Information and Computation **169**, 174–185 (2001)
doi:10.1006/inco.2000.3023, available online at http://www.idealibrary.com on IDEAL®

Computing a Context-Free Grammar-Generating Series

Bruce Litow

School of Information Technology, James Cook University, Townsville, 4811 Australia

Received October 1, 1998

The parallel complexity of computing context-free grammar generating series is investigated. It is known that this problem is in DIV, but in terms of n^σ rather than n, where n is the index of the desired coefficient and σ is the grammar size. A new method is presented which is in DIV in terms of $2^{2^{O(\sigma)}} \cdot n$. Evidence is provided that any direct application of elimination theory to this problem leads to a space and time resource factor that is nearly exponential in grammar size.

1. INTRODUCTION

The Basic Problem

We are interested in understanding how hard it is in terms of the NC model of parallel computation to compute the generating series of a context-free grammar. We describe the model in the next subsection. The model may be regarded as an embodiment of the concept of "polynomially many simple processors operating in polylogarithmic time." Details about context-free grammars and their generating series are presented in the following subsection.

The context-free grammar generating series problem has been investigated already by Huynh [12, 13] and Bertoni *et al*. [3]. Henceforth we will refer to this as the CFG-GEN problem. In Bertoni *et al*. it is shown that the CFG-GEN problem, i.e., the computation of the nth coefficient of a context-free grammar generating series, is no harder than computing the quotient of two integers, each requiring at most n bits in binary notation. However, there remains a basic open question about this problem. The method of [3] leads to a factor of n^σ in both their time and space bounds where σ is the size of the grammar. Can this exponential influence of the grammar size be eliminated while retaining their parallel complexity result?

In this paper we do three things. We present a simple algorithm that runs in $(\log(\sigma \cdot n))^3$ time and $(\sigma \cdot n)^{O(1)}$ processors and then an algorithm whose performance is similar to that in [3], but based on a quite different approach. In our approach, which uses the elimination theory for finite systems of polynomials, the grammar size σ appears doubly exponentially, but it only multiplies n, i.e., the algorithm is in DIV in terms of $2^{2^{O(\sigma)}} \cdot n$. In addition this factor of $2^{2^{O(\sigma)}}$ is the result of a single preprocessing step namely an application of elimination via the Kuich–Salomaa Algorithm. Finally we present evidence that any algorithm for the CFG-GEN problem that is based on the use of elimination theory for systems of polynomial equations must involve σ exponentially.

The NC Model of Parallel Computation

We give brief definitions for the parallel complexity classes that figure in this paper. Let k be a positive integer. A wealth of information on Boolean circuit theory may be found in [23]. A family of Boolean circuits $\{C_n, n = 1, \ldots\}$, where n indicates the number of input bits is said to have polynomial size and $(\log n)^k$ depth if C_n contains $n^{O(1)}$ gates and can be topologically sorted into $(\log n)^k$ levels. A circuit family is said to be log-uniform if there is a deterministic Turing machine operating in $O(\log n)$ space that can produce a description of C_n. Details about the notion of uniformity can be found in [4, 21, 6]. NCk is the collection of problems that can be computed by log-uniform, polynomial size and $(\log n)^k$ depth circuit families. The union over all NCk is called NC. The parameter k accords with the notion of $(\log n)^k$ parallel time.

DIV is the class of problems that can be NC1 reduced to integer division. That is for each problem in DIV there is a polynomial size, $\log n$ depth, log-uniform circuit family that maps each problem instance

0890-5401/01 $35.00

of size n bits to an instance of integer division. The exact NC time complexity of integer division is not known. It is somewhere between $\log n$ depth and $\log n \times \log\log n$ depth. See [2, 20, 17].

The Context-Free Grammar Generating Series Problem

A context-free grammar (CFG) G is a tuple $G = (A, X, x_1, P)$, where A is an alphabet, $X = \{x_1, \ldots, x_r\}$ is the nonterminal set, x_1 is the initial nonterminal and P is the set of productions. Recall that a production has the form $x_i \to \beta$, where β is a string in $(A \cup X)^*$. Of course, P is a finite set. See [ ] for details about CFGs.

A CFG is said to be in Chomsky normal form if all of its productions fall under one of three types.

- $x_1 \to \lambda$, where λ is the empty string.
- $x_i \to a$, where $a \in A$.
- $x_i \to x_j x_k$.

Any CFG can be converted into a CFG in Chomsky normal form in time polynomial in the size of the grammar. We will assume that our grammars are in Chomsky normal form. In addition, without loss of generality we will assume that the production $x_1 \to \lambda$ is excluded.

