Arithmetic Complexity, Kleene Closure, and Formal Power Series

Eric Allender* V Arvind† Meena Mahajan‡

April 2, 2003

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 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 Kleene closure is NLOG-complete, and there is a finite set for which inversion and root extraction are GapNC^1-complete and Kleene closure is NC^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.

Some results in this paper show properties of complexity classes that are interesting independent of formal power series considerations, including some useful closure properties and complete problems for GapL.

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

*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 grants CCR-9509603, CCR-9734918 and CCR-0104823.

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

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

NC^1 [Bar89, BT88, MPT91] essentially follows from the algebraic study of subclasses of 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, Lan96, 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 to 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 identifies 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, CMTV98].

We will also consider arithmetic complexity classes defined by branching programs. A constant-width branching program is a layered directed acyclic graph, with a constant number of vertices at each layer, and the problems of interest are (a) to count the number of source-to-sink paths, and (b) to decide if one exists. Such problems bear close relations to NC^1 and its counting versions (see [Bar89, CMTV98]), while the versions where the branching programs are allowed to have arbitrary width bear similar relations to NLOG, #L and GapL (see [Ven92, Tod91]).

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, Vin91b, MV97]); and GapNC^1 was also shown to have natural complete problems in [CMTV98]. Another reason for interest in these classes was provided in [AAD00]: $\#\mathrm{AC}^0$ and GapAC^0 were shown to characterize TC^0. (See [All97] for a detailed survey on arithmetic circuit classes. For more complete treatment, we refer the reader to the textbook [Vol99].) 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. Unfortunately these upper bounds are not tight enough in the context of the arithmetic classes we consider.

In the remainder of this section, we sketch the outline of the rest of the paper. Section 2 presents the basic definitions and notation for formal power series.

In Section 3 we prove a useful closure property of GapL. (We note that this has subsequently found application in other work [HT03, HT02, Wat99].)

Section 4 shows that computing the determinant and permanent of matrices having a very restricted format is complete for the complexity classes we consider.

In Sections 5 and 6 we prove our main results; we pinpoint the complexity of inversion and root extraction, and 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}; thus GapL forms a subgroup of GapP. 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 closed downward under AC^0 reductions, 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 finite sets (and in fact for all 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 $\mathrm{GapAC}^0 \neq \mathrm{GapNC}^1$ [AAD00].

In the results of Section 6, the Kleene closure operation 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 6 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 $w \in A^*$). We show that under appropriate reducibilities:

1. There is an AC^0 language A for which $\#A^*$ is complete for $\#\mathrm{L}$ (Theorem 6.2(a)),

2. For every regular language A, $\#A^*$ is in #BP (Theorem 6.4(c)), and

3. There is a finite language A for which $\#A^*$ is complete for #BP (Theorem 6.5(b)).

Thus, another contribution of this paper is to make explicit the connection between Kleene closure and computing determinants.

An 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 7 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. In this paper, the only rings $\mathcal{R}$ we consider are the integers ($\mathbb{Z}$) and the rationals ($\mathbb{Q}$).

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 $R \ll \Sigma^* \gg$.

The operations $+$ and $\cdot$ of the ring $\mathcal{R}$ are extended to $R \ll \Sigma^* \gg$ as follows. Let f and g be formal power series in $R \ll \Sigma^* \gg$. Then

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

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

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 $g \in \mathcal{C}$. It is also useful to have notation denoting formal power series corresponding to *language* classes, as opposed to *function* classes; 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 $\mathbb{Z} \ll \Sigma^* \gg(\mathrm{GapL})$ and $\mathbb{Z} \ll \Sigma^* \gg(\mathrm{GapLogCFL})$ by $\mathcal{GL}$ and $\mathcal{GLCFL}$ respectively.

3 A useful closure property of GapL

It is useful to define the following connection between the determinant function and classes of functions $\mathcal{C}$: Det$\cdot\mathcal{C}$ 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 $\mathbb{Z}$. The class* Det$\cdot\mathcal{C}$ *is the class of functions $f : \Sigma^* \longrightarrow \mathbb{Z}$ such that (1) there is a logspace many-one reduction h mapping each $w \in \Sigma^*$ to a square matrix M_w over Σ^*, (2) there is a function $g \in \mathcal{C}$ that transforms M_w to the matrix N_w over $\mathbb{Z}$ by the application of g to each entry in M_w, and (3) the determinant of N_w equals $f(w)$.*

Clearly, Det$\cdot\mathcal{C} \subseteq \mathrm{GapL}^{\mathcal{C}}$. The following results improve this bound for $\mathcal{C} = \mathrm{GapL}$.

Theorem 3.2 *Let B be an $n \times n$ matrix, whose entries are each polynomials of degree n in $\mathbb{Z}[x]$, 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.

Our proof is obtained by modifying either the reduction of [Ber84] or of [MV97], in which the problem of computing the determinant is reduced to the problem of counting paths in a DAG (i.e., a directed acyclic graph). To be specific, we will use as our starting point the reduction presented in [MV97]. As observed in Section 6.2 of [MV97], this reduction has the following property. Given a matrix M over a ring $\mathcal{R}$, the reduction produces a weighted DAG G with source s and sinks t_1, t_2, such that the determinant of M is equal to the difference $a_1 - a_2$, where a_l is the sum, over all paths ρ from s to t_l, of the product of the weights of the edges on the path ρ.

We must deal with the particular case where the ring $\mathcal{R}$ is the ring of integer polynomials. The first step in our modification is to is to make $n^2 + 1$ copies v_j of each vertex v of the DAG G, and add edges to maintain the property that for all vertices v of the original dag, if the sum over all paths from s to v is the polynomial $\sum_p d_p x^p$, then in the modified DAG G' the sum over all paths from s to v_p will be d_p. This is easily accomplished, by replacing edges from v to w labeled with a polynomial $\sum_q c_q x^q$ in the original graph G by edges labeled c_q from v_p to w_{p+q} in the modified graph G'. At this point, the coefficient of x^r in the determinant polynomial is the difference, of the sums (over all paths ρ from s to $t_{1,r}$, respectively $t_{2,r}$) of the product of all coefficients labeling edges on path ρ.

