s transition function in such a way that only ordered sequences occur. ∎

Corollary 4.10 *$GapNC^1$ is closed under subtraction, weak sum, weak product, and binomial coefficients.*

Proof. The only nontrivial point is closure under binomial coefficients, but this follows from the other closure properties by expressing binomial coefficients involving negative numbers as in [17] using differences of binomial coefficients with only nonnegative numbers. ∎

Corollary 4.11 *PNC^1 is closed under union and intersection.*

Proof. Follows from the established closure properties by reproducing the corresponding proof for PP from [10] or for PL from [2] essentially word for word. ∎

We remark that in addition to the above it can be shown e.g. that $\#BP$ is closed under weak summation and weak product, that $FNC^1 \subseteq \#BP$, and that $\#NC^1$ is closed under binomial coefficients. However, it was our main goal to show that $GapNC^1$ has all closure properties $GapL$ is known to have, and we only proved what is needed to obtain this result.

But some other interesting closure properties hold, that probably do not hold in the logspace context:

Theorem 4.12 *$\#BP$ and $\#NC^1$ are closed under signum and modified subtraction from 1, i.e., for every $f \in \#BP$ ($\#NC^1$), also $\mathrm{sgn}(f) = \min\{1, f\} \in \#BP$ ($\#NC^1$) and $1 \ominus f = \max\{0, 1-f\} \in \#BP$ ($\#NC^1$).*

Proof. The result for $\#NC^1$ follows from closure of NC^1 under complements and the fact that NC^1-circuits can be made unambiguous. The result for $\#BP$ follows from closure of NBP under complementation and the fact that finite automata can be made deterministic. ■

5 Leaf languages

In this section we generalize the class $C_{\mathcal{T}}BP$. For technical reasons it is convenient to replace the set F of final states in a nondeterministic automaton $A = (Q, \Delta, q_0, \delta, F)$ with a function from Q to a finite set Γ (hence the case $\Gamma = \{0, 1\}$ yields automata as we know them). We call the resulting six-tuple $A = (Q, \Delta, \Gamma, q_0, \delta, F : Q \to \Gamma)$ a *leaf automaton*.

Consider a program P, i.e. a projection $P = (\Sigma, \Delta, S, B, E)$, over the leaf automaton $A = (Q, \Delta, \Gamma, q_0, \delta, F)$. For each state $q \in Q$ and each word $w \in \Delta^*$ we define the ordered labeled tree $\mathit{tree}_q(w)$ by:

- $\mathit{tree}_q(\varepsilon)$, where ε is the empty word, is a leaf labeled with q,
- $\mathit{tree}_q(ub)$, where $b \in \Delta$, is the tree $\mathit{tree}_q(u)$ with all leaves labeled c replaced by the tree of depth 1 and width $\#\delta(c, b)$, with leaves labeled (in order) by
$$\delta(c, b, 1), \ldots, \delta(c, b, \#\delta(c, b)) \; .$$

The leaves of $\mathit{tree}_q(w)$ form a word over Q that is called $\mathsf{leaves}_q(w)$. The *computation tree* of P on input x is $\mathit{tree}_{q_0}(P(x))$. The string $\mathsf{leaves}(P, x)$ is the concatenation of all the $F(c)$ where c is a leaf of $\mathit{tree}_{q_0}(P(x))$. This string is called the *leaf string* of P on input x.

Let Y be a language over Γ. The language *recognized* by P with *leaf language* Y is:

$$\mathsf{Leaf}^P(Y) = \{x \in \Sigma^* \mid \mathsf{leaves}(P, x) \in Y\} \; .$$

The class $\mathsf{Leaf}^{NC^1}(Y)$ is the class of languages recognized by uniform polynomial length programs[3] with leaf language Y.

[3] In view of Theorem 3.1, Stephen Bloch appropriately suggested that a more suitable name for this class might be $\mathsf{Leaf}^{BP}(Y)$. However, we will stick to $\mathsf{Leaf}^{NC^1}(Y)$ because the latter name better reflects the fact that the leaf language class obtained is a refinement of the leaf language classes defined using log space or polynomial time machines.

5.1 Padding techniques

Leaf language classes have been studied in the context of polynomial time, logarithmic space, and logarithmic time computations [12, 19, 20, 21]. The numerous characterizations obtained in those papers make use of padding. Here we observe that in some sense the padding can be done by an automaton. This allows transferring in one blow many known characterizations to the $\mathsf{Leaf}^{NC^1}(\cdot)$ setting.

Consider the following automaton, in which we have allowed two distinct transitions on the same input symbol between a pair of states. This automaton is able to copy its input in padded form to the leaves. Its initial state is q_I (Z means "Zero" and O means "One") and its input alphabet is $\{0,1\}$.