For our purposes a formal series F is a mapping $F : A^* \to \mathbf{N}$, where $\mathbf{N}$ designates the nonnegative integers. It is customary to write $F = \sum_w F(w) \cdot w$. A formal polynomial is just a formal series F for which there exists $n \in \mathbf{N}$ such that $|w| \geq n$ ($|w|$ is string length) implies $F(w) = 0$. See [1] for details about formal series and their relation to grammars. It is classical that a CFG G can be viewed as a finite system of equations $x_i = \beta_1 + \cdots + \beta_q$, one for each nonterminal x_i. The list $\beta_1, \ldots, \beta_q$ includes each righthand side of a production $x_i \to \beta_j$. For a CFG in Chomsky normal form either $\beta_j \in X^2$, or $\beta_j \in A \cup \{\lambda\}$. A solution of G is a list of formal series $F_1, \ldots, F_r$ such that the substitutions $x_i \leftarrow F_i$ convert the equations of G to identities in formal series. Note that two formal series are equal iff they are equal term-by-term. It is well-known that a Chomsky normal form CFG has a unique solution. See Chapter 14 of [16]. It is also well-known that $F_i = \sum_w F_i(w) \cdot w$, where $F_i(w)$ is the number of distinct leftmost derivations of w from x_i. Under the mapping that sends every symbol of A to the same symbol, say, t, we get the generating series $\hat{F}_i$ of the nonterminal x_i, i.e.,

$$\hat{F}_i = \sum_{n=0}^{\infty} \sum_{|w|=n} F_i(w) \cdot t^n.$$

We define $\hat{F}_i(n)$ by $\hat{F}_i(n) = \sum_{|w|=n} F_i(w)$. In particular, we will refer to $\hat{F}_1$ as the generating series of the CFG G. Note that if G is an unambiguous CFG, $\hat{F}_1(n)$ is the number of strings of length n in the language generated by G.

An instance of CFG-GEN problem is the computation of $\hat{F}_1(n)$ for a given grammar G, and a nonegative integer n.

2. A SIMPLE REFINEMENT OF BACK SUBSTITUTION

The technique sketched in this section has other uses outside the scope of this paper. An example of its range of applicability can be found in [18].

Let $G = (A, X, x_1, P)$ be a CFG and let n be a positive integer. We construct another CFG, G_n, which produces all words of length n produced by G and only them. The grammar G_n is defined to be $G_n = (A, X \times \{1, \ldots, n\} \times \{1, \ldots, n\}, (x_1, 1, n), P')$. The elements of the production set P' are given next.

- If $x_i \to a$, and $1 \leq p \leq n$, $(x_i, p, p) \to a (a \in A)$.
- If $x_i \to x_j x_k$, and $1 \leq p < q \leq n$, $(x_i, p, q) \to (x_j, p, p')(x_k, p' + 1, q)$, for all p' satisfying $p \leq p' < p$.

Observe that the role of the nonterminal (x_i, p, q) is to produce the segment from position p through position q of a word produced by x_i in G. Also, strictly speaking the set of nonterminals for G_n is that subset of $X \times \{1, \ldots, n\} \times \{1, \ldots, n\}$ for which, under the rules just itemised there is at least one production.

The equation sytem G_n has at most $\sigma \cdot n^2/2$ equations, where σ is the size of G. It is straightforward to prove that G_n has a unique solution $(F_1', \ldots, F_r')$ such that

$$F_1' = \sum_{|w|=n} F_1(w) \cdot w.$$

The proof method follows very closely any standard proof of correctness of the Cocke–Younger–Kasami Algorithm. See its description in [11] or its original formulation in [24].

From this point on we regard the symbols of A as commuting with each other and the symbols of X. It is easy to compute F_1' from G_n by using a careful form of back-substitution. Define the rank of a nonterminal (x_i, p, q) of G_n to be $q - p$. Thus, rank goes from 0 to $n - 1$. Observe that if $(x_i, p, q) \to (x_j, p, p')(x_k, p' + 1, q)$, at least one of $p' - p$ and $q - p' - 1$ is at most $(q - p)/2$. Define V_0 to be the set of nonterminals of rank 0. If ℓ is a positive integer, V_ℓ is the set of nonterminals of rank greater than $2^{\ell-1} - 1$ and less than 2^ℓ. It is clear that $1 \le \ell < \lceil \log n \rceil$.

Consider all of the equations E_ℓ of G_n in which the lefthand side nonterminal is in V_ℓ. If $\ell = 0$, we can replace these lefthand side nonterminals by the righthand sides which are formal series. If $\ell > 0$, and all nonterminals in $V_{\ell-1} \cup \cdots \cup V_0$ have been replaced by formal series, E_ℓ is now a linear system in which all remaining nonterminals occurring on the righthand side of an equation are in V_ℓ. This system can be solved by conventional means to produce formal series for all of the nonterminals in V_ℓ. In this way we finally obtain the full solution to G_n in $O(\log n)$ steps. The size (number of equations) of the system to be solved at step ℓ is $O(\sigma \cdot \frac{n^2}{2^{2\ell}})$, so the sequential time to solve G_n is $O((\sigma \cdot n)^{2\eta})$, where η is the exponent for the time required to solve linear systems. The parallel time is $O((\log(\sigma \cdot n))^3)$ since there are $\log n$ linear systems of $(\sigma \cdot n)^{O(1)}$ size. We do not go into arithmetic complexity.

3. A GENERATING SERIES ALGORITHM IN DIV BASED ON FURSTENBERG'S THEOREM

The Rightlinear Case

We start by motivating our approach with the special case of rightlinear grammars. A rightlinear grammar can be converted into a system of equations, one for each nonterminal, which is linear in the nonterminals. If every occurrence of every terminal symbol in these equations is replaced by a new indeterminate, say, t, and t and the nonterminals are regarded as real variables, then the system can be solved and each nonterminal will have as its solution a rational function in t. Under very mild assumptions, the series expansion about $t = 0$ of the rational function solution for a nonterminal will yield the generating series for that nonterminal. In other words, we will get a series $\sum_{n=0}^{\infty} c_n t^n$, such that c_n is the sum over all length n terminal strings of all derivations starting from the given nonterminal.