In the statement of the lemma, however, the coefficients labeling the edges are not given explicitly, but are computable in GapL. As a simpler case, consider first the situation where the coefficients are computable in #L. In this case, we can replace an edge of G' labeled with the coefficient of x^p in $B[i,j](x)$ by the corresponding DAG in which the number of source-to-sink paths equals this coefficient (and identifying its source and sink with the endpoints of the edge in G'). It is easily seen that the construction is correct in this case.

In the more general case where the coefficients are GapL-computable (and thus not necessarily non-negative), each DAG H_e corresponding to a coefficient labeling edge e has two sinks, and we must keep track of their contributions separately. This is achieved by making two copies v_{p+} and v_{p-} of each vertex v_p of our modified DAG G', with the property that the sum over all paths from s to v_p in G' will be equal to the difference in the number of paths from s to v_{p+} and v_{p-} in this new graph. Again, this is easily accomplished by making two copies of H_e, and connecting their sources and sinks as shown in Figure 1.

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

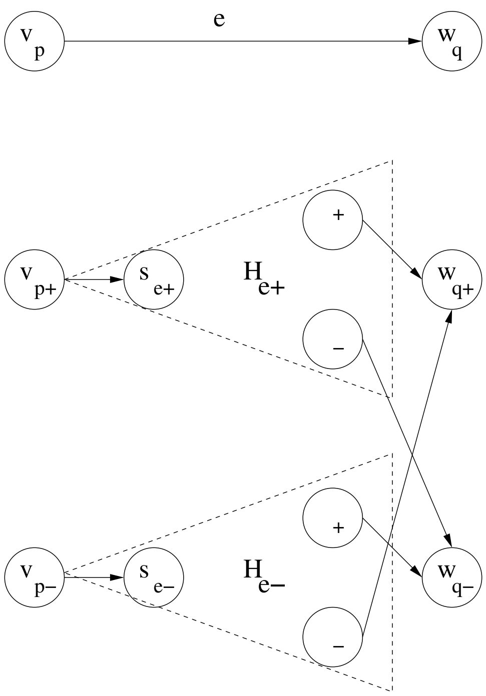


Figure 1: An edge e with weight computed in GapL by H_e, and the corresponding unweighted DAG.

The following corollary will be useful to us, and has also found application elsewhere [HT03, HT02, Wat99].

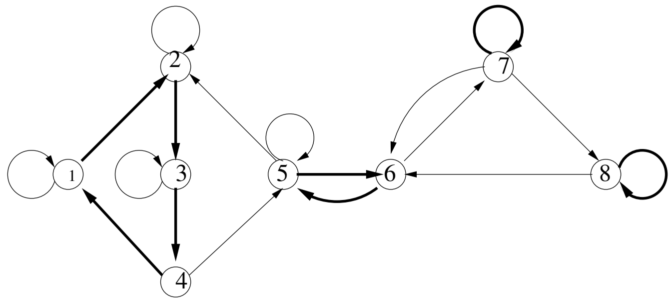
Corollary 3.3 Det·GapL = GapL. *i.e. Let B be an $n \times n$ matrix, whose entries are each computable in* GapL. *Then det(B) is computable in* GapL.

Corollary 3.4 Det·GapLogCFL = GapLogCFL.

Proof. This follows by an essentially-identical proof. The difference is that the graph H_e (representing an edge e in the digraph G') is not built explicitly, but instead is represented by (1) a start vertex s and (2) the two end vertices v_{p+} and v_{p-}, along with (3) the word w_e such that the value labeling edge e in G' is the value of the GapLogCFL function g applied to w_e. It is shown in [Vin91a] that every function in #LogCFL can be represented as the number of accepting computations of a nondeterministic logspace-bounded auxiliary pushdown automaton, and it is easy to see that a similar characterization holds for GapLogCFL. Now consider a nondeterministic logspace-bounded auxiliary pushdown automaton that follows a path through the graph G'. In this traversal, instead of following a path through each DAG H_e, it guesses a computation on input w_e, continuing from vertex v_{p+} if it finds an accepting computation on w_e, and continuing from v_{p-} otherwise. It is straightforward to show that this proves membership in GapLogCFL. ∎

$$
\begin{matrix}
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1
\end{matrix}
$$

An elt matrix and a permanent term



The corresponding graph and cycle cover

Figure 2: A term of the permanent of an elt matrix.

4 Some useful complete problems

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 5 and 6. A matrix M is said to be *elt* if $A_{i,j} = 0$ whenever $j > i + 1$.

Theorem 4.1 *Computing the determinant (or permanent) of elt matrices over the integers is complete for* GapL *w.r.t. logspace many-one reductions. Computing the permanent of elt matrices over the nonnegative integers is complete for* #L *w.r.t. logspace many-one reductions.*

Proof.

Recall that the permanent of a matrix M is given by:

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

where the sum is taken over all permutations π on the indices. If we consider M to be the adjacency matrix of a weighted digraph, G, then each nonzero term in this expression corresponds to a *cycle cover*: a collection of edges forming disjoint cycles such that each vertex lies on a cycle. Cycle covers in elt matrices have a particularly simple form. Each cycle in such a cover is either a self-loop, or it consists of adjacent rows corresponding to a cycle of the form $i, i+1, \ldots, i+j, i$. For example, Figure 2 shows a decomposition of a zero-one elt matrix into blocks corresponding to cycles of a cycle cover in the corresponding digraph.