	0	1
q_I	q_I, q_Z	q_I, q_O
q_O	q_O, q_O	q_O, q_O
q_Z	q_Z, q_Z	q_Z, q_Z

On input $x \in \{0,1\}^n$, the leaf string produced is $Ix_n x_{n-1}^2 x_{n-2}^4 \ldots x_1^{2^{n-1}}$. Building on this strategy, it is not hard to design a 9-state automaton having no duplicate transitions between states and producing the same leaf string, but on input $x_1x_1x_2x_2x_3x_3\ldots x_nx_n$. More generally, for a polynomial $p(n)$, let us denote by $\lceil p(n) \rceil$ a power of 2 upper bounding $p(n)$ and computable in log time from a length-n string. In proving Theorem 5.1, we will assume a uniform program Π having the property that $\mathsf{leaves}(\Pi, x_1 \cdots x_n) = Ix_1x_2^2x_3^4 \cdots x_n^{2^{n-1}} \$^{2^{\lceil p(n) \rceil} - 2^n}$, where $\$ \notin \{0,1\}$ (the strategy discussed above suggests an underlying 13-state automaton).

Let F be a set of finite functions (i.e., a set for which there is an $m \in \mathbf{N}$ such that every function is from $\{0,1,\ldots,m\}^r \to \{0,1\ldots,m\}$ for some r). Denote by $(F)LOGTIME$ ($(F)P$, resp.) the class of all sets decidable by nondeterministic Turing machines operating in logarithmic time (polynomial time, resp.) with F as locally definable acceptance type. (That is, essentially with every leaf node of a computation tree of such a machine a value from $\{0,1\ldots,m\}$ is associated and in every inner node a function from F is applied. Which value to take and which function to evaluate depends only on the state of the machine. For an exact definition, see [18, 20].) Then we can state the following:

Theorem 5.1 *$\mathsf{Leaf}^{NC^1}((F)LOGTIME) = (F)P$.*

Proof. The proof follows that of Theorem 3.1 in [21] and that of Theorem 3.1 in [20]: Given $A \in \mathsf{Leaf}^{NC^1}((F)LOGTIME)$, composing the projection with the $(F)LOGTIME$ decision procedure for the leaf language yields an $(F)P$ algorithm. For the converse, let $A \in (F)P$ via a nondeterministic machine working in polynomial time $p(n)$. For Π the uniform program discussed above, the length of $\mathsf{leaves}(\Pi, x_1 \cdots x_n) = Ix_1x_2^2x_3^4 \cdots x_n^{2^{n-1}}\$^{2^{\lceil p(n)\rceil}-2^n}$ is $2^{\lceil p(n)\rceil} \geq 2^{p(n)}$. Then the padded version of A, i.e. $\{\mathsf{leaves}(\Pi, x) : x \in A\}$, is in $(F)LOGTIME$, proving that $A \in \mathsf{Leaf}^{NC^1}((F)LOGTIME)$. (Note that the $(F)DLOGTIME$ machine finds its ith virtual input at position 2^i and that a padding symbol different from 0 or 1 was chosen in order to allow the machine to detect trying to read beyond its nth virtual input.) ∎

Let CH denote the counting hierarchy, i.e., the union of all classes of the oracle hierarchy obtained by using the class PP as base class (see [38]).

Corollary 5.2 *1.* $\mathsf{Leaf}^{NC^1}(\Sigma_k LOGTIME) = \Sigma_k^p$.

2. $\mathsf{Leaf}^{NC^1}(\Pi_k LOGTIME) = \Pi_k^p$.

3. $\mathsf{Leaf}^{NC^1}(AC^0) = PH$.

4. $\mathsf{Leaf}^{NC^1}(ACC^0) = MOD\text{-}PH$.

5. $\mathsf{Leaf}^{NC^1}(TC^0) = CH$.

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

Proof. All mentioned classes are locally definable as shown in [18, 20]. ∎

Using our leaf language padding techniques we can now observe the following:

Theorem 5.3 *1. If* $C \subseteq L \subseteq \mathsf{Leaf}^{NC^1}(C)$, *then* $C \neq \mathsf{Leaf}^{NC^1}(C)$.

2. If $C \subseteq P \subseteq \mathsf{Leaf}^{NC^1}(C)$, *then* $C \neq \mathsf{Leaf}^{NC^1}(C)$.

Proof. Suppose $C \subseteq P \subseteq \mathsf{Leaf}^{NC^1}(C) = D$. If follows from [19] (translating their padding into our context, just as we did above to obtain Corollary 5.2) that then $\mathsf{Leaf}^{NC^1}(D) \supseteq EXPTIME$. Thus the assumption that $C = \mathsf{Leaf}^{NC^1}(C)$ leads to the wrong conclusion that $P = EXPTIME$. The proof for the first claim is completely analogous. ∎

Corollary 5.4 *1.* $ACC^0 \neq MOD\text{-}PH$.

2. $TC^0 \neq CH$.

Proof. Immediate from Theorem 5.3 and known leaf language characterizations from [19, 21]. ■

Statement 1 of Corollary 5.4 improves a result by Allender and Gore [4] who showed that $ACC^0 \subset C_=P$.