An explicit rational function $P(t)/Q(t)$, where $P(t)$ and $Q(t)$ are rational polynomials, and whose expansion about $t = 0$ is the generating series for a given nonterminal can be computed by solving a linear system of size σ, where σ is the size of the rightlinear grammar. If the grammar size is regarded as a constant, then it is not difficult to logspace reduce the computation of c_n to division of n-bit integers. We follows the usual convention and normalise the coefficients of Q so that $Q(0) = 1$. Note that the sizes of P and Q depend only on the grammar size, and so are constants. Write $Q(t) = 1 - R(t)$, such that $R(0) = 0$. Note that c_n can be obtained by summing all coefficients of t^n in $P(t)(1 + R(t) + \cdots + (R(t))^n)$. The computations required to do involve $O(n)$-fold additions and multiplications of $O(n)$-bit integers and by results in [2], this is logspace reducible to n-bit integer division (DIV).

We point out that we did not need to assume anything about the rightlinear grammar. In particular, the above method works whether or not the grammar is unambiguous. If we consider the grammar size σ to be a variable, computation of $P(t)/Q(t)$ is in NC2 in terms of σ. Subsequent arithmetic computations will involve n-fold multiplication of $O(\log \sigma)$-bit integers, so σ spoils the simple DIV complexity bound.

What we want to do in this section is directly generalise the rational function approach that works for rightlinear grammars to any context-free grammar. We will present an algorithm for computing the coefficients of the generating series of a given nonterminal which is in DIV in word length, subject to treating the grammar size as a constant. The extension is achieved by combining three techniques: the Kuich–Salomaa algorithm, Furstenberg's theorem and ordinary polynmial interpolation via the fast Fourier transform.

Bertoni, Goldwurm, and Massazza [3] have shown that computing the coefficients of a G-generating series is in DIV. However, the grammar size σ is interleaved with the word length n in an essential way. In particular, they need to work with a size n^σ domain of σ-tuples of integers. These tuples are used in carrying out a multivariate Lagrange interpolation. Our technique is rather different in that only a precomputation whose complexity is a function of σ is used, after which straightforward univariate polynomial interpolation suffices.

It may be useful to the reader to point out some related research on ranking languages. For words w, $v \preceq w$ indicates that either $v = w$ or v strictly precedes w in ordinary lexicographic ordering. If L is a language and $w \in L$, then the rank of w is Card $\{v \in L \mid v \preceq w\}$. Computing the binary notation for the ranks of words in L is known as ranking L. Huynh [12, 13] has shown that

- Ranking regular languages presented by deterministic finite automata is in DIV.
- Ranking languages accepted by one way unambiguous auxiliary PDA operating in polynomial time is in NC2. An auxiliary PDA is an ordinary pushdown automaton augmented by an additional $O(\log n)$ bits of work tape. This model was introduced by Cook in [5].

It is clear that given a deterministic finite automaton, computing the coefficients of its language's generating series is logspace reducible to ranking the language. Let L be the language with an alphabet of size m and let a be the highest ranking symbol. In order to compute the n-th coefficient for L, take a^n and compute its rank w.r.t. L and its complement $\bar{L}$. If $w \in L$, then its rank is the n-th coefficient, otherwise we have computed the n-th coefficient c_n of $\bar{L}$ so that the n-th coefficient of L is simply $m^n - c_n$. Despite this, Huynh's result on regular language ranking is not directly comparable with the approach used in our motivating example involving rightlinear grammars because there we did not require assumptions on the grammar. Note that Huynh's second result shows among other things that unambiguous CFL can be ranked in NC2.

The Kuich–Salomaa Algorithm

It is known [ ] that the generating series of each nonterminal of a CFG is algebraic. This is not the same thing as saying that the generating series of the language of G is algebraic. Let L be the context-free language generated by a CFG G. The generating series F of L itself is defined to be

$$F = \sum_{n=0}^{\infty} F(n) \cdot t^n,$$

where $F(n)$ is the number of length n words in L. The generating series of the initial nonterminal of G need not coincide with F, in fact it does so only when G is unambiguous. Indeed, if F is transcendental, then it cannot be generated by any unambiguous CFG and so L must be an inherently ambiguous language.

A very concrete demonstration that G-generating series are algebraic is achieved by the next result due to Kuich and Salomaa [16]. Before discussing this result (Kuich–Salomaa elimination algorithm) we remind the reader about the fundamental idea of elimination. Let $P_1, \ldots, P_h$ be polynomials with rational coefficients in the complex variables $t_1, \ldots, t_q, x_1, \ldots, x_h$. Assume that in some neighborhood of $t_1 = 0, \ldots, t_q = 0$ there are unique power series $F_1, \ldots, F_h$ in $t_1, \ldots, t_q$ such that the equations $P_1 = 0, \ldots, P_h = 0$ are satisfied when F_i is substituted for $x_i, i = 1, \ldots, h$. In this situation the Kuich–Salomaa algorithm always produces a rational polynomial P in the variables $t_1, \ldots, t_q, x_1$ such that in the same neighborhood $\hat{F}_1$ is the unique solution to $P(t_1, \ldots, t_h, F_1) = 0$.

There is also a more general notion of elimination. See either [7, 14] for details.