It is easy to see that the permanent of elt matrices over the nonnegative integers is in #L: since the cycle covers have such simple form, an NLOG machine can guess such a cycle cover π, and generate $\prod_i M[i, \pi(i)]$ accepting paths for each such π.

Similar observations were made in the proof of Lemma 4.7 of [Li92], where it is shown that the permanent can be reduced to the determinant for elt matrices (and it is implicit in [Li92] that computing the determinant and permanent of elt matrices are equivalent under many-one reductions). Thus the results of [Li92] show that these problems are in GapL.

It remains only to show that the permanent of integer elt matrices is hard for GapL, and the permanent of nonnegative integer matrices is hard for #L.

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 in the number of paths from 1 to n and the number of paths from 1 to $n+1$ in G.

$$h_1(G) = \#\text{paths}_G(1 \rightsquigarrow n) - \#\text{paths}_G(1 \rightsquigarrow (n+1))$$

Without loss of generality, we can assume that no edge leaves vertex n, and that all paths from 1 to n or 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.

Toda [Tod91] showed how to express $h_1(G)$ as the determinant of a related matrix. However that matrix is not elt, so we need a somewhat different construction which we describe below.

Given such a digraph G we associate an elt matrix N_G with it as follows: Let B be the adjacency matrix of G. Note that B will have zeroes on and below the diagonal, given the structure of G. Modify B by putting 1s on the diagonal, negating the (n, n) entry, and setting the $(n, n+1)$ entry to 1, to get matrix C. Note that C is the adjacency matrix of a graph G' obtained by adding self-loops at all vertices of G and adding an edge from n to $n+1$. These new edges have weight 1, except for the self-loop at n which has weight -1. $1 \rightsquigarrow (n+1)$ paths in G' are in bijection with paths in G from 1 to n or $n+1$, with the following property:

1. A $1 \rightsquigarrow (n+1)$ path ρ in G' is of even length iff it uses the $n \longrightarrow n+1$ edge in G' iff it corresponds to a $1 \rightsquigarrow n$ path of G.
2. A $1 \rightsquigarrow (n+1)$ path ρ in G' is of odd length iff it does not use the $n \longrightarrow n+1$ edge in G' iff it corresponds to a $1 \rightsquigarrow (n+1)$ path of G.

Let D be the transpose of matrix C. N_G is obtained by deleting the first row and the last column of D. N_G looks like this:

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

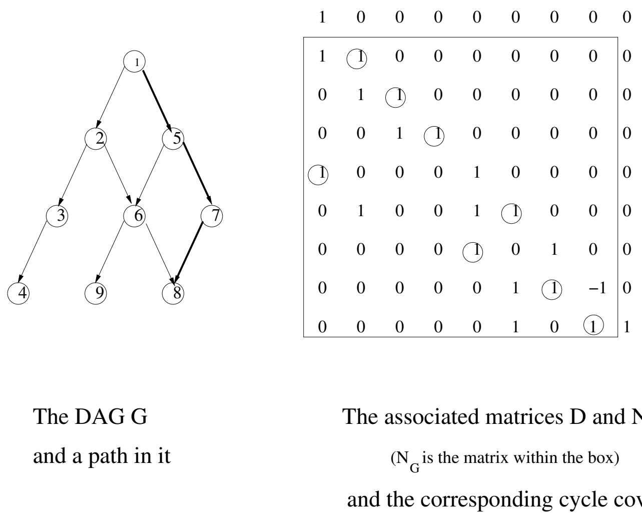


Figure 3: A $1 \rightsquigarrow n$ path and the decomposition in the associated elt matrix.

Note that the reduction from G to N_G can be easily implemented in logspace.

We now show that the permanent of N_G is equal to $h_1(G)$. The key observation is that paths in G from 1 to n or $n+1$ are in 1-1 correspondence with terms $\prod_i N_G[i, \pi(i)]$ of $perm(N_G)$. Why? Let η be such a path in G, and let ρ be the corresponding path in G'. Let S be the following set of edges of G':

$$S = \{(i,j) \mid (i,j) \in \rho\} \cup \{(i,i) \mid i \notin \rho\}.$$

In the graph G', for each vertex $i \neq n+1$, either a path successor or the self-loop at i is picked. Similarly, for each vertex $j \neq 1$, either a path predecessor or the self-loop at j is picked. So if we delete row 1 and column $n+1$ of D, we have covered each row and each column exactly once, and we get a term of $perm(N_G)$. If η is a path to n in G, then ρ uses the weight 1 edge $n \longrightarrow n+1$ of G', otherwise ρ uses the weight -1 self-loop at n in G'. So the corresponding term of $perm(N_G)$ is positive for paths to n and negative for paths to $n+1$.

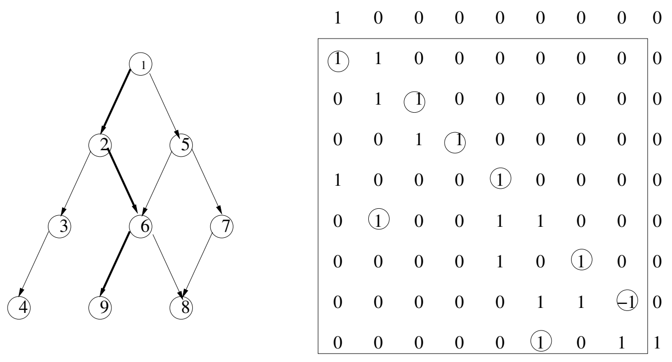
Note further that the length of ρ is exactly the number of cycles in the corresponding cycle cover of $perm(N_G)$; this fact will be useful in proving Theorem 6.2.

The reverse mapping, from terms of $perm(N_G)$ to paths in G, can be similarly argued. This completes the proof of hardness for GapL.