We remark that it is straightforward to observe that the separations just given carry over from the case of polynomial circuit size to that of quasipolynomial circuit size, proving for example that $qACC^0 \neq$ *MOD-PH*.

5.2 Improved characterizations for the class *PSPACE*

In this section, we combine our padding techniques with a simple observation in order to prove that *PSPACE* can be characterized by leaf languages from *logspace-uniform* AC^0. Then we give a two-fold improvement to the surprising (but already somewhat technical) result, shown in [21], that $\mathsf{Leaf}^{\mathit{Logspace}}(\mathit{DCFL}) = \mathit{PSPACE}$: first, we can produce the leaf string with an automaton instead of a log space Turing machine, and second, the language at the leaves can be taken to be a *linear* deterministic language. (A context-free language is said to be *linear* if it is recognized by a pushdown automaton whose stack head changes direction only once.) A similar proof shows that the language at the leaves can be taken instead to be a *one-counter* deterministic language. (A context-free language is said to be *one-counter* if it is recognized by a deterministic pushdown automaton having only one stack symbol (plus the end-of-stack symbol).)

Theorem 5.5

$$\mathsf{Leaf}^{NC^1}(\mathit{logspace\text{-}uniform}\ AC^0) = \mathit{PSPACE}.$$

Proof. Let $Y \in \mathit{PSPACE}$. Using a uniform projection, it is possible to pad Y into Y' so that Y' can be recognized in linear space. Let a string y be mapped to y' by this projection. Now by applying the padding technique of Theorem 5.1 to Y', it is possible to find a language Y'' such that $Y = \mathsf{Leaf}^{NC^1}(Y'')$. Furthermore, if y'' is the leaf string resulting from y', then $y'' \in Y''$ iff $y' \in Y'$, and the bits of y' are found within y'' at positions 2^i, $1 \leq 2^i \leq |y''| = 2^{|y'|}$. Clearly then the language Y'' is in nonuniform AC^0 by a simple table-lookup procedure. In fact, because Y' is in linear space, the lookup table can be computed in space linear in $|y'|$, hence logarithmic in $|y''|$ (had we not padded Y out to Y', this would be polylogarithmic in $|y''|$). But then an appropriate circuit gate numbering makes it possible to construct a linear size AC^0 circuit for Y'' in logspace, fulfilling the requirements for the membership of Y'' in logspace-uniform AC^0. This concludes the proof that $PSPACE \subseteq \mathsf{Leaf}^{NC^1}(\text{logspace-uniform } AC^0)$; the reverse inclusion follows from the known fact that $\mathsf{Leaf}^{NC^1}(L) \subseteq PSPACE$ [21]. ■

Corollary 5.6 *1. If AC^0 is equal to logspace-uniform AC^0 then we have $PH = PSPACE$.*

2. If ACC^0 is equal to logspace-uniform ACC^0 then we have $MOD\text{-}PH = PSPACE$.

3. If TC^0 is equal to logspace-uniform TC^0 then we have $CH = PSPACE$.

What can be said about a converse to Corollary 5.6?[4] Using padding tricks as in [3] one can show that e.g. the collapse of logspace and dlogtime uniformity for AC^0 is equivalent to the identity of linear space and the alternating linear time hierarchy. Certainly, the latter equality immediately implies $PH = PSPACE$, but it is not known if the converse holds.

Theorem 5.7 *$\mathsf{Leaf}^{NC^1}(Y) = PSPACE$ for some deterministic one-counter context-free Y.*

Before we give the proof, we introduce some notation.

We write $\mathcal{S}_5$ for the permutation group on five points and we denote its identity by $()$. When w is a word over $\mathcal{S}_5$, we write $\mathcal{S}_5(w)$ for the $\mathcal{S}_5$ element resulting from multiplying out the elements in w.

The expression $\bar{\forall} X\, E(X)$ is shorthand for $\neg(\forall X E(X))$. We use the following version of the **QBF** problem. Given a quantified Boolean formula of the form $\bar{\forall} X_1\ \bar{\forall} X_2 \ldots \bar{\forall} X_n\ E(X_1,\ldots, X_n)$, is this formula true? The boolean expression $E(X_1,\ldots,X_n)$ is a formula in conjunctive normal form, that is, $E(X_1,\ldots, X_n) = C_1(X_1,\ldots, X_n) \wedge \ldots \wedge C_n(X_1,\ldots, X_n)$, where $C_i(X_1,\ldots, X_n)$ is the clause $L_{i,1} \vee \ldots \vee L_{i,n}$, the literal $L_{i,j}$ being equal to the variable X_j, the negation of the variable X_j, or the Boolean constant "False". The problem is coded as a string over the alphabet $\{X, \bar{X}, F, \wedge, m, \bullet, \$\}$. The clause C_i is coded by a word M_i such that $M_{i,j} = X$ if the variable X_j belongs to the clause, $M_{i,j} = \bar{X}$ if the negation of the variable X_j belongs to the clause, and $M_{i,j} = F$ otherwise. The character m will be used to enable counting, $\wedge$ and $\bullet$ are markers, whereas \$ is a padding symbol. For reasons

