Electronic Colloquium on Computational Complexity, Report No. 8 (1999)

Arithmetic Complexity, Kleene Closure, and Formal Power Series

Eric Allender* V Arvind† Meena Mahajan‡

March 19, 1999

Abstract

The aim of this paper is to use formal power series techniques to study the structure of small arithmetic complexity classes such as GapNC^1 and GapL. More precisely, we apply the Kleene closure of languages and the formal power series operations of inversion and root extraction to these complexity classes. We define a counting version of Kleene closure and show that it is intimately related to inversion and root extraction within GapNC^1 and GapL. We prove that Kleene closure, inversion, and root extraction are all hard operations in the following sense: There is a language in AC^0 for which inversion and root extraction are GapL-complete, and there is a finite set for which inversion and root extraction are GapNC^1-complete, with respect to appropriate reducibilities.

The latter result raises the question of classifying finite languages so that their inverses fall within interesting subclasses of GapNC^1, such as GapAC^0. We initiate work in this direction by classifying the complexity of the Kleene closure of finite languages. We formulate the problem in terms of finite monoids and relate its complexity to the internal structure of the monoid.

1 Introduction

The interplay between formal language theory (also automata theory) and computational complexity has been a fruitful research theme for over a decade. Fundamental ideas from formal languages and automata theory have enriched complexity theory — in particular circuit complexity — and provided new insights. For instance, the fine structure of NC^1 [Bar89, BT88, MPT91] essentially follows from the algebraic study of subclasses of

*Department of Computer Science, Rutgers University, Hill Center, Busch Campus, P.O. Box 1179, Piscataway, NJ 08855-1179, USA. allender@cs.rutgers.edu Some of this work was performed while a visiting scholar at The Institute of Mathematical Sciences, Chennai, India. Supported in part by NSF grant CCR-9509603.

†The Institute of Mathematical Sciences, C.I.T. Campus, Chennai 600 113, India. arvind@imsc.ernet.in

‡The Institute of Mathematical Sciences, C.I.T. Campus, Chennai 600 113, India. meena@imsc.ernet.in

ISSN 1433-8092

regular languages [Pin86]. Another example concerns the class LogCFL (the logspace closure of context-free languages) which is an important subclass of P and has been studied intensively (both from the perspective of parallel computation and circuit complexity) [Coo85, Lan93, Ven91, BLM93].

Viewing languages as formal power series is an important unifying paradigm in formal language theory [SS78, Gin75, KS85, Sal90]. It has led to an arithmetization of the theory and to the unification of disparate-looking proofs in the area. Furthermore, the general approach has also yielded several new results.

With this success in mind, it is natural to try and apply formal power series techniques in the study of complexity classes. Since a language (or a function) can be viewed as a formal power series over an appropriate ring, a complexity class (of languages or functions) is a set of such formal power series. Li first studied the complexity classes FP, #P and GapP in the formal power series setting [Li92]. He identified the invertible elements in each of these complexity classes as the interesting objects (since the other elements are merely finite variations of these) and studies their various algebraic properties. It turns out in his study that the invertible elements in GapP form a group, and FP is a subgroup with interesting properties.

In this paper we are concerned with the study of *small* arithmetic complexity classes using formal power series. Our motivation for this study comes from circuit complexity. The circuit complexity classes we will refer to most often are AC^0 (problems having constant-depth, polynomial-size circuits of unbounded-fan-in AND and OR gates), ACC^0 (similar to AC^0, but now Mod_m gates are also allowed), TC^0 (as above, but now MAJORITY gates are allowed) and NC^1 (problems having logarithmic-depth circuits of fan-in two AND and OR gates). The well-known complexity classes NLOG and LogCFL also can be characterized in terms of classes of circuits [Ven91, Ven92]. Of particular interest to us here are the classes of functions that result from "arithmetizing" the classes AC^0, NC^1, NLOG, and LogCFL by replacing each AND (OR) gate by a multiplication (addition) gate over the natural numbers. The resulting classes of functions are denoted by $\#AC^0$, $\#NC^1$, #L, and #LogCFL, respectively. (#L also corresponds to counting the number of accepting paths of an NLOG machine; similar characterizations hold for the other arithmetic classes.) If we augment these arithmetic circuits by allowing the constant -1, then we obtain the classes of functions known as $GapAC^0$, $GapNC^1$, GapL, and GapLogCFL. These classes $GapAC^0$, $GapNC^1$, GapL, and GapLogCFL can also be characterized as the difference of two functions in $\#AC^0$ (or $\#NC^1$ or #L or #LogCFL, respectively) [FFK94, ABL98, CMTV96].

Why do we care about these arithmetic circuit classes? One reason is because they characterize natural and important problems. For instance, computing the determinant of integer matrices is complete for GapL (see [Tod91, Vin91, MV97]); and $GapNC^1$ was also shown to have natural complete problems in [CMTV96]. Another reason for interest in these classes was provided in [AAD97]: $\#AC^0$ and $GapAC^0$ were shown to characterize TC^0. (See [All] for a detailed survey on arithmetic circuit classes.) Since we have lower bounds for $GapAC^0$, but essentially no lower bounds for TC^0, it is hoped that perhaps formal power series techniques might offer a new avenue for proving lower bounds for threshold circuits, and this might also help us understand the internal structure of $GapNC^1$.

Following Li [Li92], the main formal power series operations we study are inversion and

root extraction. Li bounded the complexity of these operations by showing that they can essentially be done in polynomial time. These upper bounds are, unfortunately, not tight enough in the context of the arithmetic classes we consider.

Our first results pinpoint the complexity of inversion and root extraction. In Sections 4 and 5 we show that these operations are essentially *equivalent* to computing the determinant. More precisely, if g is a formal power series we show that g^{-1} and $g^{1/r}$ (for any integer $r > 1$) can be computed in GapL^g (in fact, if $g \in \mathrm{GapL}$ so is g^{-1}). Furthermore, we construct a formal power series in AC^0 for which inversion and root extraction are both hard for GapL. Thus, for any complexity class properly contained in GapL and containing AC^0, the invertible elements of that class *do not* form a subgroup of the invertible elements of GapL.

When we similarly examine the power of inversion and root extraction within GapNC^1, it turns out that, in fact, there is a *finite* language whose inverse (or r^{th} root for an integer $r > 1$) is a GapNC^1-complete function. On the other hand, inversion and root extraction for regular languages can be done in GapNC^1.

A motivation for this study was to see if formal power series could provide a tool in understanding the class GapAC^0. In contrast to Li's result [Li92] that FP is a subgroup of GapP, it turns out that GapAC^0 is *not* a subgroup of GapL. This is a consequence of our above result: Every subgroup of GapL that is closed downward under any reasonable reducibility (even projections) must actually contain GapNC^1, and we know that $\mathrm{GapAC}^0 \neq \mathrm{GapNC}^1$ [AAD97]. Furthermore, there is *no* proper subgroup of GapL that is closed downward under AC^0-reductions.

In the results of Section 5, the Kleene closure plays a central role. It turns out to be a useful bridge that links inversion and root extraction to the complexity of computing the integer determinant (and arithmetic branching programs). In fact, the hardness results in Section 5 are proved using a counting function associated with the Kleene closure (for a given language A this function $\#A^*$ counts the number of A-factorizations of a given word $w \in A^*$). Thus, another contribution of this paper is to make explicit the connection between Kleene closure and computing determinants.

