INFORMATION AND COMPUTATION **134**, 59–74 (1997)
ARTICLE NO. IC972621

A Quasi-polynomial-time Algorithm for Sampling Words from a Context-Free Language

Vivek Gore* and Mark Jerrum*

Department of Computer Science, University of Edinburgh, The King's Buildings, Edinburgh EH9 3JZ, United Kingdom

Sampath Kannan and Z. Sweedyk

Department of Computer and Information Science, University of Pennsylvania, Philadelphia, Pennsylvania 19104
E-mail: kannan@central.cis.upenn.edu, zzz@saul.cis.upenn.edu

and

Steve Mahaney

DIMACS Center, Rutgers University, Piscataway, New Jersey 08855
E-mail: mahaney@dimacs.rutgers.edu

A quasi-polynomial-time algorithm is presented for sampling almost uniformly at random from the n-slice of the language $L(G)$ generated by an arbitrary context-free grammar G. (The n-slice of a language L over an alphabet Σ is the subset $L \cap \Sigma^n$ of words of length exactly n.) The time complexity of the algorithm is $\epsilon^{-2}(n\,|G|)^{O(\log n)}$ where the parameter ϵ bounds the variation of the output distribution from uniform, and $|G|$ is a natural measure of the size of grammar G. The algorithm applies to a class of language sampling problems that includes slices of context-free languages as a proper subclass. For the restricted case of homogeneous languages expressed by regular expressions without Kleene-star, a truly polynomial-time algorithm is presented.

1. PROBLEM SPECIFICATION, HISTORY, AND MOTIVATION

We address the problem of sampling (almost) uniformly at random a word of length n from a context-free language L, and the related problem of *estimating* (or approximately counting) the number of words of length n in L. Ideally, we would like to obtain a sampling procedure that runs in time polynomial in the length n

* Supported by Esprit Working Group No. 7097, "RAND."

0890-5401/97 $25.00

and in the size of the grammar used to specify L. The problem of *exactly counting* the number of words of length n in a context-free language L has been considered by Hickey and Cohen [8] and Mairson [15], who have proposed efficient solutions based on dynamic programming, but always restricted to the special case of unambiguous grammars. Note that even the class of regular grammars is not included in the class of unambiguous grammars. While regular grammars are not inherently ambiguous, an equivalent unambiguous grammar to a given regular grammar could be exponentially larger in size. (See, for example, [4, 7].) No polynomial-time algorithm has been proposed for general context-free grammars or even for regular grammars.

Let G be a context-free grammar generating the language $L = L(G) \subseteq \Sigma^*$, and let n be a positive integer. The n-slice of L is the subset $L \cap \Sigma^n$ containing all words in L of length n. The problem of exactly determining the size of the n-slice of $L(G)$ is $\#$P-complete and remains so even when the grammar G is restricted to be regular. (See Section 6 for a sketch proof.) Thus determining the size of such a slice is computationally equivalent to counting the number of accepting computations of a polynomial-time nondeterministic Turing machine, or counting the number of satisfying assignments to a Boolean formula. (See Garey and Johnson [5, Section 7.3] for a description of $\#$P and its completeness class.) This completeness result does not, however, rule out the possibility of efficient *sampling* from slices of context-free languages, nor efficient *estimation* of the size of slices. As we shall see in Section 2, sampling and estimation are computationally equivalent, and we do not always distinguish between them in what follows.

Kannan, Sweedyk and Mahaney [11] presented a "quasi-polynomial-time" (i.e., with running time $\exp(\text{polylog}(|G|, n))$) algorithm for the case of a regular grammar G by defining and using an extended version of the Karp–Luby technique for sampling from a union of sets [13]. Gore and Jerrum [6] extended this result to arbitrary context free grammars G. In fact, [6] operates within the more general setting of "$\{\cup, \cdot\}$-programs" which contains context-free grammars as a special case. (See Theorem 3.1 for an exact statement of the main result.) In addition to the techniques used by [11], reference [6] also uses depth reduction of algebraic programs, a technique due to Valiant *et al.* [21].

This paper is a synthesis of the ideas and results in [11] and [6]. In addition, Section 5 contains some new material, concerning the existence of a *truly* polynomial-time sampling procedure for languages represented by restricted regular expressions.

Problems of uniform generation and exact counting of words in a regular language are motivated by several applications, mainly in computational biology. It is common to measure the performance of algorithms that process DNA sequences by testing on "random" sequences. It has been recently proposed [18] that DNA sequences be modelled as words in regular languages or in languages higher up in the Chomsky hierarchy such as context-free languages or indexed languages. Uniform generation of words in these languages provides a good set of test data for these algorithms.

Regular expressions are also used to describe binding sites of proteins on DNA. These expressions are then used to screen DNA sequences to discover binding sites. Therefore, the number of words generated by a regular expression is of interest in

estimating the significance of finding such a site in a DNA sequence. Biologically, counting the number of sites corresponds to making a statement about the "specificity" of a DNA-binding protein (given that the protein is governed by a regular expression). Knowing how "specific" the binding of a protein is allows, for example, for speculations as to the possible involvement of a second binding factor. Restriction sites of restriction enzymes are also modelled by regular expressions [20]. Again, information about the count of the number of words is useful in estimating sizes of the restriction fragments, etc.

Finally, being able to generate members of a context-free language uniformly allows us to generate random test programs to test parsers for programming languages.