[4] In our original abstract (Proceedings of the 11th IEEE Conference on Computational Complexity 1996, pages 12–21) we mentioned that a converse could be proved using ideas from [3]. However, our claim was based on a wrong argument.

which will become clear, the expression $\bar{\forall} X_1 \ldots \bar{\forall} X_n E(X_1, \ldots, X_n)$ is coded by the word

$$M_1 \wedge \ldots \wedge M_n \bullet \underbrace{m^n \wedge \ldots \wedge m^n}_{n \text{ times}} \bullet \ .$$

We denote by $\mathbf{QBF}_?$ the set of such words and by $\mathbf{QBF}_V$ the set of words that code true expressions. The **QBF** language is the set of words in $\mathbf{QBF}_V$ possibly padded arbitrarily with the symbol \$.

Let K be a language in *PSPACE*. There exists a uniform projection $P_K = (\Sigma, \{X, \bar{X}, F, \wedge, m, \bullet, \$\}, S, B, E)$ from K to **QBF**. (Sketch: The acceptance condition of an alternating polynomial time Turing machine can be expressed as a deterministic polynomial time query quantified by a string of "$\neg\forall$" quantifiers. This query can be answered by a polynomial size circuit. The computation of this circuit can be expressed as a formula over the variables already quantified and over as many new existentially quantified variables as there are circuit nodes, for example by expressing the circuit gate $a = b \wedge c$ as $v_a \leftrightarrow (v_b \wedge v_c)$. This formula is finally transformed into conjunctive normal form, the existential quantifiers are transformed into "$\neg\forall$", and the reduction is completed by adding dummy variables and padding appropriately. The details of the construction of this projection, which are tedious but which require no new ideas besides those just sketched, can be found in [16].

Proof. (of Theorem 5.7) Consider the set of words

$$u_0 \#^{\alpha_0} u_1 \#^{\alpha_1} u_2 \#^{\alpha_2} \ldots u_{a-1} \#^{\alpha_{a-1}} u_a \#^{\alpha_a} \tag{1}$$

over the alphabet $\mathcal{S}_5 \cup \{\#\}$ where, for each i, $u_i \in \mathcal{S}_5$, $\alpha_i \geq 1$, and $\#^{\alpha_i}$ represents α_i consecutive occurrences of the symbol $\#$. We define the language L_{D1C} as the set of words of the form (1) which are accepted by the following procedure:

```
p := ()
i := 0
while i ≤ a do
        p := p * u_i
        i := i + α_i
if p = (12345) then accept
```

Clearly L_{D1C} is a deterministic one-counter context-free language. We will now prove that $PSPACE \subseteq \mathsf{Leaf}^{NC^1}(L_{D1C})$; the reverse inclusion is immediate from [21].

We will describe a leaf automaton $A = (Q, \{X, \bar{X}, F, \wedge, m, \bullet, \$\}, \mathcal{S}_5 \cup \{\#\}, q_0, \delta, F)$ such that $\mathsf{Leaf}^{P_K}(L_{D1C}) = K$, where P_K is the uniform program $(\Sigma, \{X, \bar{X}, F, \wedge, m, \bullet, \$\}, S, B, E)$ over A that projects K to $\mathbf{QBF}$. The automaton will never change its state when reading the symbol \$, so we assume for simplicity that $P_K(x) \in \mathbf{QBF}_?$. Now consider a *"Barrington word"* [6], defined over the group $\mathcal{S}_5$ and the variables X and Y as

$$\overset{1}{(125)}\overset{2}{X}\overset{3}{(235)}\overset{4}{Y}\overset{5}{(253)}\overset{6}{X}\overset{7}{X}\overset{8}{X}\overset{9}{X}\overset{10}{(235)}\overset{11}{Y}\overset{12}{Y}\overset{13}{Y}\overset{14}{Y}\overset{15}{(45)}\overset{16}{(23)} \quad .$$

This word was obtained using Barrington's construction [6] on the constant size $\{\neg, \wedge\}$-circuit for $\mathrm{NAND}(X, Y)$. This word has the property that it calculates $\mathrm{NAND}(X,\ Y)$ if the identity permutation () represents "false" and the permutation (12345) represents "true".

The leaf automaton A will nondeterministically perform 16-way branches at most steps of its computation. Its behavior along each branch will depend on its current state and, often, on the Barrington word. By the *access path* of a node in a computation tree of the automaton A, we will mean the word in $\{1, \ldots, 16\}^*$ identifying the sequence of choices leading A from the root of its computation tree to this node, with the proviso that deterministic branches are not recorded in the word representing this access path.