The assumption about the uniqueness of power series solutions to a system of equations is satisfied in case the equations are obtained from a context-free grammar where the terminals become the variables $t_1, \ldots, t_q$ and $x_1, \ldots, x_h$ represent the nonterminals. See Chapter 16 of [16].

THEOREM 3.1 (Salomaa and Kuich). *Let $F_1, \ldots, F_r$ be the solution of a CFG of size σ. For the initial nonterminal x_1, a rational polynomial $P(a_1, \ldots, a_s, x_i)$ can be produced in $2^{2^{O(\sigma)}}$ time such that*

- *P is irreducible over $\mathbf{Q}$.*
- *$z = F_1$ is the unique solution to*

$$P(a_1, \ldots, a_s, z) = 0$$

in some neighborhood of $a_1 = \cdots = a_s = 0$, where the a_j are understood to be complex variables.

Proof. We refer the reader to pp. 346–350 of [16] for the proofs of items 1 and 2. We will sketch a verification of the time bound.

We outline the steps in the Kuich–Salomaa algorithm and give upper bounds on their running times. Recall that r is the number of nonterminals, which satisfies $r = O(\sigma)$, where σ is the size of the grammar. This follows since σ can be taken to be the sum of the lengths of all productions. Every nonterminal in a Chomsky normal form CFG can be assumed to be involved in the left-hand size of at least one production.

There are $2r + 1$ main steps in the Kuich–Salomaa algorithm. G will be written as a system of r equations, one for each nonterminal. The equation for x_i has the form $x_i - Q_i = 0$, where Q_i is an integer coefficient polynomial in possibly all of the symbols in $X \cup A$. These symbols are regarded as complex variables.

At each step one will have a system of $O(\sigma)$ polynomial equations in $O(\sigma)$ variables. There are three main operations performed in these steps.

- Resultants of polynomials are formed in $O(\sigma)$ steps. If R and S are k-variate polynomials with integer coefficients of size b bits and degree c and d, in one of the variables, say z, respectively, their resultant with respect to z is a polynomial in the remaining variables (hence a rational in the 1-variate case) which is the determinant of the Sylvester matrix. This matrix has the form

$$\begin{pmatrix} R_0 & R_1 & \cdot & \cdot & \cdot & R_c & 0 & 0 & \cdot & 0 \\ 0 & R_0 & R_1 & \cdot & \cdot & \cdot & R_c & 0 & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & \cdot & \cdot & R_0 & R_1 & \cdot & \cdot & \cdot & R_c \\ S_0 & S_1 & \cdot & \cdot & \cdot & S_d & 0 & 0 & \cdot & 0 \\ 0 & S_0 & S_1 & \cdot & \cdot & \cdot & S_d & 0 & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & \cdot & S_0 & S_1 & \cdot & \cdot & \cdot & \cdot & S_d \end{pmatrix},$$

where R_i is the coefficient of z^{c-i} and S_i is the coefficient of z^{d-i}.

If g is the maximum degree of any of the remaining variables in either R or S, the resultant will be a polynomial whose maximum degree is at most $g \cdot (c + d)$ and with coefficients having a bit size of at most $O(b \cdot (c + d))$. In our case we can overestimate b, c, d, g at the initial step by σ. This means that after $O(\sigma)$ steps, in the worst case, we will produce a polynomial whose coefficients have a bit size of

$$O\left(\sigma^{2^{O(\sigma)}}\right) = 2^{2^{O(\sigma)}},$$

with a similar upper bound on the maximum degree of any remaining variable.

- Computing the GCD (greatest common divisor) of a list of polynomials. This computation can be reduced to the computation of pairs of polynomial GCDs. That is, $\mathrm{GCD}(p, q, w) = \mathrm{GCD}(\mathrm{GCD}(p, q), w)$, etc. The time complexity analysis of the GCD of a pair of multivariate polynomials is somewhat involved, but [8] reports an upper bound of $O(m^{2r+1} \cdot d^3)$, where m is the maximum

degree of any variable, r is the number of variables and d is the maximum bit-size of any coefficient. The GCD polynomial cannot have higher degree or larger coefficients than the polynomials it divides, so there is no explosion as with resultants. At each step, GCD calculation is dominated (pessimistically) by

$$O\left(\left(\sigma^{2^{O(\sigma)}}\right)^{2\sigma+1} \cdot 2^{3\cdot 2^{O(\sigma)}}\right) = 2^{2^{O(\sigma)}}.$$

The overall time for GCD is the same kind of expression since multiplying by the number of steps $O(\sigma)$ has no effect on the upper bound.

- Factoring a polynomial into irreducible factors. Kaltofen [15] has shown that multivariate rational polynomial factorisation can be done in time polynomial in the size of a polynomial in the dense representation. In this representation, all terms, even those with zero coefficients are written out. In our case we get an upper bound of

$$2^{2^{O(\sigma)}}. \qquad \blacksquare$$

A rational polynomial P for which F_1 is the unique solution to $P(a_1, \ldots, a_s, F_1) = 0$ in a neighborhood of $a_1 = \cdots = a_s = 0$ will be called a defining polynomial for F_1.

One could also use elimination via Gröbner bases to obtain a defining polynomial. The resulting system of equations would not involve any nonterminal except x_1. By squaring these equations and adding them together, one obtains a defining polynomial for x_1. A discussion of elimination theory via Gröbner bases is given in [7]. However, the complexity of this approach appears also to be doubly exponential in the grammar size. See [19].