See Figure 3 for an illustration. The $v_1 \longrightarrow v_5 \longrightarrow v_7 \longrightarrow v_8$ marked in graph G corresponds to the even length path $v_1 \longrightarrow v_5 \longrightarrow v_7 \longrightarrow v_8 \longrightarrow v_9$ in G' and to the cycle cover (1,2,3,4) (5,6) (7) (8) marked in N_G.

Similarly, in Figure 4, the odd length path $v_1 \longrightarrow v_2 \longrightarrow v_6 \longrightarrow v_9$ marked in graph G corresponds to the same path in G' and to the cycle cover (1) (2,3,4,5) (6,7,8) marked in N_G.

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. ∎



Another path in G | The corresponding cycle cover

Figure 4: A $1 \rightsquigarrow (n+1)$ path and the decomposition in the associated elt matrix.

5 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 $f(\epsilon) \in \{1, -1\}$, where ϵ denotes the empty string. If $f(\epsilon) = -1$, then it is easy to verify that $f^{-1} = -(-f)^{-1}$, so without loss of generality we will consider only those f with $f(\epsilon) = 1$.

Lemma 5.1 (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:

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

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

Similarly, from Corollary 3.4 we have:

Proposition 5.3 *The set of formal power series $\mathcal{GLCFL}$ is closed under inverses.*

Thus $\mathcal{GL}$ is a subgroup of $\mathcal{GLCFL}$ which is a subgroup of GapP.

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 5.4 (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 for $k \geq 2$, $\sigma_r(k)$ 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 5.5 *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 5.1).

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 for $k \leq n$.

Lemma 5.6 *Let f be a formal power series with rational coefficients (i.e., $f \in \mathbb{Q} \ll \Sigma^* \gg$), 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 5.4, $f(w)$ 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

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

Hence, 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. By the basic closure properties of GapL (see [AO96]), it follows that $h(w)$ is computable in GapL.

Consider the following matrix, where x is an indeterminate:

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

The determinant of this matrix, $\mathrm{Det}(B_x)$, is a polynomial of degree $n-1$ in x. We claim that the coefficient of x^{n-k} in $\mathrm{Det}(B_x)$ is precisely $A_k(w)$. This follows by considering the cycle covers of this elt matrix, as in the proof of Theorem 4.1. The non-zero terms containing $(-x)^{n-k}$ correspond to permutations with sign $(-1)^{n-k}$, and they are easily seen to sum to $A_k(w)$. By Theorem 3.2, these coefficients are computable in GapL. ∎

Theorem 5.7 *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 5.6. 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 $H(|w|)$. Since it has recently been shown that division can be computed in logspace [CDL01] and even in uniform TC^0 [Hes01] (see also [All01, HAB02] for alternative expositions), the theorem follows. ∎

At this point it is natural to ask if there is any complexity class smaller than GapL with the property given in Proposition 5.2, or with properties analogous to those in Lemma 5.6 or Theorem 5.7. The next section essentially gives a negative answer to these questions.

6 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 6.1 *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 6.2 *(a) There is a language $A \in \mathrm{AC}^0$ such that $\#A^*$ is complete for* #L *under logspace many-one reductions (even* AC^0 *projections).*

(b) There is a language $B \in \mathrm{AC}^0$ such that computing g^{-1}, where g is the formal power series χ_B, is complete for GapL *under logspace many-one reductions (and in fact under* AC^0 *projections).*

(c) For the language $A \in \mathrm{AC}^0$ of part (a), computing the rth root of the formal power series χ_A (for all $r > 1$) is hard for #L *under logspace-Turing reductions.*

One proof of part (a) of this theorem essentially follows from results of Flajolet and Steyaert [FS74]. In [FS74], a language $A \in$ DLOG is constructed, and A^* is shown to be complete for NLOG. In [Lan96] it is pointed out that this language A is actually in AC^0. It is easy to verify that that the chain of reductions showing that A^* is NLOG-complete 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 it is also easy to verify that A^* is complete under AC^0 projections. (These are the same as first-order projections. See, for e.g., [Imm87, ABI97, Ste93, Ste95] for background and definitions on projections.)

However, we include a different proof here, both to keep the paper self-contained, and also because the outline of this proof serves to prepare the ground for some subsequent proofs in this paper.

Proof.

(a) It is obvious that, for any language A in DLOG, $\#A^*$ is in #L. (In fact, it is clear that this holds also for any A in ULOG, and by [RA00, ARZ99] this is likely to hold for all A in NLOG, too.) Thus it suffices to show #L-hardness.

We use the construction in the proof of Theorem 4.1. Let h_2 be the #L-complete problem considered there. It suffices to show that there is a language $A \in \mathrm{AC}^0$, 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 4.1, $h_2(x) = \#A^*(r(x))$.

Consider the matrix N_G from the proof of Theorem 4.1. It has exactly one negative entry, in position $[n-1, n]$ (corresponding to the $[n, n]$ entry of the matrix C defined there). We now consider the Boolean matrix M_G obtained by setting this entry also to $+1$.

The reduction r we build will take a graph G as input, and will output a particular string $\mathrm{enc}(M_G)$ that encodes the matrix M_G described above.[1] This string is a list, in row major form, of the encodings of each position (i, j) in M_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 M_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 $\mathrm{enc}(M_G)$ 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, where w satisfies the following properties

(1) w is of the form

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

where $N(j, 1) = 1$, **OR**

(2) w is of the form

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

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

(3) The only word $w \in A$ containing $\mathrm{enc}(n, n)$ as a substring is

$$\begin{array}{lll} & & enc(n-1,n) \\ enc(n,1) & \cdots & enc(n,n) \end{array}$$

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

That is, if w is a substring of $\mathrm{enc}(M_G)$ for a graph G satisfying the restrictions described earlier, then w is in A if it is a piece of $\mathrm{enc}(M_G)$ that corresponds to one cycle in a cycle cover (as in Theorem 4.1), where we impose the additional condition we *only* allow cycle covers containing a self-loop on node $n+1$. (Pieces of the matrix corresponding to a cycle cover are shown by the dashed lines in Figure 2.) Cycles in the cycle cover have the form $(1, 2, \ldots, j)$ in case (1) and $(i+1, i+2, \ldots, j)$ in case (2) above. It is easy to see that A is in AC^0.