To simplify the exposition, it is convenient to define a function $\mathit{Barr} : \{1, \ldots, 16\} \to \mathcal{S}_5 \cup \{0, 1\}$ as follows. If $j \in \{1,\ 3,\ 5,\ 10,\ 15,\ 16\}$ then $\mathit{Barr}(j)$ is the permutation numbered j in the Barrington word, if $j \in \{2,\ 6,\ 7,\ 8,\ 9\}$ then $\mathit{Barr}(j) = 0$ and if $j \in \{4,\ 11,\ 12,\ 13,\ 14\}$ then $\mathit{Barr}(j) = 1$. Barr is extended to a function from $\{1, \ldots, 16\}^*$ to $(\mathcal{S}_5 \cup \{0, 1\})^*$ in the obvious way. In the sequel, when C is a clause with k literals and $\mathit{Barr}(w) \in \{0, 1\}^k$, we will write $C(\mathit{Barr}(w))$ for the Boolean value taken by C when the k variables involved in the k literals are respectively assigned the k values in the string $\mathit{Barr}(w)$ (if a literal l is "False", then $l(0) = l(1) =$ "False").

Again for ease of exposition, we partition the set Q of states of A into the "value 0" states, the "value 1" states, and the "constant c" states, for $c \in \mathcal{S}_5$. It is useful to note at the outset that A will never move from a "constant" state to a "value" state. Moreover, the string over $\mathcal{S}_5$ ultimately produced by A depends on the above partition of Q, in the following manner: the function F in the six-tuple defining A maps a "constant c" state to c, a "value 0" state to (), and a "value 1" state to (12345).

We now describe the automaton A in detail. We proceed by explaining the behavior of A on longer and longer prefixes π of the $\mathbf{QBF}$ instance $P_K(x)$ projected from an input x, and we will use F_π to represent the concatenation of all the $F(c)$ where c is a leaf of $tree_{q_0}(\pi)$.

$\underline{\pi = M_1\wedge}$:

- $\pi = M_{1,1}$. The jth choice, $1 \le j \le 16$, out of its initial state q_0 leads A to a state of type:
  - "constant $Barr(j)$" if $j \in \{1, 3, 5, 10, 15, 16\}$,
  - "value $L_{1,1}(Barr(j))$" otherwise.

  The product of F_π over $\mathcal{S}_5$ represents the truth value of the expression $\bar{\forall} X_1 L_{1,1}(X_1)$ (this expression is always true).
- $\pi = M_{1,1} \ldots M_{1,i}$. The jth choice, $1 \le j \le 16$, from a "value v" state, leads A to a state of type:
  - "constant Barr(j)" if $j \in \{1, 3, 5, 10, 15, 16\}$,
  - "value $v \vee L_{1,i}(Barr(j))$" otherwise.

  A node having access path $S \in \{1, \ldots, 16\}^i$ and labeled with a "value v" state is such that

  $$v = L_{1,1}(Barr(S_1)) \vee L_{1,2}(Barr(S_2)) \vee \ldots \vee L_{1,i}(Barr(S_i)).$$

  The jth choice, $1 \le j \le 16$, from a "constant c" state leads A to a state of type:
  - "constant c" if $j = 1$,
  - "constant ()" otherwise.

  The product of F_π over $\mathcal{S}_5$ represents the truth value of the expression $\bar{\forall} X_1 \ldots \bar{\forall} X_i \; L_{1,1}(X_1) \vee \ldots \vee L_{1,i}(X_i)$.

The transition of the automaton on the $\wedge$ symbol following M_1 (and in fact on any $\wedge$ symbol) is deterministic. The nodes at a given level in A's computation tree are numbered from left to right. After reading $M_1\wedge$, the current leaves are numbered from 0 to $16^n - 1$. We write $c_1(S)$ for the number of the node whose access path is $S \in \{1, \ldots, 16\}^n$.

After reading $\pi = M_1\wedge$, the product of F_π over $\mathcal{S}_5$ represents the truth value of the expression $\bar{\forall} X_1 \ldots \bar{\forall} X_n \; C_1(X_1, \ldots, X_n)$. If we had $C_1 \wedge C_2 \wedge \ldots \wedge C_n$ instead of C_1 alone in the latter expression, we would be done. However, in order to produce the values of $C_1 \wedge C_2 \wedge \ldots \wedge C_n$ at the appropriate leaves, the automaton would have needed to remember all its guesses; this is clearly not possible. Starting with the reading of M_2 described below, we will lose the property that the product over $\mathcal{S}_5$ of all available leaves represents the truth value of longer and longer prefixes of the coded formula. On the other

hand, throughout all subsequent stages, the property will remain true of precisely 16^n selected leaves, equally spaced from one another.
$\underline{\pi = M_1 \wedge \ldots \wedge M_i \wedge}$: The automaton behaves as in the above described case of $\pi = M_1 \wedge$ with the exception that, from every "constant c" state, A branches out to 16 "constant c" states (of course, not all descendants of these states will participate in the final product of $\mathcal{S}_5$ elements).