Furstenberg's Theorem

The core of our method is an explicit representation of the generating series for the initial nonterminal $\hat{F}_1$ as the diagonal of a bivariate rational series that can be written in closed form as a rational function in terms of the polynomial P produced by the Kuich–Salomaa algorithm. The diagonal $\mathcal{D}(f)$ of any multivariate series f is the sum of all terms of f in which all symbols occur to the same power. We give the proof by Furstenberg [10] because we will need to use it later on.

THEOREM 3.2. *If $P(x, y)$ is a polynomial in x, y over any field* **K** *such that the following all hold*:

- $P(0, 0) = 0$
- $\partial P(0, 0)/\partial y \neq 0$
- $P(x, f) = 0$, *where* $f \in K[[x]]$, *and* $f(0) = 0$

then f can be written as the diagonal of the series expansion of a rational function as follows:

$$f = \mathcal{D}\left(\frac{1}{P(xy, y)} \frac{y^2 \partial P(xy, y)}{\partial y}\right).$$

Proof. We can write $P(x, y)$ as

$$P(x, y) = (y - f(x)) \cdot Q(x, y), \tag{1}$$

where $Q(x, y)$ is a polynomial in y with coefficients in $\mathbf{K}[[x]]$. Eq. 1, $\frac{\partial P(0,0)}{\partial y} \neq 0$ and $f(0) = 0$ imply that $Q(0, 0) \neq 0$. We also have from Eq. 1 that

$$\frac{1}{P(x, y)} \frac{\partial P(x, y)}{\partial_y} = \frac{1}{y - f(x)} + \frac{1}{Q(x, y)} \frac{\partial Q(x, y)}{\partial y}. \tag{2}$$

Note that by expanding in a geometric series

$$\frac{y^2}{y - f(xy)} = \sum_{n=0}^{\infty} y^{-n+1} \cdot (f(xy))^n. \tag{3}$$

Also, since $Q(0,0) \neq 0$, we can expand $1/Q(xy, y)$ as a power series in x and y and it follows that

$$\mathcal{D}\left(\frac{y^2}{Q(xy,y)} \frac{\partial Q(xy,y)}{\partial y}\right) = 0. \tag{4}$$

Eq. 2, Eq. 3 and Eq. 4 imply that

$$\mathcal{D}\left(\frac{1}{P(xy,y)} \frac{y^2 \partial P(xy,y)}{\partial y}\right) = \mathcal{D}\left(\sum_{n=0}^{\infty} y^{-n+1} (f(xy))^n\right).$$

However, it is clear that

$$\mathcal{D}\left(\sum_{n=0}^{\infty} y^{-n+1} (f(xy))^n\right) = f(xy),$$

and the theorem is proved. ∎

Define $A(x, y)$ by

$$A(x,y) = \frac{y^2 \partial P(x,y)}{\partial y}$$

The next technical lemma is the key to our algorithm.

LEMMA 3.1. *Let $P(x, y) \in \mathbf{K}[x, y]$ satisfy the conditions of Theorem 3.2. Let $f \in \mathbf{K}[[x]]$ be as in Theorem 3.2. If $a \neq 0$, then define $P_a(x)$ by*

$$P(xa, a) = bx^k(1 - P_a(x)), \tag{5}$$

such that $P_a(0) = 0$. Let d be the maximum of the x and y degrees of $P(x,y)$. We can then write

$$\frac{A(xa,a)}{bx^k} \sum_{h=0}^{n} (P_a(x))^h = \sum_{i=1}^{dn} c_{a,i} x^i,$$

such that $c_{a,i} - c_i(a)$ where $c_i \in \mathbf{K}[1/y, y]$ is a Laurent polynomial, and $[y^n]c_n = [x^n]f$.

Proof. Equations 2, 3, and 4 show that

$$\frac{A(xy,y)}{P(xy,y)}$$

is a power series in x such that

$$[x^i]\frac{A(xy,y)}{P(xy,y)}$$

is in $\mathbf{K}[1/y, y]$, the y degree is at most di, and the $1/y$ degree is at most $i - 1$. Letting

$$c_n = [x^i]\frac{A(xy,y)}{P(xy,y)} \tag{6}$$

it is clear from Theorem 3.2 that

$$f = \mathcal{D}\left(\frac{A(xy, y)}{P(xy, y)}\right) = \sum_{n=0}^{\infty} [y^n] c_n \cdot x^n.$$

We can expand

$$\frac{1}{P(xa, a)} = \frac{1}{bx^k(1 - P_a(x))}$$

in a geometric series obtaining

$$\frac{1}{P(xa, a)} = \frac{1}{bx^k} \sum_{h=0}^{\infty} (P_a(x))^h \tag{7}$$

Using Eq. 6 and Eq. 7, we have

$$\frac{A(xa, a)}{P(xa, a)} = \sum_{i=0}^{\infty} c_{a,i} x^i = \frac{A(xa, a)}{bx^k} \sum_{h=0}^{\infty} (P_a(x))^h$$

with $c_{a,i}$ as required. ■

The Algorithm

In order to apply Furstenberg's theorem via Lemma 3.1, we need to check that

$$\frac{\partial P(0, 0)}{\partial y} \neq 0$$

where $P(x, y)$ is the output polynomial of the Kuich–Salomaa algorithm. In the next lemma we asume that some grammar and one of its nonterminals, x_i have been given.