The main interesting open question arising out of this paper is whether it is possible to classify finite languages A so that $\#A^*$ falls within subclasses of GapNC^1, such as GapAC^0, ACC^0, and AC^0. At first sight, this may not appear meaningful. However, in Section 6 we formalize one possible setting in which it makes sense to consider subclasses of finite languages and study the complexity of their Kleene closure.

2 Definitions

Let $\mathcal{R} = (R, +, \cdot)$ be a ring, and let Σ be a finite alphabet.

A formal power series is a mapping $g : \Sigma^* \to R$. Following the usual convention [KS85], we express g as if it were a series, $g = \sum_{w \in \Sigma^*} g(w)w$ although no summation is actually implied; a formal power series should merely be viewed as an infinite list, associating to each w the value $g(w)$. The set of all such formal power series is denoted by $R \ll \Sigma^* \gg$.

Let f and g be formal power series in $R \ll \Sigma^* \gg$. Then

$$f + g = \sum_{w \in \Sigma^*} (f(w) + g(w))w$$

$$f \cdot g = \sum_{w \in \Sigma^*} \sum_{w=xy} (f(x) \cdot g(y))w$$

where $f(w) + g(w)$ and $f(x) \cdot g(y)$ are the binary operations in the ring $\mathcal{R}$.

Let $\mathcal{C}$ be any complexity class of functions mapping Σ^* to R. A formal power series representation of a function $f \in \mathcal{C}$ is the series $\sum_{w \in \Sigma^*} f(w)w$. We denote by $R \ll \Sigma^* \gg(\mathcal{C})$ the set of all formal power series g such that $g \in \mathcal{C}$. (Let $\mathcal{L}$ be any complexity class of languages over Σ^*. Then we denote by $R \ll \Sigma^* \gg(\mathcal{L})$ the set of all formal power series g such that $\exists L \in \mathcal{L}$, $g = \chi_L$.)

We denote $R \ll \Sigma^* \gg(\text{GapL})$ by $\mathcal{GL}$.

In this paper, we consider the rings of integers or rationals only.

3 Some closure properties of GapL

It is useful to define the following connection between the determinant function and classes of functions $\mathcal{C}$: Det·$\mathcal{C}$ denotes the class of functions expressible as the determinant of a matrix whose entries are computable in $\mathcal{C}$. Formally,

Definition 3.1 *Let $\mathcal{C}$ be any class of functions from Σ^* to Z. The class* Det·$\mathcal{C}$ *is the class of functions $f : \Sigma^* \longrightarrow Z$ such that (1) there is a logspace many-one reduction h mapping each $w \in \Sigma^*$ to a matrix M_w over Σ^*, (2) there is a function $g \in \mathcal{C}$ that transforms M_w to the matrix N_w over Z by the application of g to each entry in M_w, and (3) the determinant of N_w equals $f(w)$.*

It is trivial to see that Det·$\mathcal{C} \subseteq \text{GapL}^{\mathcal{C}}$. The following results improve this bound.

Theorem 3.2 *Let B be an $n \times n$ matrix, whose entries are each polynomials of degree n in $Z[x]$, where the coefficients of each polynomial $B[i,j](x)$ are computable in* GapL. *Then the coefficients of the polynomial det(B) are computable in* GapL.

Proof.

To show that the determinant over integers can be computed in GapL, the determinant problem is reduced to counting paths in a DAG [Tod91, Vin91, MV97]. To generalize this to polynomials where the coefficients are non-negative integers given explicitly, we start with the DAG construction in the $\{0,1\}$ case and do an easy two-step modification as follows. The DAG we start with has all edges labeled with weight 1. In step 1, we split the contribution of each matrix entry (a polynomial) so that the contributions to different monomials are maintained in different nodes. The DAG width expands by a factor of $n^2 + 1$ (the number of monomials in the product polynomials). And each edge e is now labeled either with the constant 1 or with a coefficient from one of the polynomials $B[i,j](x)$. At this point, the sum (over all paths p from the source to the ith sink in the graph) of the product of all coefficients labelling edges on path p, is equal to the coefficient of x^i in the determinant polynomial.

In step 2, we replace the coefficients labelling the edges, with subgraphs, because the coefficients in each $B[i,j](x)$ are not supplied directly, but instead are computable in #L or GapL. If the coefficient is #L-computable, then we can replace the edge labeled $B[i,j](x)$

by the DAG in which the number of source-to-sink paths equals $B[i,j](x)$. However, if the coefficients are GapL-computable (and thus not necessarily non-negative), then each corresponding DAG has two sinks, and we must keep track of their contributions separately. This is achieved by expanding the DAG width by an additional factor of 2. The resulting DAG has n^2+1 positive sinks and n^2+1 negative sinks, and the coefficient of x^i in the determinant polynomial is given by the difference in the number of paths from the source to the ith positive sink and the number of paths from the source to the ith negative sink.

(An alternate proof is possible, using a theorem of Toda [Tod92] characterizing GapL in terms of "weakly skew" arithmetic circuits.) ∎

Corollary 3.3 Det·GapL = GapL. *i.e. Let B be an $n \times n$ matrix, whose entries are each computable in* GapL. *Then det(B) is computable in* GapL.

We note that to capture the hardness of GapL, determinants of matrices having a special structure suffice. The special structure we consider is that of extended lower triangular (elt) matrices, which arose in [Li92] in the context of inverting formal power series. Considering matrices of this restricted form is useful in proving the results of Sections 4 and 5. A matrix M is said to be *elt* if $A_{i,j} = 0$ whenever $j > i+1$.

Theorem 3.4 *Computing the determinant (or permanent) of elt matrices over the integers is complete for* GapL *w.r.t. logspace many-one reductions.*

Proof. We prove the theorem for the permanent of elt matrices. The result for determinant follows from the many-one equivalence of permanent and determinant for elt matrices (this equivalence is implicit in Lemma 4.7 of [Li92] where only the reduction from permanent to determinant is claimed).

A standard problem complete for GapL is, given a topologically-sorted directed acyclic graph G with $n+1$ nodes (with edges of the form $i \to j$ only if $i < j$, and with $n, n+1$, and 1 as distinguished nodes) compute as output $h_1(G)$, which is the difference of the number of paths from 1 to n and 1 to $n+1$ in G. Furthermore, we can assume that the graphs have the restricted form such that all paths from 1 to n are of length $2m$ and all paths from 1 to $n+1$ are of length $2m+1$, for some positive integer m. A related function that is #L-complete is $h_2(G)$, which counts the number of paths from 1 to n in a similarly restricted graph.

Given such a digraph G we associate the following elt matrix N_G with it:

$$\begin{bmatrix} b_{1,2} & 1 & 0 & 0 & \cdots & 0 \\ b_{1,3} & b_{2,3} & 1 & 0 & \cdots & 0 \\ b_{1,4} & b_{2,4} & b_{3,4} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & 0 \\ b_{1,n} & b_{2,n} & b_{3,n} & \cdots & b_{n-1,n} & -1 \\ b_{1,n+1} & b_{2,n+1} & b_{3,n+1} & \cdots & b_{n-1,n+1} & 1 \end{bmatrix}$$

Here, $b_{i,j}$ is equal to 1 if there is an edge from i to vertex j in G. Notice that the row indices are offset by 1 in the matrix, so that the diagonal entries are of the form $b_{i,i+1}$. (We have used here the fact that there are no edges entering 1 and no edges leaving n or $n+1$.)