Note that any leaf in the tree obtained at this point has an access path of the form

$$S_1 S_2 \ldots S_i \in (\{1, \ldots, 16\}^n)^i.$$

Now, of all such nodes labeled with a "value v" state, only those having

$$S = S_1 = S_2 = \ldots = S_i, \tag{2}$$

for some $S \in \{1, \ldots, 16\}^n$, are relevant to the evaluation of the formula (the other such "value v" nodes evaluate the clauses C_j on truth values that differ from clause to clause). Now because all the stages past the reading of M_1 simply clone the "constant c" nodes, it is a pleasing fact that we can (and need to) also take as relevant all the "constant c" nodes having an access path of the form (2). The total set of relevant nodes therefore numbers precisely 16^n. Letting $c_i(S)$ be the number of the node having an access path of the form (2), we observe that $c_i(S) = 16^n c_{i-1}(S) + c_1(S)$ for $2 \le i \le n$. Moreover, an induction on i shows that the product over $\mathcal{S}_5$ of the leaves numbered $c_i(S)$ in $F_{M_1 \wedge \ldots \wedge M_i \wedge}$, for all $S \in \{1, \ldots, 16\}^n$, represents the truth value of the expression $\bar{\forall} X_1 \ldots \bar{\forall} X_n (C_1 \wedge \ldots \wedge C_i)$.
$\underline{\pi = M_1 \wedge \ldots \wedge M_n \bullet}$: A node of access path $S^n \in (\{1, \ldots, 16\}^n)^n$, labeled with a "value v" state, is such that $v = E(Barr(S))$.

The product over $\mathcal{S}_5$ of the leaves numbered $c_n(S)$ in $F_{M_1 \wedge \ldots \wedge M_n \bullet}$, $S \in \{1, \ldots, 16\}^n$, represents the truth value of the expression $\bar{\forall} X_1 \ldots \bar{\forall} X_n E(X)$. We have $c_n(S) = C \cdot c_1(S)$ where $C = (16^n)^{n-1} + (16^n)^{n-2} + \cdots + (16^n) + 1$. In other words, the relevant leaves are numbered $0, C, 2C, \ldots, (16^n - 1)C$.
$\underline{\pi = M_1 \wedge \ldots \wedge M_n \bullet m^n \wedge \ldots \wedge m^n \bullet}$: The rest of the coded formula is used to allow replacing each former state q leaf by a subtree $\tau(q)$ having C leaves. We leave it to the reader to solve the puzzle of how to use the second half of π to create exactly the right number of leaves (or see [16]). In $\tau(q)$, the leftmost leaf is a state q leaf, and all other leaves are state $q_\#$ leaves, for some new state $q_\#$ for which we set $F(q_\#)$ to $\#$.

In total, F applied to $\mathsf{leaves}(P_K, x)$ yields a word of the form (1), and membership of this word in L_{D1C} determines whether the product of the $\mathcal{S}_5$

leaves at positions $0, C, 2C, \ldots, (16^n - 1)C$ (prior to introducing the subtrees $\tau(q)$) yields (12345). ■

Theorem 5.8 *$\mathsf{Leaf}^{NC^1}(Y) = PSPACE$ for some deterministic linear context-free language Y.*

Proof. We define the language L_{DLIN} over the alphabet $\mathcal{S}_5 \cup \{\#\}$ as the set of words of the form

$$\begin{aligned}&\#^{\alpha_a} u_a \#^{\alpha_{a-1}} u_{a-1} \#^{\alpha_{a-2}} \ldots \\ &\quad u_2 \#^{\alpha_1} u_1 v_0 \#^{\beta_0} v_1 \#^{\beta_1} \ldots v_{b-1} \#^{\beta_{b-1}} v_b \#^{\beta_b}\end{aligned} \tag{3}$$

accepted by the procedure

$p := ()$
$i := j := 0$
while $j \leq b$ do
 $p := p * v_j$
 $i := i + \beta_j$
 if $(i > a)$ then reject
 $p := u_i * p$
 $j := j + \alpha_j$
if $p = (12345)$ then accept

where $u_i, v_j \in \mathcal{S}_5$, $\alpha_i \geq 1$, and $\beta_j \geq 1$. This language is a deterministic linear context-free language. We now explain the minor modification to the proof of Theorem 5.7 which shows that $PSPACE \subseteq \mathsf{Leaf}^{NC^1}(L_{DLIN})$.

As before, we construct the tree with relevant leaves at positions $0, C, 2C, \ldots (16^n - 1)C$. But now the automaton keeps track of the relevant leaf at position $(16^n/2)C$ (this is the "middle" relevant leaf and it corresponds to v_0 in the word (4) above). The automaton can do this, because $c_n(S) = (16^n/2)C$ implies $S = (91^{n-1}1)^n$, and the latter access path is easy to follow.

During the last stage of its computation, upon replacing each q leaf by a subtree $\tau(q)$ with C leaves, the automaton behaves differently according to the position of the q leaf being replaced relative to the middle relevant leaf. When the q leaf sits to the left of the middle, the leaves of $\tau(q)$ are arranged to be $q_\#^{C-1} q$ in that order. When the q leaf is the middle relevant leaf itself or when the q leaf sits to the right of the middle relevant leaf, the leaves of $\tau(q)$ are arranged to be $q q_\#^{C-1}$ in that order.