LEMMA 3.2. *If $P(x, y)$ is produced by the Kuich–Salomaa algorithm, such that P is the defining polynomial for $\hat{F}_1$, then $\frac{\partial P(0,0)}{\partial y} \neq 0$.*

Proof. We treat x and y as ordinary complex variables. In some neighborhood of $x = 0$, we can identify y with the power series $\hat{F}_1$. In particular, since $\hat{F}_1$ is algebraic, it certainly has a nonzero radius of convergence. This follows directly from the upper bound estimate on the asymptotic growth rate of the coefficients. See [9]. In fact, this radius can be obtained easily in the case of interest corresponding to a CFG. If there are m terminals and r nonterminals, any derivation of a length n terminal string can involve at most $2n$ productions since the CFG is in Chomsky normal form. This yields an upper bound of $m^n \cdot r^{2n}$ for the size of the n-th coefficient in the generating series of the CFG. In turn this implies that if $|x| < \frac{1}{m \cdot r^2}$, $\hat{F}_1$ will converge.

It follows from the proof of the implicit function theorem [22] that if $\frac{\partial P(0,0)}{\partial y} = 0$, y cannot be a differentiable function of x at $x = 0$. This contradicts the fact that y is uniquely determined to be $\hat{F}_1$ in a neighborhood of $x = 0$, and so must be differentiable at $x = 0$. ■

Next, we describe our algorithm. The algorithm has four main steps. A context-free grammar and a positive integer n are the inputs. The grammar is treated as fixed while n may vary.

1. Apply the Kuich–Salomaa algorithm to obtain a defining polynomial $P(t, x_1)$ for $\hat{F}_1$. Let d be the maximum of the x and y degrees in $P(x, y)$.

2. Let $N = (d + 1)n$ and let ϵ be a primitive N-th complex root of unity. For $j = 0, 1, \ldots, N-1$, compute $P_{\epsilon^j}(x)$. The reader may wish to recall the definition of P_a in Eq. 5.

3. For $j = 0, 1, \ldots, N-1$, compute the expansion (Lemma 3.1)

$$\frac{A(x\epsilon^j, \epsilon^j)}{bx^k} \sum_{h=0}^{n} (P_{\epsilon^j}(x))^h = \sum_{i=0}^{dn} c_{\epsilon^j,i} x^i$$

4. Compute $[x^n]\hat{F}_1$ from the data: $c_{\epsilon^j,n}$, for $j = 0, 1, \ldots, N-1$. This can be done by standard polynomial interpolation. Note that $[x^0]\hat{F}_1 = 0$ because the grammar does not generate λ. This implies $P(0,0) = 0$. This and Lemma 3.2 imply that the conditions of Lemma 3.1 are satisfied so that $[x^n]\hat{F}_1 = [y^n]c_n$. Now, Lemma 3.1 also shows that $y^{-n+1}C_n$ is a rational polynomial of degree at most $dn + n - 1$. The Fast Fourier transform can be used to obtain $[y^n]c_n = [x^n]\hat{F}_1$ from the data.

THEOREM 3.3. *The coefficients of context-free grammar generating series can be computed in DIV in terms of* $2^{2^{O(\sigma)}} \cdot n$, *where n is the coefficient index and* σ *is the grammar size.*

Proof. It has already been observed in Step 4 that $[x^n]\hat{F}_1$ is computed by the algorithm. It remains to check the time complexity.

Let σ be the grammar size. By Theorem 3.2, Step 1 requires at most $2^{2^{O(\sigma)}}$ time. This means that the degree of the defining polynomial computed in Step 1 has the same upper bound. This means that N, which appears in the remaining steps satisfies

$$N = O\left(2^{2^{O(\sigma)}} \cdot n\right).$$

Steps 2, 3 and 4, including rational approximations for the ϵ^j all involve at worst $N^{O(1)}$-fold additions and multiplications of integers whose bit size is initially $N^{O(1)}$. Each of these computations can be carried out in DIV in terms of N, and the number of processors is needed overall is clearly $N^{O(1)}$. Note that the fast fourier transform can be carried out by multiplication of an $N \times N$ matrix and $N \times 1$ vector. This can actually be carried out in NC1 in terms of N. ∎

4. AN OBSTRUCTION TO EFFICIENT USE OF ELIMINATION THEORY

We address the influence of grammar size σ on computation of G-generating series co-efficients. We will show in theorem 4.1 that there are grammars for which any polynomial defining $\hat{F}_1$ must have degree exponential in r, the number of nonterminals. This strongly suggests that any direct use of elimination theory in computing CFG generating series will require $2^{\Omega(r)}$ time. For a CFG in Chomsky normal form, $r = \Omega(\sqrt{\sigma})$.

We require an algebraic lemma first.

LEMMA 4.1. *Let* $g(t) = g^{(1)}(t) = \sqrt{1-4t}$, *and for any integer* $k > 1$, *let* $g^{(k+1)} = g(g^{(k)}(t))$. *If* $\gamma = 1/p$ *such that p is an odd prime number, then for any positive integer k,* $\mathbf{Q}[\sqrt{-1}, g^{(k)}(\gamma)]$ *has degree* 2 *over* $\mathbf{Q}[\sqrt{-1}, g^{(k-1)}(\gamma)]$.