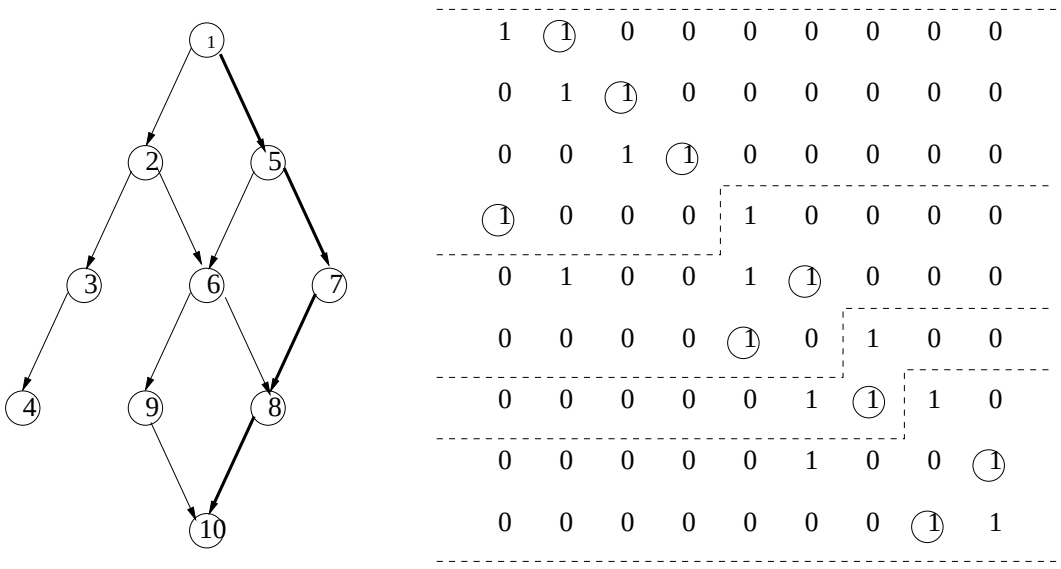

Figure 1: A path and the associated decomposition of an elt matrix.

To complete the proof, we claim that the permanent of N_G is the number of paths in G from 1 to n minus the number of paths in G from 1 to $n+1$. By definition, the permanent of N_G is given by:

$$perm(N_G) = \sum_{\pi} \prod_{i} N_G[i, \pi(i)]$$

There is a one-to-one correspondence between the nonzero terms $\prod_i N_G[i, \pi(i)]$ in the expression for $perm(N_G)$ (which are cycle covers corresponding to N_G) and paths in G from 1 to n or $n+1$. Furthermore, since $N_G[n-1, n] = -1$, observe that each term in $perm(N_G)$ corresponding to a path from 1 to $n+1$ contributes a -1, whereas each term corresponding to a path from 1 to n contributes a 1.

Notice that the permanent of the top-left $(n-1) \times (n-1)$ submatrix of N_G is $h_2(G)$, which shows that computing the permanent of nonnegative elt matrices is #L-complete.

See Figure 1 for an illustration for #L-hardness. The path $v_1 \longrightarrow v_5 \longrightarrow v_7 \longrightarrow v_8 \longrightarrow v_{10}$ marked in graph G corresponds to the cycle cover (1,2,3,4) (5,6) (7) (8,9) marked in N_G.

Finally, notice that the reduction can be easily implemented in logspace. ∎

4 Upper bounds for inversion and root extraction

In order to investigate the algebraic structure offered by formal power series, it is useful as a first step to consider the matter of multiplicative inverses. A formal power series f over the integers has a multiplicative inverse if and only if $f(\epsilon) \in \{1, -1\}$. To simplify the statement of the results that follow, we will consider for the moment only those f with $f(\epsilon) = 1$.

Lemma 4.1 (Lemmas 4.2, 4.6, 4.7 and Theorem 4.8 in [Li92]) *Let g be a formal power series with $g(\epsilon) = 1$. Then $g^{-1}(\epsilon) = 1$, and for $w \neq \epsilon$,*

$$g^{-1}(w) = \sum_{\substack{w = w_1 \cdots w_k \\ |w_i| \geq 1}} (-1)^k g(w_1) \cdots g(w_k)$$

That is, $g^{-1}(a_1 \cdots a_n)$ is given by the determinant of the following matrix:

$$\begin{bmatrix} -g(a_1) & -1 & 0 & \cdots & 0 \\ -g(a_1a_2) & -g(a_2) & -1 & \cdots & 0 \\ -g(a_1a_2a_3) & -g(a_2a_3) & -g(a_3) & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ -g(a_1 \cdots a_n) & -g(a_2 \cdots a_n) & -g(a_3 \cdots a_n) & \cdots & -g(a_n) \end{bmatrix}$$

Referring to Definition 3.1, this means that for any formal power series g, the inverse g^{-1} is computable in Det$\cdot\{g\}$. As an immediate consequence of this and Corollary 3.3, we have the following proposition.

Proposition 4.2 *The set of formal power series $\mathcal{GL}$ is closed under inverses.*

A closely related operation is computing powers or roots of a formal power series. It is easy to see that if $f \in \mathcal{GL}$, then f^r can also be computed in $\mathcal{GL}$. The converse is not so easy; note that there are $g = f^r \in \mathcal{GL}$ such that f does not even have integer coefficients. Such an f is not in $\mathcal{GL}$ at all. (For example, consider g given by $g(\epsilon) = g(a) = 1$, and $g(w) = 0$ otherwise.) We show some upper bounds on the complexity of computing f in such cases.

Lemma 4.3 (Lemma 5.5 in [Li92]) *Let f be a formal power series, and let $g = f^r$ for some positive integer r. Then, $\forall w \in \Sigma^*$,*

$$f(w) = \begin{cases} 1 & \text{if } w = \epsilon \\ \displaystyle\sum_{\substack{w = x_1 \cdots x_k; \\ x_i \neq \epsilon}} \sigma_r(k) g(x_1) g(x_2) \cdots g(x_k) & \text{if } w \neq \epsilon \end{cases}$$

where $\sigma_r(k)$ is defined as follows: $\sigma_r(0) = 1$, $\sigma_r(1) = 1/r$, and for $k > 1$,

$$\sigma_r(k) = (-1/r) \sum_{\substack{k = i_1 + \cdots + i_r \\ 0 \leq i_j < k}} \sigma_r(i_1) \cdots \sigma_r(i_r)$$

By rearranging terms notice that $\sigma_r(k)$, for $k \geq 2$, satisfies the following recurrence relation:

$$\sum_{\substack{k = i_1 + \cdots + i_r \\ 0 \leq i_j \leq k}} \sigma_r(i_1) \cdots \sigma_r(i_r) = 0$$

Using generating functions we can easily solve this recurrence to obtain an easy-to-compute closed-form expression.