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 M_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 M_G that include the self-loop on n as a cycle. Thus, $\#A^*(w)$ equals $\mathrm{perm}(N'_G)$, where N'_G is the top-left $(n-1) \times (n-1)$ submatrix of N_G. From the proof of Theorem 4.1, this is equal to $h_2(G)$.

(b) We obtain language B by removing the stipulation (3) in the definition of A in part (a). Let g be the formal power series χ_B. That g^{-1} can be computed in GapL follows from Proposition 5.2. It remains to show that g^{-1} is hard for GapL; we show this by reducing h_1 to g^{-1}. Recall that

$$\begin{aligned} h_1(G) &= \#\mathrm{paths}_G(1 \rightsquigarrow n) - \#\mathrm{paths}_G(1 \rightsquigarrow (n+1)) \\ &= \#\text{even paths}_{G'}(1 \rightsquigarrow (n+1)) - \#\text{odd paths}_{G'}(1 \rightsquigarrow (n+1)) \\ &= (\#\text{cycle covers of } M_G \text{ with even number of cycles}) \\ &\qquad\qquad - (\#\text{cycle covers of } M_G \text{ with odd number of cycles}). \end{aligned}$$

(The last equality follows from the proof of Theorem 4.1.)

Consider the same mapping r defined in part (a). For any w of the form $r(G)$ we have,

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

The properties of the encoding imply that the set of decompositions $w = w_1 \cdots w_k$ such that $\prod_{i=1}^k \chi_B(w_i) = 1$ is in bijective correspondence with all cycle covers of G. Furthermore, notice that $k = 2m$ for each cycle cover corresponding to a path from 1 to n and $k = 2m-1$ for each cycle cover corresponding to a path from 1 to $n+1$. Thus, the sum defining $g^{-1}(w)$ is $h_1(G)$ as desired.

(c) Let g be the formal power series χ_A for the language A defined in part (a), and let f be the formal power series $g^{1/r}$ for some positive integer r. To compute f, apply the expression for root extraction derived in Lemmas 5.4 and 5.5, and note that the only decompositions of $w = r(G)$ have an even number of pieces. Thus the expression simplifies to the following:

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

Now, using the fact that division and iterated integer multiplication are computable in uniform TC^0 [Hes01], it follows that a TC^0 circuit with a single oracle gate[2] for f can compute the #L-hard function $\#A^*(w)$. (Equivalently, a logspace machine making a single query to the functional oracle for f can compute the #L-hard function $\#A^*(w)$.) This completes the proof. ∎

We observe without proof that Theorem 6.2 generalizes to levels of the #L hierarchy. This hierarchy was defined in [AO96] as follows: define #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 6.3 *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 6.2 effectively puts an end to any plans to investigate the group-theoretic structure of 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.

First, we need to recall the relationship between GapNC^1 and branching programs. As defined in [CMTV98], the class #BP is the class of functions that can be expressed as the number of accepting paths in branching programs of width $O(1)$ and polynomial length. Although it is not known if #BP is equal to $\#\mathrm{NC}^1$, it is known that GapNC^1 is equal to the difference of two #BP functions. For a proof, and for more formal definitions, please consult [CMTV98].

Theorem 6.4 *Let A be any regular set and let g be the formal power series χ_A. Then*

(a) $g \in \mathbb{Z} \ll \Sigma^* \gg(\mathrm{NC}^1)$.

(b) $g^{-1} \in \mathbb{Z} \ll \Sigma^* \gg(\mathrm{GapNC}^1)$.

(c) $\#A^* \in \#\mathrm{BP}$.

Proof. (a) Follows from the fact that NC^1 includes all regular sets.

(b) 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

[2]We assume that f is accessed as a functional oracle, but it does not matter very much how the oracle answers are presented. For a query string w, the oracle can be assumed to return pair of integers $(p, q), q \neq 0$ defining the rational $p/q = f(w)$, or the oracle can return a binary string representing $f(w)$ to polynomially-many bits of accuracy.

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 [CMTV98] 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.

(A direct constructive proof follows from a minor modification of the proof below.)

(c) Let A be accepted by the deterministic finite-state automaton $M = (Q, \Sigma, \delta, q_0, F)$. We need to compute the number of ways in which the input w can be decomposed into non-empty pieces each of which is in A. We construct a #BP model which tracks such decompositions. Assume that $\Sigma = \{0, 1\}$. The idea is as follows: First consider the branching program of length n, width $|Q|$, that corresponds to M in the obvious way. 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 A. At the dummy level, apart from all the identity transitions, there are also transitions from each final state to the start state. Each path in this branching program corresponds uniquely to a distinct valid decomposition of w.