Proof. Let $\gamma_i = g^{(i)}(\gamma_0)$, where $\gamma_0 = \gamma$ and let $\iota = \sqrt{-1}$.

First we show that for any complex value β either $\mathbf{Q}[\iota, g(\beta)]$ has degree 2 over $\mathbf{Q}[\iota, \beta]$ or $g(\beta) \in \mathbf{Q}[\iota, \beta]$. If $P(g(\beta)) = 0$ where P is a rational polynomial which has terms of both odd and even degree, then already $g(\beta) \in \mathbf{Q}[\beta]$. To see this write $P(g(\beta)) = R(g(\beta)) + S(g(\beta))$, where R has even degree terms and S has odd degree terms. Let $R'(\beta) = R(g(\beta))$, and $S(g(\beta)) = g(\beta)S'(\beta)$, where R' and S' are rational polynomials. This means that

$$(g(\beta)) = -\frac{R'(\beta)}{S'(\beta)}$$

implying that $g(\beta) \in \mathbf{Q}[\beta]$. If $P(g(\beta)) = 0$ and P has only even degree terms, then $P'(\beta) = P(g(\beta)) = 0$, where the degree of P' is half that of P. If P has only odd degree terms, then $P(g(\beta)) = g(\beta)P'(\beta) = 0$, where P' has less than half the degree of P.

By the previous paragraph it suffices to show that $\gamma_{i+1} \notin \mathbf{Q}[\iota, \gamma_i]$. To do this we will actually show that if $i > 0$, and $a = b/2^m$, where b is an integer, and m is nonnegative, then $\sqrt{a - \gamma_i} \notin \mathbf{Q}[\iota, \gamma_i]$. The case $\gamma_{i+1} = \sqrt{1 - 4\gamma_i}$ reduces to this via $\gamma_{i+1} = 2\sqrt{1/4 - \gamma_i}$.

For the induction step we will assume that $\sqrt{a - \gamma_i} \in \mathbf{Q}[\iota, \gamma_i]$ and obtain a contradiction. Our assumption means that

$$\sqrt{a - \gamma_i} = A\gamma_i + B$$

where A and B have lower degree than γ_i over $\mathbf{Q}[\iota]$. We get from this

$$a - \gamma_i = A^2\gamma_i^2 + 2AB\gamma_i + B^2$$

It is straightforward that

$$a = A^2\gamma_i^2 + B^2$$

and

$$2AB = -1$$

These facts lead to the quadratic equation

$$Y^2 - \frac{aY}{\gamma_i^2} + \frac{1}{4\gamma_i^2} = 0$$

where $Y = A^2$. Solving, we get

$$Y = \frac{a \pm \sqrt{a^2 - \gamma_i^2}}{2\gamma_i^2}$$

Now, we argue that $Y \notin \mathbf{Q}[\iota, \gamma_{i-1}]$, but since $Y = A^2$, we obtain a contradiction because A has lower degree than γ_i. For the expression under the radical we have

$$a^2 - \gamma_i^2 = a^2 - 1 + 4\gamma_{i-1}$$

Note that

$$\frac{a^2 - 1}{4}$$

has the required form $b/2^m$. By induction hypothesis,

$$\sqrt{\frac{a^2 - 1}{4} + \gamma_{i-1}} = \iota\sqrt{\frac{1 - a^2}{4} - \gamma_{i-1}} \notin \mathbf{Q}[\iota, \gamma_{i-1}]$$

which shows that $Y \notin \mathbf{Q}[\iota, \gamma_{i-1}]$.

It remains to treat the basis of the induction. We must show that

$$\sqrt{a - \gamma_0} \notin \mathbf{Q}[\iota]$$

The only way for the above radical to be a complex rational is if $a - \gamma_0 = \pm c^2/d^2$, such that c, d are

integers. We can assume that c and d are coprime. This yields, using $a = b/2^m$ that

$$\frac{pb - 2^m}{p} = \pm\frac{2^m c^2}{d^2}$$

Since $pb - 2^m$ and p are coprime, c^2/d^2 cannot be an integer. However, this means that d^2 divides p, which is impossible. ■

THEOREM 4.1. *There are infinitely many distinct context-free grammars such that any defining polynomial for $\hat{F}_1$ has degree at least $2^{\hat{\Omega}(r)}$, where r is the number of nonterminals.*

Proof. We will construct a grammar with $A = \{t\}$ and $X = \{x_1, \ldots, x_r\}$ such that $P(t, x_1)$ satisfies the claim. Since A has only one symbol, $\hat{F}_i$ coincides with F_i, so we drop the 'hat' notation. The grammar productions, which we write as equations from the outset are as follows. For $1 \le i < r$, $x_i = x_i^2 + x_{i+1}$, and $x_r = x_r^2 + t$. If $g(w) = \sqrt{1 - 4w}$, then it is easy to see that $x_1 = g^{(r)}(t)$, where $g^{(k)}$ designates k-fold composition.