Lemma 4.4 *For every $r, k > 0$, $\sigma_r(k) = \binom{1/r}{k}$.*

Putting this together, we have the following expression for $f = g^{1/r}$ when $w \neq \epsilon$:

$$f(w) = \sum_{\substack{w = x_1 \cdots x_k; \\ x_i \neq \epsilon}} \binom{1/r}{k} g(x_1)g(x_2)\cdots g(x_k)$$

It is interesting to note that when $r = -1$, the above expression coincides with the expression for inverse (Lemma 4.1).

The closed-form expression for $\sigma_r(k)$ derived above is convenient except in one respect: $\sigma_r(k)$ is a rational quantity. We will use a related integral quantity in our computations $N_r(0^n, k) = r^n n! \sigma_r(k)$, which is clearly computable in GapAC^0.

Lemma 4.5 *Let f be a formal power series with rational coefficients, and let r be a positive integer. Define the formal power series h as follows:*

$$\forall w \in \Sigma^*, h(w) = r^{|w|}(|w|)! f(w)$$

If $f^r \in \mathcal{GL}$, then so is h.

Proof.

Let $g = f^r$. Following the expression in Lemma 4.3, $f(w)$ can be written as the sum of n summands, where the kth summand collects the contribution from all decompositions of w into exactly k pieces. That is, for $n \geq 1$ and for $w = a_1 a_2 \cdots a_n \in \Sigma^*$, $f(w) = \sum_{k=1}^{n} \sigma_r(k) A_k(w)$, where

$$A_k(w) = \sum_{\substack{w = x_1 \cdots x_k; \\ x_i \neq \epsilon}} g(x_1)g(x_2)\cdots g(x_k)$$

And so h can be rewritten as $h(w) = \sum_{k=1}^{n} N_r(0^n, k) A_k(w)$.

Each $N_r(0^n, k)$ is computable in GapL. We show below that each $A_k(w)$ is also computable in GapL. Because GapL is closed under addition and multiplication, it follows that $h(w)$ is computable in GapL.

Consider the following matrix, where x is an indeterminate:

$$B_x = \begin{bmatrix} g(a_1) & -x & 0 & \cdots & 0 \\ g(a_1a_2) & g(a_2) & -x & \cdots & 0 \\ g(a_1a_2a_3) & g(a_2a_3) & g(a_3) & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ g(a_1\cdots a_n) & g(a_2\cdots a_n) & g(a_3\cdots a_n) & \cdots & g(a_n) \end{bmatrix}$$

The determinant of this matrix, $\mathrm{Det}(B_x)$, is a polynomial of degree $n-1$ in x, and it is easy to see that the coefficient of x^{n-k} in $\mathrm{Det}(B_x)$ is precisely $A_k(w)$. From Theorem 3.2, these coefficients are computable in GapL. ∎

Theorem 4.6 *Let f be any formal power series with integer coefficients. If, for some positive integer r, $g = f^r \in \mathcal{GL}$, then the coefficients of f can be computed in* $\mathrm{FL}^{\#\mathrm{L}}$.

Proof. We outline an algorithm for computing $f = g^{1/r}$:

Let h be the formal power series defined in Lemma 4.5. Define $H(.)$ to be the multiplicative constant here; $H(k) = r^k k!$. Both $h(w)$ and $H(|w|)$ are computable in GapL. (In fact, $H(|w|)$ is computable in GapAC^0.) To compute $f(w)$, we need to divide $h(w)$ by $H(|w|)$ in a uniform way. It follows from the results of Beame, Cook and Hoover [BCH86] that there is a uniform NC^1 circuit that takes as input 3 n-bit numbers M, x, y, and if M is the product of the first n^2 primes, then the circuit outputs the n-bit number $\lfloor \frac{x}{y} \rfloor$.

Since h and H are GapL functions, there is a polynomial p such that both $h(w)$ and $H(|w|)$ represented in binary are at most $p(|w|)$ bits long. Let $Q(k)$ denote the product of the first k^2 primes. For uniform division, we require the product $Q(p(|w|))$, which is easily seen to be in GapL. To compute $f(w)$, it suffices to evaluate the NC^1 circuit for division mentioned above on inputs $Q(p(|w|)), h(w), H(|w|)$. A uniform NC^1 circuit can be evaluated in DLOG. Consider such an evaluation procedure on the division circuit described above. Whenever an input bit is required, the DLOG machine finds the appropriate bit (from h or H or Q) by querying a GapL oracle. ∎

At this point it is natural to ask if there is any complexity class smaller than GapL with the property given in Proposition 4.2, or with properties analogous to those in Lemma 4.5 or Theorem 4.6. The next section essentially gives a negative answer to these questions.

5 Lower bounds for inversion and root extraction

In order to formalize the hardness of inverse and root extraction in a uniform fashion, we first define a new counting function on languages.

Definition 5.1 *Let $A \subseteq \Sigma^*$. We define $\#A^*$ to be the following function:*

$$\#A^*(w) = \sum_{\substack{w = w_1 \cdots w_k \\ |w_i| \geq 1}} \chi_A(w_1) \cdots \chi_A(w_k)$$

Given a language A in a specific complexity class, what is the complexity of the decision problem A^* and the function $\#A^*$? We first show hardness results for $\#A^*$ and then derive corresponding hardness results concerning computing the inverse or extracting the root of a formal power series.

Theorem 5.2 *There is a language $A \in \mathrm{AC}^0$ such that*

(a) $\#A^*$ is hard for #L.

(b) computing g^{-1}, where g is the formal power series χ_A, is complete for GapL *under logspace many-one reductions (and in fact under* AC^0 *projections).*

(c) (for all $r > 1$) computing the rth root of the formal power series χ_A is hard for #L *under (P-uniform)* NC^1 *reductions.*

One proof of this theorem follows from results of Flajolet and Steyaert [FS74]. In [FS74], a language $A \in$ LOG is constructed, and its Kleene closure is shown to be complete for NLOG by reducing a language B to it; B was known to be NLOG-complete from [Sud73]. It is easy to verify that A is actually in AC^0, and that the chain of reductions is "parsimonious" (i.e., that in the reduction f from a problem $B \in$ NLOG to A^*, where B is accepted by an NLOG machine M, the number of accepting paths of $M(x)$ is the same as the number of decompositions of $f(x)$ showing that $f(x)$ is in A^*) and is a projection. In order to keep the paper self-contained we include a different proof, which is more in line with the other proofs in this paper.

Proof.

(a) We use the construction in the proof of Theorem 3.4. Let h_2 be the #L-complete problem considered there. It suffices to show that there is a language $A \in AC^0$, with the property that there is an easy reduction from h_2 to the problem of computing $\#A^*$. More precisely, we will present a reduction r such that for any x which is the adjacency matrix of a graph G of the restricted type described in the proof of Theorem 3.4, $h_2(x) = \#A^*(r(x))$.