Formally, the branching program for inputs of size n has length $2n$, and width $|Q|$. The nodes of the branching program are labeled (i, k), for $k \in Q$, $0 \leq i \leq 2n$. The source is the node $(0, q_0)$. The sink is the node $(2n, q_0)$. For $1 \leq j \leq n$, include the edges $\langle(2j-1, k), (2j, k))\rangle$ for each $k \in Q$. Also include the edges $\langle(2j-1, k), (2j, q_0))\rangle$ for each $k \in F$. For $0 \leq j < n$, if $x_j = 1$ then include the edges $\langle(2j, k), (2j+1, k')\rangle$ where $k' = \delta(k, 1)$; otherwise include the edges $\langle(2j, k), (2j+1, k'')\rangle$ where $k'' = \delta(k, 0)$. It is straightforward to verify that this branching program has the desired number of accepting paths. ∎

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 [AAD00]. Since the formal power series of regular languages are in GapNC^1 (as indicated by Theorem 6.4), 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 that 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 6.5 *There is a finite language A such that*

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

(b) *$\#A^*$ is complete for* GapNC^1 *under* AC^0 *reductions which make exactly one oracle access.*

(c) *for each $s > 1$, computing the sth root of the formal power series χ_A is hard for* GapNC^1 *under* TC^0 *reductions using exactly one oracle access.*

Proof. (a) By Theorem 6.4 computing the inverse of the formal power series χ_A is in GapNC^1 for any finite language A. Thus, we only need to show hardness.

We use the fact that GapNC^1 coincides with GapBP[CMTV98]. A complete problem for GapBP and hence for GapNC^1 is to compute the difference of two functions in width-six #BP. That is, for each function f in GapNC^1, we can build a family of width-6 BPs B_n, each with a source vertex s and two sink vertices t and t', such that

$$\forall n, \forall x : |x| = n, \quad f(x) = \#paths_{B_n}(s \rightsquigarrow t) - \#paths_{B_n}(s \rightsquigarrow t')$$

Without loss of generality, we assume that s is the first vertex at layer 1 of the BP, and vertices t and t' are the vertices numbered 1 and 2 at the last layer of the BP. We can also assume that all source-to-sink paths in the BP are of exactly the same length, say m.

We show that there is a finite A such that computing the inverse of the formal power series χ_A is hard for GapBP. The language A we construct will have alphabet Σ equal to the set of all relations on $\{1, 2, \cdots, 6\}^2$. Each element of Σ can be viewed as a bipartite graph on $\{1, 2, \cdots, 6\}^2$, with two vertical columns of 6 vertices each, and edges going from the left column to the right column. Thus a word in Σ^* represents a width-6 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. Clearly, A is finite as it contains no word of length greater than 21.

Next, we describe an AC^0 projection r, such that for any $w \in \Sigma^*$, we have

$$\chi_A^{-1}(r(w)) = \#\text{paths}_w(s \rightsquigarrow t) - \#\text{paths}_w(s \rightsquigarrow t')$$

This will prove the hardness result.

(1) The first step in computing $r(w)$ is to insert 6 copies of the relation $\{(i, i{+}1 (\text{mod } 6)) | 1 \leq i \leq 6\}$ in between each two consecutive symbols of w. Note that inserting these 6 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'.

(2) The next step is to insert the identity relation $id = \{(i, i) | 1 \leq i \leq 6\}$ between each two symbols of w' to get w''.

(3) Finally, the symbol α defining the relation $\{(i, i) | i \neq 2, 1 \leq i \leq 6\} \cup \{(2, 1)\}$ is appended at the end of the string w''.

Consider an edge from i to j in some column (not the last) in the original word w. In $r(w)$, this edge is replaced by the following sequence of length 14:

$$i \to j \to j \to j + 1(\text{mod}6) \to j + 1(\text{mod}6) \cdots \to j + 5(\text{mod}6) \to j + 5(\text{mod}6) \to j \to j$$

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

Now for each edge $i \to j$, the corresponding sequence of 14 edges in $r(w)$ visits 1 twice if $j \neq 1$ and four times if $j = 1$. Also, if we 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 subsequence of edges from a 1 to a 1, avoiding 1 in any intermediate position, occurs if $j = 6$ and $k = 2$ and is of length 21.

We now consider the decompositions of $r(w)$ into words in A corresponding to the different $s \rightsquigarrow t$ and $s \rightsquigarrow t'$ paths in w.

Recall that the formal power series for χ_A^{-1} on an input $r(w)$ is given by

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

We claim that there is a bijective correspondence between $s \rightsquigarrow t$ and $s \rightsquigarrow t'$ paths in w and the decompositions $r(w) = w_1 \cdots w_k$ such that $\prod_{i=1}^k \chi_A(w_i) = 1$. More precisely, the $s \rightsquigarrow t$ paths in w are in bijective correspondence with decompositions $r(w) = w_1 \cdots w_k$ such that $\prod_{i=1}^k \chi_A(w_i) = 1$, where $w_k = \alpha$ (equivalently $|w_k| = 1$). On the other hand, the $s \rightsquigarrow t'$ paths in w are in bijective correspondence with decompositions $r(w) = w_1 \cdots w_k$ such that $\prod_{i=1}^k \chi_A(w_i) = 1$, where $w_k \neq \alpha$ (equivalently $|w_k| > 1$).

To see this, first notice that, by construction, each decomposition with $w_k = \alpha$ defines a unique $s \rightsquigarrow t$ path in w. Conversely, two distinct $s \rightsquigarrow t$ paths in w give rise to distinct decompositions of $r(w)$ into words in A. (Why? 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.)

For the case $w_k \neq \alpha$, observe that α is the last letter of w_k. Let $w_k = w_k' \alpha$. For w_k to be in A, it must have a path from 1 in the leftmost column to 1 in α without any 1 in intermediate positions. This is possible only if w_k' has a path from 1 in the leftmost column to the 2 in its rightmost column. Now, it is easy to see that decompositions with $w_k \neq \alpha$ are in bijective correspondence with $s \rightsquigarrow t'$ paths in w.

Finally, we claim that $k = 2m$ if $w_k = \alpha$ and $k = 2m - 1$ otherwise. To see this, note that $|r(w)|$ is even with the last symbol as α and every intermediate symbol in even position as id. First, consider $r(w) = w_1 \cdots w_k$ such that $\prod_{i=1}^k \chi_A(w_i) = 1$, where $w_k = \alpha$. Since id is neither a proper prefix nor a proper suffix of any word in A, it follows that $|w_{2i}| = 1$, and each $w_{2i} = id$, for all i such that $2i < k$. By construction, the substrings w_{2i-1} correspond to the m edges of the underlying $s \rightsquigarrow t$ path. Thus $k = 2m$. Next, for $w_k \neq \alpha$, notice that w_k corresponds to the last edge of the $s \rightsquigarrow t'$ path underlying the decomposition. Hence, $k = 2m - 1$.