By Lemma 4.1, If p is an odd prime and $\gamma = 1/p$, then for any positive integer k, the field $\mathbf{Q}[\sqrt{-1}, g^{(k)}(\gamma)]$ has degree 2 over $\mathbf{Q}[\sqrt{-1}, g^{(k-1)}(\gamma)]$. It follows from this and basic Galois theory that $\mathbf{Q}[\sqrt{-1}, g^{(r)}(\gamma)]$ has degree 2^r over $\mathbf{Q}[\sqrt{-1}]$. Now assume that $Q(t, x_1)$ is a polynomial of degree less than 2^r such that $Q(t, F_1) = 0$. Choose p large enough so that γ is inside the radius of convergence of F_1. We would have $Q(\gamma, g^{(r)}(\gamma)) = 0$, which is impossible since $g^{(r)}(\gamma)$ has degree 2^r over $\mathbf{Q}[\sqrt{-1}]$. Note also that we do get distinct grammars for distinct r since the corresponding F_1 evaluate to distinct values at infinitely many choices for γ (the reciprocals of primes, at least). ■

5. CONCLUSION

The method for solving CFG-GEN given in this paper makes use of elimination theory to transform a system of polynomial equations obtained from a CFG into a defining polynomial for the CFG's generating series. Furstenberg's theorem is then used to essentially reduce the computation to that used for regular language generating series. The worst-case analysis of the Kuich–Salomaa elimination algorithm provides an upper bound on the size of the defining polynomial that is doubly exponential in the grammar size. Unlike the algorithm in [3] where the coefficient index n is coupled to the grammar size as n^σ, in the method of this paper the coupling has the form $2^{2^{O(\sigma)}} \cdot n$. It is an open question whether some form of elimination can deliver a defining polynomial of singly exponential size in the grammar size. By Theorem 4.1 we know that this is nearly best possible.

The CFG-GEN problem raises a general question concerning the relationship of different levels of polylogarithmic time. At present no hierarchy theorem exists for NC. That is, it is conceivable that NC = NC1. One possible strategy for exhibiting a kind of stratification within NC would be to find a problem whose data divide into a fixed parameter, e.g., a CFG of size σ, and the usual problem size, e.g., word length n, and then to show that for some k any NCk algorithm introduces σ into the circuit size or depth in a way that can be avoided in NCk' for some $k' > k$. CFG-GEN is a candidate for this kind of problem.

REFERENCES

1. Aho, A., and Ullman, J. (1972), "The Theory of Parsing, Translation and Compiling," Prentice Hall, New York.
2. Beame, P., Cook, S., and Hoover, H. (1986), Log depth circuits for division and related problems, *SIAM J. Comput.* **15**, No. 4, 994–1003.
3. Bertoni, A., Goldwurm, M., and Massazza, P. (1990), Counting problems and algebraic formal power series in noncommuting variables, *Inform. Process. Lett.* **34**, 117–121.
4. Borodin, A. (1977), On relating time and space to size and depth, *SIAM J. Comput.* **6**, 733–744.
5. Cook, S. (1971), Characterizations of pushdown machines in terms of time-bounded computers, *J. Assoc. Comput. Mech.* **18**, No. 1, 4–18.
6. Cook, S. (1985), A taxonomy of problems with fast parallel algorithms, *Inform. and Control* **64**, 2–22.
7. Cox, D., Little, J., and O'Shea, D. (1992), "Ideals, Varieties and Algorithms," Springer-Verlag, Berlin/New York.

8. Davenport, J. H., Siret, Y., and Tournier, E. (1988), "Computer Algebra," Academic Press, San Diego.
9. Flajolet, P. (1987), Analytic models and ambiguity of contextfree languages, *Theoret. Comput. Sci.* **49**, 282–309.
10. Furstenberg, H. (1967), Algebraic functions over finite fields, *J. Algebra* **7**, 271–277.
11. Hopcroft, J., and Ullman, J. (1979), "Introduction to Automata Theory, Languages and Computation," Addison–Wesley, Reading, MA.
12. Huynh, D. (1991), Effective entropies and data compression, *Inform. and Comput.* **90**, 67–85.
13. Huynh, D. T. (1990), The complexity of ranking simple languages, *Math. Systems. Theory* **23**, 1–19.
14. Jacobson, N. (1974), "Basic Algebra I," Freeman, New York.
15. Kaltofen, E. (1985), Polynomial time reductions from multivariate to bi- and univariate integral polynomial factorization, *SIAM J. Comput.* 469–489.
16. Kuich, W., and Salomaa, A. (1986), "Semirings, Automata, Languages," Springer-Verlag, Berlin/New York.
17. Litow, B. (1992), On iterated integer product, *Inform. Process. Lett.* **42**, No. 5, 269–272.
18. Litow, B. (1996), Bounded length ucfg equivalence, *in* "Proc. 7th Intl. Symp. on Algorithms and Computation," pp. 239–246, Springer-Verlag, Berlin/New York.
19. Mayr, E., and Meyer, A. (1982), The complexity of the word problem for commutative semigroups and polynomial ideals, *Adv. in Math.* **46**, 305–329.
20. Reif, J. (1986), Logarithmic depth circuits for algebraic functions, *SIAM J. Comput.* **15**, 231–242.
21. Ruzzo, W. (1981), On uniform circuit complexity, *J. Comput. System Sci.* **22**, 365–383.
22. Spivak, M. (1965), "Calculus on Manifolds," Addison–Wesley, Reading, MA.
23. Wegener, I. (1987), "The Complexity of Boolean Functions," Wiley, New York/Teubner, Stuttgart.
24. Younger, D. H. (1967), Recognition and parsing of context-free languages in time n^3, *Inform. and Control* **10**, 189–208.