The reduction r we build will take a graph G as input, and output a particular string $\mathrm{enc}(N_G)$ that encodes N_G.[1] This string is a list, in row major form, of the encodings of each position (i, j) in N_G. We will encode each position (i, j) by a string $\mathrm{enc}(i, j)$ rather than merely the bit at that position. This encoding consists of the triple $(i; j; b)$; indicating that the (i, j) entry of the matrix N_G is b. We will encode this in binary, and thus we need encodings of the symbols $\{), (, ;, 0, 1\}$. To be specific, let us encode the binary bits 0 and 1 by 110 and 111, respectively, and we will let 000 and 001 encode the left and right parentheses, respectively, and 011 will encode the semicolon. All indices i and j will be represented using $t = \lceil \log n \rceil$ bits (with leading zeros), and each of these t bits will be encoded as above. So for each (i, j), $\mathrm{enc}(i, j)$ is a string of length $6t + 18$. And $\mathrm{enc}(N_G)$ is the string $\mathrm{enc}(1,1)\mathrm{enc}(1,2)\ldots\mathrm{enc}(1,n)\mathrm{enc}(2,1)\ldots\mathrm{enc}(n.n)$. Let us call this string $r(G)$. It is clear that r is computable in uniform AC^0 (and in fact r is a projection).

The language A in AC^0 is the set of all strings w such that w is a substring of $\mathrm{enc}(N)$ for some matrix N, and w is of one of the following forms:

- w is of the form

$$\begin{array}{ccccccc} enc(1,1) & \cdots & enc(1,j) & \cdots & enc(1,n) \\ enc(2,1) & \cdots & enc(2,j) & \cdots & enc(2,n) \\ \vdots & & \vdots & \vdots & \vdots \\ enc(j,1) & \cdots & enc(j,j) \end{array}$$

  where $N(j, 1) = 1$.

- w is of the form

$$\begin{array}{ccccccc} & & enc(i,i+1) & \cdots & enc(i,j) & \cdots & enc(i,n) \\ enc(i+1,1) & \cdots & enc(i+1,i+1) & \cdots & enc(i+1,j) & \cdots & enc(i+1,n) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ enc(j,1) & \cdots & enc(j,i+1) & \cdots & enc(j,j) \end{array}$$

[1] Note that the reduction need not check if G has the restricted form.

where in addition $N(j, i+1) = 1$.

That is, if w is a substring of an $\mathrm{enc}(N_G)$ for a graph G satisfying the restrictions described earlier, then w is in A if it is a piece of $\mathrm{enc}(N_G)$ that corresponds to one cycle in a cycle cover (as in Theorem 3.4). The cycle in question is $(1, 2, \ldots, j)$ in case 1 and $(i+1, i+2, \ldots, j)$ in case 2. It is easy to see that A is in AC^0. See Figure 1 for an illustration.

Now recall that

$$\#A^*(w) = \sum_{\substack{w = w_1 \cdots w_k \\ |w_i| \geq 1}} \chi_A(w_1) \cdots \chi_A(w_k)$$

For any string w of the form $r(G)$, a decomposition of the form $w = w_1 \cdots w_k$ with $w_i \in A$ consists of a decomposition of N_G corresponding to a cycle cover having k cycles. It follows that if w is of the form $r(G)$, then $\#A^*(w)$ counts the number of non-zero cycle covers in N_G; $\#A^*(w)$ equals $\mathrm{perm}(N_G)$. From Theorem 3.4, this is equal to $h_2(G)$.

(b) That g^{-1} can be computed in GapL follows from Proposition 4.2. We first show hardness for #L. In the construction described in part (a), note that all non-zero cycle covers in N_G have an equal number of cycles, in fact precisely $2m$. (This was an assumption made, w.l.o.g., in the proof of Theorem 3.4.) So for any string w of the form $r(G)$, the following also holds:

$$\#A^*(w) = \sum_{\substack{w = w_1 \cdots w_k \\ |w_i| \geq 1}} (-1)^k \chi_A(w_1) \cdots \chi_A(w_k)$$

So if g is the formal power series χ_A, then $g^{-1}(w) = \#A^*(w)$ for w of the form $r(G)$. #L hardness now follows from part (a). Using h_1 instead of h_2 in the construction of part (a) and arguing as above shows that the corresponding g^{-1} is GapL-complete.

(c) Let f be a formal power series, and let $g = f^r$ for some positive integer r. Letting $g = \chi_A$ for the language A of part (a) in the expression for root extraction derived in Lemmas 4.3 and 4.4, we get

$$g(w) = \binom{1/r}{2m} \#A^*(w)$$

To complete the reduction we need integer division and integer multiplication to cancel $\binom{1/r}{2m}$. This can be done as in the proof of Theorem 4.6. ∎

We observe without proof that Theorem 5.2 generalizes to levels of the #L hierarchy. This hierarchy was defined in [AO96] as follows: define $\#\mathrm{LH}(1)$ to be #L, and let $\#\mathrm{LH}(i+1)$ be the class of functions f such that, for some logspace-bounded nondeterministic oracle Turing machine M, and some function $g \in \#\mathrm{LH}(i)$, $f(x)$ is the number of accepting computations of M on input x with oracle g.

Theorem 5.3 *For each i, there is a language A in $\mathrm{L}^{\#\mathrm{LH}(i)}$ such that $\#A^*$, and consequently, the inversion and root extraction of χ_A are hard for $\#\mathrm{LH}(i+1)$.*

Theorem 5.2 effectively puts an end to any plans to investigate the group-theoretic structure formal power series corresponding to complexity classes smaller than GapL. Still, given

the close relationship between subclasses of NC^1 and classes of regular sets ([MPT91]), might it be possible to say something useful about these classes by investigating formal power series corresponding to regular sets? The characteristic function of any regular set is given by a "rational" formal power series over the integers. (See, e.g., [SS78, BR84].) Thus the following result indicates that by considering only regular sets, it might be possible to study at least the class GapNC^1.

Theorem 5.4 *Let g be an invertible rational formal power series over the integers. Then g, $g^{-1} \in R \ll \Sigma^* \gg(\mathrm{GapNC}^1)$.*

Proof. The inverse of any invertible rational formal power series is also rational. A fundamental theorem about rational series is that the rational formal power series coincide with the "recognizable" formal power series. The definition of "recognizable" formal power series makes it immediate that computing the coefficient of a word w in a series can be performed by multiplying together $O(|w|)$ k-by-k integer matrices, where k depends on the series, but does not depend on w. (For these and other basic facts about formal power series, please consult a text such as [SS78, BR84].)

It is observed in [CMTV96] that all functions that can be reduced to iterated multiplication of $O(1)$-by-$O(1)$ matrices are in GapNC^1. This completes the proof. ∎

Remark 1 *For the reader who wishes to see a self-contained proof of inversion of regular sets, a construction based directly on finite-state automata and branching programs is described in the appendix.*

Our main motivation in the use of formal power series within GapNC^1 was to see if it could aid in our understanding of the class GapAC^0, which in turn provides an alternative characterization of threshold circuit classes [AAD97]. Since the formal power series of regular languages are in GapNC^1 (as indicated by Theorem 5.4), the question of interest is whether subclasses of this class of formal power series give new characterizations of GapAC^0 and other function classes inside GapNC^1.