It now follows that the coefficient of $r(w)$ in the formal power series for χ_A^{-1} is the difference in the number of $s \rightsquigarrow t$ paths and number of $s \rightsquigarrow t'$ paths in w.

Figure 5 illustrates how a $s \rightsquigarrow t$ path ($s \rightsquigarrow t'$ path) in w is mapped to a $s \rightsquigarrow t$ path in $r(w)$ and how this path allows $r(w)$ to be decomposed into an even (odd, respectively) number of pieces.

(b) From Theorem 6.4(c) it follows that $\#A^*$ can be computed in $\#\mathrm{NC}^1$.

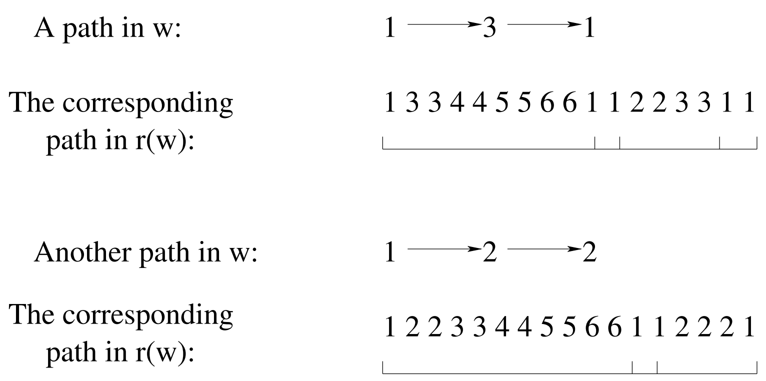


Figure 5: From width-6 BPs to $\#A^*$.

To show hardness under AC^0 reductions, we show how an AC^0 circuit with oracle gates for $\#A^*$ can compute $\#paths_{B_n}(s \rightsquigarrow t) - \#paths_{B_n}(s \rightsquigarrow t')$.

The circuit first computes from w two strings very similar to $r(w)$. Let us call them $r_1(w)$ and $r_2(w)$. $r_1(w)$ is obtained from $r(w)$ by appending id at the end of w'' instead of α. Note that k is even for every decomposition $r(w) = w_1 \cdots w_k$ such that $\prod_{i=1}^{k} \chi_A(w_i) = 1$, and hence, as argued in part (a) above, these decompositions are in bijective correspondence with $s \rightsquigarrow t$ paths. Similarly, $r_2(w)$ is obtained from $r(w)$ by appending the cyclic shift $(i+1 \longrightarrow i)$ at the end of w'' instead of α. Now k is odd for every decomposition $r(w) = w_1 \cdots w_k$ such that $\prod_{i=1}^{k} \chi_A(w_i) = 1$, and hence these decompositions are in bijective correspondence with $s \rightsquigarrow t'$ paths.

Thus we have

$$f(w) = \#paths_{B_n}(s \rightsquigarrow t) - \#paths_{B_n}(s \rightsquigarrow t') = \#A^*(r_1(w)) - \#A^*(r_2(w))$$

which can be computed by an AC^0 circuit with two oracle gates.

This construction can be further modified to yield an AC^0 circuit with a single oracle gate computing this quantity. Since B_n is of constant width and polynomially long, $\#paths_{B_n}(s \rightsquigarrow t')$ can be represented using only polynomially many bits, say m bits. We can, using standard techniques, convert B_n to another width-6 branching program B (whose length is $2m$ more than that of B_n) with source s_0, sink t_0, such that

$$\#paths_B(s_0 \rightsquigarrow t_0) = 2^m \#paths_{B_n}(s \rightsquigarrow t) + \#paths_{B_n}(s \rightsquigarrow t').$$

Now applying the construction r_1 to this branching program, we can compute all the bits of $\#paths_{B_n}(s \rightsquigarrow t)$ and $\#paths_{B_n}(s \rightsquigarrow t')$ in a single oracle call and then compute the required difference.

(c) For the reduction r_1 of part (b), similar to the proof of part (c) in Theorem 6.2, we get, for any $s > 1$

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

The new results of [Hes01] show that division (and hence iterated integer multiplication) is computable in uniform TC^0. Thus there is a uniform TC^0 circuit which has exactly one oracle gate accessing the bits of $\chi_A^{1/s}(r(w))$ and which computes $\#A^*(r(w))$. This completes the proof. ∎

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 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 6.2. 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 6.6 *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* NLOG $= \mathcal{C}$.

7 The Kleene closure of finite languages

In light of Theorem 6.5, 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 7.1 *Let $(A, \circ)$ be a finite monoid.*[3] *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.

To give the intuition behind the formal definition, consider Barrington's construction[Bar89] of branching programs for NC^1. There, width-5 graphs are considered, where the edges at each layer form a permutation. Acceptance and rejection correspond, respectively, to the permutation product across layers evaluating to identity or to some other fixed permutation. Implementing the proof idea of Theorem 6.5 on these branching programs, we can state membership in any NC^1 language as an $(A_{F,r})^*$ closure problem.

Corollary 7.2 *Let A be the permutation group S_5, let $F = \{\pi \in S_5 \mid \pi(1) = 1\}$, and let $r = 17$. 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.

[3]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.