This strategy yields a word of the form (4) which belongs to L_{DLIN} iff the product over $\mathcal{S}_5$ formed by the 16^n relevant leaves yields (12345). ■

6 Conclusion

We have defined and studied counting classes at the level of NC^1. We have seen that these classes capture the complexity of multiplying constant dimension matrices over $\mathbf{N}$ and over $\mathbf{Z}$. Specifically, we have exhibited such matrix multiplication problems complete for $\#BP$, for $GapNC^1$, for $C_=NC^1$ and for PNC^1 under $DLOGTIME$-uniform projections.

One interesting class however which lacks complete problems so far is $\#NC^1$. Given that $\#NC^1$ is the set of functions computed by (uniform) log depth $\{+, \times\}$-circuits over $\mathbf{N}$, it is likely that a constrained form of a log depth $\{+, \times\}$-formula evaluation problem over $\mathbf{N}$ can be shown $\#NC^1$-complete. However, a more interesting question first raised by Martin Beaudry in the context of $\#NC^1$ is whether the general problem of evaluating a $\{+, \times\}$-formula over the natural numbers is $\#NC^1$-complete. Beyond a struggle with $DLOGTIME$-uniformity, this question again seems to bring to the fore the question of how efficiently an algebraic formula over a semiring can be balanced (see for instance [14]).

In Section 5 of this paper, we have refined leaf language characterizations of complexity classes. In particular, we now have delicate characterizations of $PSPACE$ in terms of restricted context-free languages and (projections over) automata. What can be made of these? How much further, if at all, can such characterizations be improved? From leaf language considerations, we have also deduced separations between complexity classes; these results have already been taken up and improved by Eric Allender (see [1]).

A model which we chose not to exploit in the present paper is the program-over-monoid model, instrumental to the known algebraic characterizations of NC^1 subclasses (see [6, 8, 29]). However, nondeterminism can be introduced into that model in natural ways. The first author has shown [16] that much of the padding discussed in the present paper can be performed using very restricted monoids. Interesting questions concerning the algebraic properties of the monoids simulating nondeterminism arise, and these may also provide an interesting avenue for future research.

Finally, we mention the intriguing question of whether $\#BP$ is equal to $\#NC^1$. Equality of these two classes unfortunately does not seem to follow immediately from the Ben-Or and Cleve simulation of an algebraic circuit over the naturals by an iterated matrix product, because the matrices involved contain negative numbers (with which we can postpone dealing, but which we cannot eliminate). But why should life be so complicated?

Hermann Jung [23] showed that arithmetic circuits of logarithmic depth can be simulated by boolean circuits of depth $\log n \cdot \log^* n$. Thus all that keeps us from declaring that $\#BP = \#NC^1 = FNC^1$ and that $C_=NC^1 = PNC^1 = NC^1$ is a $\log^* n$ factor for the depth plus uniformity considerations.

Acknowledgments. Thanks are due to Eric Allender, Martin Beaudry, Stephen Bloch, and Klaus-Jörn Lange, for helpful comments and for pointers to the literature. Klaus-Jörn Lange also brought to our attention that a claim which we made at the 1996 IEEE Complexity Theory conference (see Corollary 5.6 here) was overly optimistic. Thanks are also due to the referees for their careful reading of our manuscript, for pointing out that Macarie's work implies $PNC^1 \subseteq L$ (see Corollary 3.5), and for suggesting avenues for further research.

References

[1] E. Allender. A note on uniform circuit lower bounds for the counting hierarchy. *Proc. 2nd International Computing and Combinatorics Conference (COCOON)*, Springer Lecture Notes in Computer Science Vol. 1090, pages 127–135, 1996.

[2] E. Allender and M. Ogihara. Relationships among PL, $\#L$, and the determinant. *Proc. 9th Structure in Complexity Theory*, pages 267–278, 1994.

[3] E. Allender and V. Gore. On strong separations from AC^0. *Proc. 8th Fundamentals of Computation Theory*, Springer Lecture Notes in Computer Science Vol. 529, pages 1–15, 1991.

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

[5] C. Alvarez and B. Jenner. A very hard log-space counting class. *Theoretical Computer Science*, 107:3–30, 1993.

[6] D. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38:150–164, 1989.

[7] D. Barrington, N. Immerman and H. Straubing, On uniformity within NC^1, *Journal of Computer and System Sciences*, 41:3:274–306, 1990.

[8] D. Barrington and D. Thérien, Finite Monoids and the Fine Structure of NC^1, *Journal of the Association of Computing Machinery*, 35:4:941–952, 1988.

[9] F. Bédard, F. Lemieux and P. McKenzie, Extensions to Barrington's M-program model, *Theoret. Comp. Sci. A* 107:1:31-61, 1993.

[10] R. Beigel, N. Reingold, and D. Spielman. PP is closed under intersection. *Proc. 23rd Symposium on Theory of Computing*, pages 1–9, 1991.