The next theorem, somewhat surprisingly, shows that there are *finite* languages such that the inverses of their formal power series are GapNC^1-complete. The decision version of this result is already known: namely, there is a finite language A such that A^* is NC^1-complete. (Every regular language whose syntactic monoid is unsolvable is NC^1-complete [Bar89], and Schutzenberger [Sch65] and Margolis (see Chapter 5.3 of [Pin86]) exhibit a finite prefix code A such that the syntactic monoid of A^* is unsolvable, thus showing that A^* is NC^1-complete.) However, the above constructions from [Sch65, Pin86] cannot be adapted to show hardness for counting Kleene closure, since languages which are prefix codes have unique decompositions in their Kleene closure. In contrast, the proof we describe below constructs a finite language which is not a prefix code, and the proof does adapt to the decision version as well.

Theorem 5.5 *There is a finite language A such that*

(a) computing the inverse of the formal power series χ_A is hard for GapNC^1 *under* AC^0 *reductions.*

(b) $\#A^*$ *is complete for* GapNC^1 *under* AC^0 *reductions.*

(c) (for all $r > 1$*) computing the* r*th root of the formal power series* χ_A *is hard for* GapNC^1 *under (nonuniform)* NC^1 *reductions.*

Proof. (a) As is pointed out in [CMTV96][Theorem 3.2], it follows from the construction of [BOC92] that every function in GapNC^1 can be reduced to the problem of finding the (1,1) entry of the result of multiplying together a sequence of 3-by-3 integer matrices, and this problem is in turn reducible to taking the difference of two functions in width-six #BP.

We show that there is a finite A such that computing the inverse of the formal power series χ_A is hard for width-six #BP under AC^0 projections. This suffices to prove the theorem.

The language A we construct will have alphabet Σ equal to the set of all relations on $\{1,2,3,4,5,6\}^2$. Each element of Σ can be viewed as a bipartite graph on $\{1,2,3,4,5,6\}^2$, with two vertical columns of six vertices each, and edges going from the left column to the right column. Thus a word in Σ^* represents a width-six graph in a straightforward way. We are now ready to define the set A. A is the set of all words over Σ^*, such that there exists a path of length ≤ 21 from 1 in the leftmost column to 1 in the rightmost column, such that this path does not visit 1 at any intermediate column. (To simplify later discussion, call the node 1 in the leftmost column s, and call the node 1 in the rightmost column t.)

Clearly A is finite (since it contains no word of length greater than 21).

A hard problem for width-six #BP is the problem of counting the number of paths from s to t in a word over Σ^*. Given any such word w, we will show how to construct a word $r(w)$, such that the number of such paths in w is equal to the coefficient of $r(w)$ in the formal power series χ_A^{-1}.

The first step in computing $r(w)$ is to insert six copies of the relation $\{(i, i{+}1(\mathrm{mod}\ 6))|1 \leq i \leq 6\}$ in between each two consecutive symbols of w. Note that inserting these six symbols has no effect on the number of paths between any two vertices, since together they are equivalent to the identity relation. Call this new word w'.

The next step is to insert the identity relation $\{(i,i)|1 \leq i \leq 6\}$ between each two symbols of w'. The result is the desired word $r(w)$. Again, note that there is a one-to-one correspondence between the paths in w and the paths in $r(w)$.

To prove hardness, note first of all that every path from s to t in $r(w)$ can be decomposed into paths of length at most 21 from 1 in one column to 1 in a later column. (Note that this corresponds to a decomposition into words in A.) To see this, consider an edge from i to j in some column in the original word w. In $r(w)$, this edge is replaced by the sequence

$$i \to j \to j \to j+1(\mathrm{mod}6) \to j+1(\mathrm{mod}6) \to j+2(\mathrm{mod}6) \to j+2(\mathrm{mod}6) \to j+3(\mathrm{mod}6)$$

$$\to j+3(\mathrm{mod}6) \to j+4(\mathrm{mod}6) \to j+4(\mathrm{mod}6) \to j+5(\mathrm{mod}6) \to j+5(\mathrm{mod}6) \to j \to j$$

Note that node 1 is visited in at least one (actually, two) of the columns above.

If we now consider the sequence of length twenty-eight in $r(w)$ that replaces the two-edge sequence $i \to j \to k$ in w, it is easy to see that the longest possible sequence of edges avoiding 1 in any intermediate position occurs if $j = 6$ and $k = 2$, in which case 1 is reached for the first time after a path of length 21.

Next, note that any two distinct s-t paths in w give rise to distinct decompositions of $r(w)$ into words in A. To see this, consider the first two edges (i,j) and (i,j') where two paths differ. The cyclic shift from j in $r(w)$ will reach 1 at a different time than the cyclic shift from j'. Thus we obtain two distinct decompositions.

To complete the proof, recall that the formal power series for χ_A^{-1} on an input $r(w)$ is given by

$$\sum_{\substack{r(w) = w_1 \cdots w_k \\ |w_i| \geq 1}} (-1)^k \chi_A(w_1) \cdots \chi_A(w_k)$$

Since every second symbol of $r(w)$ is the identity relation (and the identity relation is a proper prefix of no other word in A), every decomposition $r(w) = w_1 \cdots w_k$ has each $|w_{2i}| = 1$, and each w_{2i} is the identity relation. Thus k is even, for each such decomposition, and thus the coefficient of $r(w)$ in χ_A^{-1} is equal to the number of s-t paths in w.

(b), (c) Since in the width-6 BP's considered in the above proof we can assume that all $s-t$ paths are of the same length $2m$, we can complete the proof in a manner similar to the proof of Theorem 5.2. ∎

The proofs in this section indicate the extent to which taking the inverse of a formal power series is similar to computing the transitive closure, or of taking the Kleene closure of a language. Let us close this section with an observation in this same spirit, which follows immediately from part (a) of Theorem 5.2. This may be seen as a slight generalization of a result in [Mon75], where a result of this form is proved for $\mathcal{C}$= DLOG.

Theorem 5.6 *Let $\mathcal{C}$ be any class of languages closed under* AC^0 *reducibility and contained in* NLOG. *Then $\mathcal{C}$ is closed under the Kleene * operation if and only if* $\mathrm{NLOG} = \mathcal{C}$.

6 The Kleene closure of finite languages

In light of Theorem 5.5, if there is to be any hope of clarifying the relative computational power of GapAC^0, TC^0, and GapNC^1 by considering the notions of inverses and Kleene closure, it can only come by considering restrictions on finite sets. In this section we attempt such a classification for the complexity of A^* based on the structure of the finite language A. Our results closely follow the connection between the structure of finite monoids and the internal structure of NC^1 [Bar89, BT88]. We leave open the possibility of a similar study for the complexity of $\#A^*$.

Definition 6.1 *Let $(A, \circ)$ be a finite monoid.*[2] *There is a natural homomorphism $v : A^* \mapsto A$ that maps a word w to its valuation $v(w)$ in the monoid A. Let F be a group contained in A and r be a positive integer. The language $A_{F,r} \subseteq A^*$ is defined as $A_{F,r} = \{w \in A^* \mid |w| \leq r, v(w) \in F\}$.*

We define the $(A_{F,r})^*$ *closure problem* as the decision problem $(A_{F,r})^*$. Firstly, since $(A_{F,r})^*$ is a regular language, notice that the $(A_{F,r})^*$ closure problem is always in NC^1.

[2]For ease of notation we denote the monoid product $a \circ b$ simply as ab and we also use simply A to denote the monoid $(A, \circ)$ in this section.