Theorem 7.3 *(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*-Turing reductions 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'.[4] 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 $A^*_{F,r}$ 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 $w = w_1w_2\ldots w_n$ 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:

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

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

$$Test := \{z \in A^* \mid z = u_1xu_2, u_1 \in W, x \in Bad_r\}$$

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 $1 \le i \le m$, $|\alpha_i| \le r$, 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 1 *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 $(A_{F,r})^*$.*

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

8 Concluding remarks

Our main contribution is to show that the study of formal power series in computational complexity, as introduced by [Li92], can be used to explore aspects of the complexity classes GapL and GapNC^1.

Although tight connections exist between $\#A^*$ and the arithmetic circuit classes $\#\mathrm{NC}^1$ and #L, it is not at all clear that this connection exists for larger arithmetic classes. In particular, it might seem natural to conjecture that there is a set A in LogDCFL such that A^* is complete for LogCFL and $\#A^*$ is complete for #LogCFL. Neither is known to be true, and in fact it would be quite surprising if this were shown to be true. For instance, it is observed in [LH97, Lan88] that applying the Kleene * operation to deterministic context-free languages characterizes $\mathrm{NLOG}^{\mathrm{LogDCFL}}$, which is contained in LogCFL but is not known to coincide with it.

It is worth mentioning one additional easy observation about the inversion of formal power series beyond GapL. 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 8.1

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

(b) *If* $A \in \mathrm{LogCFL}$ *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 5.1 can be computed in GapLogCFL. Part (b) follows because of the results of [RA00]. 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, an interesting 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 [CMTV98, All97].

Acknowledgments

We thank David Mix Barrington and Howard Straubing for pointing out that the decision version of Theorem 6.5 easily follows from a classical result (see [Pin86]).

References

[AAD00] M. Agrawal, E. Allender, and S. Datta. On TC^0, AC^0, and arithmetic circuits. *Journal of Computer and System Sciences*, 60:395–421, 2000.

[ABI97] Eric Allender, José Balcázar, and Neil Immerman. A first-order isomorphism theorem. *SIAM Journal on Computing*, 26(2):557–567, April 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.

[All97] E. Allender. Making computation count: Arithmetic circuits in the nineties. *ACM SIGACT News Complexity Theory Column*, 28(4):2–15, 1997.

[All01] E. Allender. The division breakthroughs. In *The Computational Complexity Column in Bulletin of the EATCS*, volume 74, pages 61–77. EATCS, 2001.

[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.

[ARZ99] E. Allender, K. Reinhardt, and S. Zhou. Isolation, matching and counting: uniform and nonuniform upper bounds. *Journal of Computer and System Sciences*, 59:164–181, 1999.

[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.

[Ber84] S. J. Berkowitz. On computing the determinant in small parallel time using a small number of processors. *Information Processing Letters*, 18:147–150, 1984.

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

[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.

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

[CMTV98] H. Caussinus, P. McKenzie, D. Thérien, and H. Vollmer. Nondeterministic NC^1 computation. *Journal of Computer and System Sciences*, 57:200–212, 1998.

[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.

[HAB02] W. Hesse, E. Allender, and D. A. M. Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *Journal of Computer and System Sciences*, 65:695–716, 2002.

[Hes01] William Hesse. Division is in uniform TC^0. In *Proceedings of the International Colloquium on Automata, Languages and Programming ICALP*, pages 104–114. Springer, 2001. Lecture Notes in Computer Science LNCS 2076.

[HT02] T. M. Hoang and T. Thierauf. The computational complexity of the inertia. In *Proceedings of 22nd International Conference on Foundations of Software Technology and Theoretical Computer Science FST&TCS*, pages 206–217, 2002. Lecture Notes in Computer Science LNCS 2556.

[HT03] T. M. Hoang and T. Thierauf. The complexity of the characteristic and the minimal polynomial. *Theoretical Computer Science*, 295(1–3):205–222, 2003.

[Imm87] H. Immerman. Languages which capture complexity classes. *SIAM J. Comput.*, 4:760–778, 1987.

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

[Lan88] K.-J. Lange. Decompositions of nondeterministic reductions. *Theoretical Computer Science*, 58:175–181, 1988.

[Lan96] K.-J. Lange. Complexity and structure in formal language theory. *Fundamenta Informaticae*, 25:327–352, 1996. Preliminary version in Proceedings of 8th IEEE Structure in Complexity Theory Conference, 1993, 224–238.

[LH97] K.-J. Lange and M. Holzer. On the complexity of iterated insertions. In G. Paun and A. Salomaa, editors, *Trends in Formal Languages, Lecture Notes in Computer Science 1218*, pages 440–453. Springer, 1997.

[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*, 1997(5), 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.

[RA00] K. Reinhardt and E. Allender. Making nondeterminism unambiguous. *SIAM Journal on Computing*, 29:1118–1131, 2000. Preliminary version in Proceedings of 38th IEEE Conference on Foundations of Computer Science, 1997, pp 244–253.

[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.

[Ste93] Iain A. Stewart. Methods for proving completeness via logical reductions. *Theoretical Computer Science*, 118(2):193–229, 27 September 1993.

[Ste95] Iain A. Stewart. Completeness of path-problems via logical reductions. *Information and Computation*, 121(1):123–134, 15 August 1995.

[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.

[Vin91a] V Vinay. Counting auxiliary pushdown automata and semi-unbounded arithmetic circuits. In *Proceedings of 6th Structure in Complexity Theory Conference*, pages 270–284, 1991.

[Vin91b] V Vinay. *Semi-unboundedness and complexity classes*. PhD thesis, Indian Institute of Science, Bangalore, 1991.

[Vol99] H. Vollmer. *Introduction to Circuit Complexity: A Uniform Approach*. Springer-Verlag New York Inc., 1999.

[Wat99] John Watrous. On quantum and classical space-bounded processes with algebraic transition amplitudes. In *IEEE Symposium on Foundations of Computer Science*, pages 341–351, 1999.