[11] M. Ben-Or and R. Cleve. Computing algebraic formulas using a constant number of register. *SIAM Journal on Computing*, 21(1):54–58, 1992.

[Bo77] A. Borodin, On relating time and space to size and depth, *SIAM Journal on Computing*, 6:4:733-744, 1977.

[12] D. Bovet, P. Crescenzi, and R. Silvestri. A uniform approach to define complexity classes. *Theoretical Computer Science*, 104:263–283, 1992.

[13] H.-J. Burtschick. Comparing Counting Classes for Logspace, One-Way Logspace, and First-Order. *Proc. 20th Symposium on Mathematical Foundations of Computer Science*, Springer Lecture Notes in Computer Science Vol. 969, pages 139-148, 1995.

[14] S. Buss, S. Cook, A. Gupta and V. Ramachandran, An optimal parallel algorithm for formula evaluation, *SIAM Journal on Computing*, 21:4:755-780, 1992.

[15] P.-F. Dietz, I. Macarie, and J. Seiferas, Bits and relative order from residues, space efficiently, *Information Processing Letters*, 50:3:123–127, 1994.

[16] H. Caussinus. Contributions à l'étude du non-déterminisme restreint. Ph. D. thesis, University of Montreal, 1996.

[17] S. Fenner, L. Fortnow, and S. Kurtz. Gap-definable counting classes. *Journal of Computer and System Sciences*, 48:116–148, 1994.

[18] U. Hertrampf. Complexity classes defined via k-valued logic. *Proc. 9th Structure in Complexity Theory*, pages 224–234, 1994.

[19] U. Hertrampf, C. Lautemann, T. Schwentick, H. Vollmer, and K.W. Wagner. On the power of polynomial time bit-reductions. *Proc. 8th Structure in Complexity Theory*, pages 200-207, 1993.

[20] U. Hertrampf, H. Vollmer, K. Wagner. On balanced vs. unbalanced computation trees. *Mathematical Systems Theory*, in press.

[21] B. Jenner, P. McKenzie, and D. Thérien. Logspace and logtime leaf languages. *Proc. 9th Structure in Complexity Theory*, pages 242–254, 1994.

[22] J. Jiao. Some questions concerning circuit counting classes and other

low level complexity classes. Manuscript.

[23] H. Jung. Depth efficient transformations of arithmetic into boolean circuits. *Proc. Foundations of Computation Theory,* Springer Lecture Notes in Computer Scienc Vol. 199, pages 167–173, 1985.

[24] J. Köbler, U. Schöning, and J. Torán. On counting and approximation. *Acta Informatica*, 26:363–379, 1989.

[25] K.-J. Lange. Unambiguity of circuits, *Theoretical Computer Science*, 107:77–94, 1993.

[26] B. Litow. On iterated integer product. *Information Processing Letters*, 42:269–272, 1992.

[27] R. J. Lipton and Y. Zalcstein. Word problems solvable in logspace. *Journal of the Association for Computing Machinery*, 24(3):522–526, 1977.

[28] I. Macarie. Space-efficient deterministic simulation of probabilistic automata. *Proc. 11th Symposium on Theoretical Aspects of Computing (STACS)*, Springer Lecture Notes in Computer Science Vol. 775, pages 109–122, 1994

[29] P. McKenzie, P. Péladeau and D. Thérien, NC^1: The Automata-Theoretic Viewpoint, *Computational Complexity* 1, 330–359, 1991.

[30] W. Ruzzo, On uniform circuit complexity, *Journal of Computer and System Sciences,* 22:3:365-383, 1981.

[31] S. Saluja, K. V. Subrahmanyam, and M. N. Thakur. Descriptive complexity of $\#P$ functions. *Proc. 7th Structure in Complexity Theory*, pages 169–184, 1992.

[32] S. Tan, Calcul et vérification parallèles de problèmes d'algèbre linéaire, thèse de doctorat, Université de Paris-Sud, U.F.R. Scientifique d'Orsay, No d'ordre 3420, 1994.

[33] S. Toda. Classes of arithmetic circuits capturing the complexity of computing the determinant. IEICE Transactions on Communications/Electronics/Information and Systems, E75-D: 116–124, 1992.

[34] L. Valiant, Completeness classes in algebra, *Proc. 11th Symposium on Theory of Computing*, pages 249–261, 1979.

[35] L. Valiant. The complexity of computing the permanent. *Theor. Comp. Sci.* 8:189–201, 1979.

[36] V. Vinay. Counting auxiliary pushdown automata and semi-unbounded arithmetic circuits. *Proc. 6th Structure in Complexity Theory*, pages

270–284, 1991.

[37] H. Vollmer and K.W. Wagner. Recursion theoretic characterizations of complexity classes of counting functions. *Theoretical Computer Science*, 163:245–258, 1996.

[38] K.W. Wagner. Some observations on the connection between counting and recursion. *Theoretical Computer Science*, 47:131–147, 1986.