To give the intuition behind the formal definition we restate the proof idea of Theorem 5.5, adapted to the decision version, in terms of an $(A_{F,r})^*$ closure problem.

Corollary 6.2 *Let A be the permutation group S_6, let $F = \{\pi \in S_6 \mid \pi(1) = 1\}$, and let $r = 21$. Then the $(A_{F,r})^*$ closure problem is* NC^1*-complete.*

The following result generalizes the above corollary to any nonsolvable monoid and shows connections between the internal structure of the underlying finite monoid and the complexity of corresponding $(A_{F,r})^*$ closure problems.

Theorem 6.3 *(a) Let A be any nonsolvable monoid. Then there exists a group $F \subseteq A$ and a constant $r > 0$ such that the $(A_{F,r})^*$ closure problem is* NC^1*-complete.*

(b) The $(A_{F,r})^*$ closure problem is reducible via AC^0 *projections to the word problem over the finite monoid A.*

(c) If A is a solvable monoid then the $(A_{F,r})^*$ closure problem is in ACC^0*. Furthermore, if A is an aperiodic monoid then the $(A_{F,r})^*$ closure problem is in* AC^0*.*

Proof.
(a) Since A is a nonsolvable monoid, A contains a nontrivial nonsolvable group G with identity e'.[3] Since the word problem over G is NC^1-complete [Bar89], it suffices to show an AC^0 reduction from the word problem over G to an appropriate $A_{F,r}$ star closure problem. To be precise, the word problem we consider is

$$W := \{w \in G^* \mid v(w) = e'\}$$

Let $G = \{g_1, g_2, \ldots, g_m\}$. Consider the word $u = \prod_{1 \le i \le m} g_i^{-1} g_i$ in A^*. Let $w = w_1 w_2 \ldots w_n$ be an instance of W. We map the instance w to the word $z = (\prod_{1 \le i \le n-1} w_i u) w_n$. Notice that $v(z) = v(w)$. Furthermore, it is not hard to see that by virtue of inserting the word u between w_i and w_{i+1} for $1 \le i \le n-1$ we have ensured that the word z can be decomposed into $z = \alpha_1 \alpha_2 \ldots \alpha_n$, where for $1 \le i \le n-1$ we have $|\alpha_i| < 4m$, w_i is included in α_i, and $v(\alpha_i) = e'$. Since $v(z) = v(w)$, it follows that $w \in W$ iff z can be decomposed as $\alpha_1 \alpha_2 \ldots \alpha_n$, where each α_i is of length at most $4m - 1$ and $v(\alpha_i) = e'$ for all i.

Letting $F = \{e'\}$ and $r = 4m - 1$ the above argument shows that $w \mapsto z$ is an AC^0 reduction from the NC^1-complete word problem W to the $(A_{F,r})^*$ closure problem.
(b) The $(A_{F,r})^*$ closure problem is the decision problem $(A_{F,r})^*$. Let us fix the following word problem over the monoid A:

$$W := \{w \in A^* \mid v(w) \in F\}$$

Let $w \in A^*$ be an instance of the problem. We will characterize words in the set $\overline{(A_{F,r})^*}$. To do so we first define the sets $Bad_r := \{x \in A^{=r} \mid v(x') \notin F \text{ for all prefixes } x' \text{ of } x\}$ and

$$Test := \{z \in A^* \mid z = u_1 x u_2, u_1 \in W, x \in Bad_r\}$$

[3] Notice that e' could be different from the monoid identity e.

We claim that $\overline{(A_{F,r})^*} = Test$. In order to see this it suffices to observe that if u and u' are two prefixes of w such that $v(u) \in F$ and $v(u') \in F$, and $u' = ux$ then it holds that $v(x) \in F$, since F is a group. This guarantees that for each $w \in (A_{F,r})^*$ we can break up $w = \alpha_1\alpha_2 \ldots \alpha_m$, where for each $1 \leq i \leq m$, $|\alpha_i| \leq r$, and α_i is the *first prefix* of $\alpha_i \ldots \alpha_m$ such that $v(\alpha_i) \in F$. The claim is now easy to see.

To complete the proof, note that to check if $z \in Test$, we can easily design an AC^0 circuit with oracle nodes that query the word problem W.
(c) This is an immediate consequence of part (b) and the results of [Bar89, BT88]. ∎

Remark 2 *We leave open the question of characterizing the complexity of* $(A_{F,r})^*$ *where* $F \subseteq A$ *is an* arbitrary subset. *It is not clear if the internal structure of the monoid* A *has an effect on the complexity of such* $(A_{F,r})^*$.

7 Concluding remarks

Finally, we have some easy observations on the inversion of formal power series beyond GapL. In particular, since context-free languages have been extensively studied using formal power series (see, for example, [KS85]), it may be of interest to know that the inverse of a context-free language is, in some sense, no more complex than the language itself. The following theorem makes this precise.

Theorem 7.1

(a) If $A \in$ LogUCFL *the formal power series* χ_A^{-1} *is computable in* GapLogCFL.

(b) If $A \in$ LogCFL *the formal power series* χ_A^{-1} *is computable in* GapLogCFL/*poly*.

The proof of part (a) is immediate once it is observed that the coefficients of the matrix presented in Lemma 4.1 can be computed in GapLogCFL. Part (b) follows because of the results of [RA97]. It is not clear how to make the second inclusion uniform, although it is pointed out in [AR98] that this inclusion does hold in the uniform setting if there are sets in DSPACE(n) with sufficiently high circuit complexity. It is open if there are corresponding hardness results.

As mentioned in the introduction, the main open question which this paper raises is to discover a classification of finite languages A so that the complexity of $\#A^*$ falls in different interesting subclasses of $GapNC^1$. This might give some insight into the internal structure of $GapNC^1$ which is still not well understood [CMTV96, All].

Acknowledgments

We thank David Mix Barrington and Howard Straubing for pointing out that the decision version of Theorem 5.5 easily follows from a classical result (see [Pin86]).

References

[AAD97] M. Agrawal, E. Allender, and S. Datta. On TC^0, AC^0, and arithmetic circuits. In *Proceedings of 12th Annual IEEE Conference on Computational Complexity*, pages 134–148, 1997.

[ABL98] A. Ambainis, D. A. M. Barrington, and H. LêThanh. On counting ac^0 circuits with negated constants. In *Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS), Lecture Notes in Computer Science 1450*, pages 409–419, 1998.

[All] E. Allender. Making computation count: Arithmetic circuits in the nineties. In L. Hemaspaandra, editor, *SIGACT News Complexity Theory Column, to appear.*

[AO96] E. Allender and M. Ogihara. Relationships among PL, #L, and the determinant. *RAIRO - Theoretical Information and Application*, 30:1–21, 1996.

[AR98] E. Allender and K. Reinhardt. Isolation, matching, and counting. In *Proceedings of 13th Annual IEEE Conference on Computational Complexity*, pages 92–100, 1998.

[Bar89] D.A. Barrington. Bounded-width polynomial size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38:150–164, 1989.

[BCH86] P.W. Beame, S.A. Cook, and H. J. Hoover. Log-depth circuits for division and related problem. *SIAM Journal on Computing*, 15:994–1003, 1986.

[BLM93] F. Bedard, F. Lemieux, and P. McKenzie. Extensions to Barrington's M-program model. *Theoretical Computer Science*, 107:31–61, 1993.

[BOC92] M. Ben-Or and R. Cleve. Computing algebraic formulas using a constant number of registers. *SIAM Journal on Computing*, 21:54–58, 1992.

[BR84] J. Berstel and C. Reutenauer. *Rational Series and Their Languages*, volume 12 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1984.

[BT88] D.A. Barrington and D. Thérien. Finite monoids and the fine structure of NC^1. *Journal of the Association of Computing Machinery*, 35:941–952, 1988.

[CMTV96] H. Caussinus, P. McKenzie, D. Thérien, and H. Vollmer. Nondeterministic NC^1 computation. In *Proceedings of 11th Annual IEEE Conference on Computational Complexity*, pages 12–21, 1996.

[Coo85] S. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64:2–22, 1985.

[FFK94] Stephen A. Fenner, Lance J. Fortnow, and Stuart A. Kurtz. Gap-definable counting classes. *Journal of Computer and System Sciences*, 48(1):116–148, February 1994.

[FS74] P. Flajolet and J. Steyaert. Complexity of classes of languages and operators. Technical Report Rap. de Recherche 92, IRIA Laboria, November 1974.

[Gin75] S. Ginsburg. *Algebraic and Automata-Theoretic Properties of Formal Languages*. North Holland, Amsterdam, 1975.

[KS85] W. Kuich and A. Salomaa. *Semirings, Automata, Languages*. EATCS Monograph. Springer-Verlag, 1985.

[Lan93] K.-J. Lange. Complexity and structure in formal language theory. In *Proceedings of 8th IEEE Structure in Complexity Conference*, pages 224–238, 1993.

[Li92] L. Li. Formal power series: An algebraic approach to the GapP and #P functions. In *Proceedings of 7th Structure in Complexity Theory Conference*, pages 144–154, 1992.

[Mon75] B. Monien. About the deterministic simulation of nondeterministic $(\log n)$-tape bounded Turing machines. In *Proceedings of 2nd GI Conference, Lecture Notes in Computer Science 33*, pages 118–126, 1975.

[MPT91] P. McKenzie, P. Péladeau, and D. Thérien. NC^1: The automata-theoretic viewpoint. *Computational Complexity*, 1:330–359, 1991.

[MV97] M. Mahajan and V Vinay. Determinant: Combinatorics, algorithms, and complexity. *Chicago Journal of Theoretical Computer Science, to appear*, 1997. Preliminary version in Proceedings of Eighth Annual ACM-SIAM Symposium on Discrete Algorithms SODA 1997, 730–738.

[Pin86] J. E. Pin. *Varieties of Formal Languages*. North Oxford Academic Publishers Ltd, London, 1986.

[RA97] K. Reinhardt and E. Allender. Making nondeterminism unambiguous. In *Proceedings of 38th IEEE Conference on Foundations of Computer Science*, 1997.

[Sal90] A. Salomaa. Formal languages and power series. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B*, pages 103–132. Elsevier, 1990.

[Sch65] M.P. Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8:190–194, 1965.

[SS78] A. Salomaa and M. Soittola. *Automata-Theoretic Aspects of Formal Power Series*. Springer-Verlag, 1978.

[Sud73] Sudborough. On tape-bounded complexity classes and multi-head finite automata. In *Proceedings of 14th IEEE Symposium on Switching and Automata Theory*, 1973.

[Tod91] S. Toda. Counting problems computationally equivalent to the determinant. manuscript, 1991.

[Tod92] S. Toda. Classes of arithmetic circuits capturing the complexity of computing the determinant. *IEICE Transactions on Information and Systems*, E75-D:116–124, 1992.

[Ven91] H. Venkateswaran. Properties that characterize LogCFL. *Journal of Computer and System Sciences*, 42:380–404, 1991.

[Ven92] H. Venkateswaran. Circuit definitions of nondeterministic complexity classes. *SIAM Journal on Computing*, 21:655–670, 1992.

[Vin91] V Vinay. *Semi-unboundedness and complexity classes*. PhD thesis, Indian Institute of Science, Bangalore, 1991.

Appendix

Alternative proof of Theorem 5.4

We use the characterization of GapNC^1 as GapBP[CMTV96]. That is, it was shown in [CMTV96] that any function in GapNC^1 can be represented as the difference of two functions that count paths through bounded-width branching programs. Let $g = \chi_L$ for some regular language L accepted by DFA $M = (Q, \Sigma, \delta, q_0, F)$. From Lemma 4.1,

$$g^{-1}(w) = \sum_{\substack{w = w_1 \cdots w_k \\ |w_i| \geq 1}} [-g(w_1)] \cdots [-g(w_k)]$$

for $w \neq \epsilon$. So what is required is to sum, over all decompositions of w into non-ϵ subwords in L, the sign of the decomposition, where the sign of a decomposition is roughly the parity of the number of subwords in the decomposition.

We construct a GapBP model which tracks such decompositions. Assume that $\Sigma = \{0, 1\}$. The idea is as follows: First consider the branching program of length $n+2$, width $|Q|$, that corresponds to M in the obvious way. (The last level is for combining all final states into one sink.) Now stretch the length of this branching program by a factor of 2 by introducing dummy levels alternating with those corresponding to M. These dummy levels provide positions to detect potential subwords in L. At the dummy level, apart from all the identity transitions, there are also transitions from each final state to the start state. Since each detected subword contributes -1 to the product along this decomposition, these edges should carry weight -1. Counting weights of paths in this branching program corresponds precisely to summing the signs of all valid decompositions. To remove negative edge weights, we stretch the width of the branching program by a factor of 2, maintaining two copies to account for parity (as in the proof of Theorem 3.3 in [CMTV96] where it is shown how iterated products of constant-size *integer* matrices can be computed in GapBP.).

Formally, the branching program for inputs of size n has length $2n + 2$, and width $2|Q|$. The nodes of the branching program are labeled (i, k, b), for $k \in Q$, $b \in \{0, 1\}$, and $0 \leq i \leq 2n$. At the last level there are two additional sink nodes t_+ and t_-. The source is the node $(0, q_0, 0)$. For odd i, include the edges $\langle (i, k, b), (i + 1, k, b)) \rangle$ for each k, b. Also include the edges $\langle (i, k, b), (i + 1, q_0, \bar{b})) \rangle$ for each $k \in F$, for each b. These edges are all labeled 1. For even $i = 2j$, include the edges $\langle (i, k, b), (i + 1, k', b) \rangle$ where $k' = \delta(k, 1)$; these edges are all labeled x_j. Also include the edges $\langle (i, k, b), (i + 1, k', b) \rangle$ where $k' = \delta(k, 0)$; these edges are all labeled $\bar{x_j}$. Finally, include edges $\langle (2n, q_0, 0), t_+ \rangle$ and $\langle (2n, q_0, 1), t_- \rangle$ labeled 1. ∎

ECCC ISSN 1433-8092
http://www.eccc.uni-trier.de/eccc
ftp://ftp.eccc.uni-trier.de/pub/eccc
ftpmail@ftp.eccc.uni-trier.de, subject 'help eccc'