2. COMPLEXITY OF COUNTING AND SAMPLING

Connections between problems of exact counting, approximate counting, and (almost) uniform sampling[1] were explored by Jerrum *et al.* [10]. For background knowledge we summarise some of the definitions and results from [10] that are relevant to our work.

DEFINITION 2.1. A relation $R \subseteq \Sigma^* \times \Sigma^*$ is a *p*-relation if

1. There exists a polynomial $p(n)$ such that $\langle x, y \rangle \in R \Rightarrow |y| \leqslant p(|x|)$.
2. The predicate $\langle x, y \rangle \in R$ can be tested in deterministic polynomial time.

The following natural problems can be defined for any *p*-relation R.

Decision. Given x determine if there exists a y such that $\langle x, y \rangle \in R$.

Construction. Given x find y (if it exists) such that $\langle x, y \rangle \in R$.

Exact Counting. Given x determine $N_x = |\{ y : \langle x, y \rangle \in R \}|$.

Randomised Approximate Counting (*Estimation*). Given x and an additional parameter $\varepsilon > 0$, output N_x such that, with probability at least $\frac{3}{4}$,

$$\frac{N_x}{(1+\varepsilon)} \leqslant \hat{N}_x \leqslant N_x(1+\varepsilon).$$

Almost Uniform Sampling. Given x and an additional parameter $\varepsilon > 0$, output y such that (a) $\langle x, y \rangle \in R$, and (b) for any y satisfying $\langle x, y \rangle \in R$,

$$\frac{1}{N_x(1+\varepsilon)} \leqslant \Pr[\, y \text{ is output} \,] \leqslant \frac{(1+\varepsilon)}{N_x}.$$

The following relationships amongst these problems are observed in [10].

[1] Note that in that article the word "generation" is used for what we call "sampling."

1. There are p-relations for which exact counting is $\#\mathrm{P}$-complete, but randomised approximate counting is possible in polynomial time.

2. Under the assumptions $\mathrm{P} \neq \mathrm{RP}$, there are p-relations for which construction is possible in polynomial time, but almost uniform sampling is not.

3. The problems of almost uniform generation and randomised approximate counting are polynomially equivalent for any "self-reducible" p-relation.

Self-reducibility of p-relations has a rather technical definition, but intuitively says that the solution to a particular instance can be expressed in terms of solutions to smaller instances. The p-relation we are concerned with in this paper is the relation $R = \langle x, y \rangle$, where x ranges over (G, n) with G a context-free grammar and n a positive integer in unary, and y is a word of length n that is generated by the grammar G. Like most other natural p-relations, this one is self-reducible (although under a nonstandard encoding). Thus fact (3) above implies that the almost uniform sampling and randomised approximate counting problems are of equivalent complexity for context-free languages. Note that, according to (1) and (2), the complexity of these two problems is not constrained either by the $\#\mathrm{P}$-completeness of the exact counting problem or the feasibility of the construction problem.

3. LANGUAGES, POLYNOMIALS, AND PROGRAMS

We do not deal directly in this article with slices of context-free languages, but instead work with the wider class of languages that can be computed by programs in which the elementary steps are union and concatenation of languages. It is necessary to formalise what we mean by "program" in this context.

Let $\Delta = (\Omega, I, O)$ be an algebra on underlying set Ω, with a distinguished subset $I \subset \Omega$ of primitive elements or *inputs*, and with operators O. A *Δ-program* is a sequence $\Pi = (u_i \in \Omega : 0 \leqslant i \leqslant C(\Pi) - 1)$ such that, for all $0 \leqslant i \leqslant C(\Pi) - 1$, either $u_i \in I$, or $u_i = u_j \circ u_k$, where $0 \leqslant j, k < i$ and $\circ \in O$. The *size* of Π is $C = C(\Pi)$. The *level* of u_i is defined inductively: if $u_i \in I$ then the level of u_i is 0; if $u_i = u_j \circ u_k$ then the level of u_i is the maximum level of u_j and u_k plus 1. The *depth* $\Lambda = \Lambda(\Pi)$ of Π is the maximum level of any u_i. We say that Π computes $f \in \Omega$ if $f = u_i$ for some i.

In this note, we deal in particular with $\{\cup, \cdot\}$-programs and $\{+, \times\}$-programs. In the case of $\{\cup, \cdot\}$-programs the algebra is $\Delta = (2^{\Sigma^*}, \Sigma, \{\cup, \cdot\})$, where Σ is a finite alphabet, 2^{Σ^*} is the set of all languages over Σ, and the operators $\cup$ and $\cdot$ are union and concatenation of languages, which are familiar from the definition of "regular expression." (Note that we confuse the symbol $\sigma \in \Sigma$ with the singleton language $\{\sigma\}$.) In the case of $\{+, \times\}$-programs the algebra is $\Delta = ([X], X, \{+, \times\})$, where X is a finite set of indeterminates, $[X]$ is the ring of polynomials with integer coefficients, and $+$ and $\times$ are usual addition and multiplication. The *degree* $\deg u_i$ of u_i in $\{+, \times\}$-program Π is defined inductively: if $u_i \in X$ then $\deg u_i$ is 1; if $u_i = u_j + u_k$ then $\deg u_i = \max\{u_j, u_k\}$; if $u_i = u_j \times u_k$ then $\deg u_i = \deg u_j + \deg u_k$. The program Π is *homogeneous* if $\deg u_j = \deg u_k$ whenever $u_i = u_j + u_k$ for some i. Define degree and homogeneous analogously for $\{\cup, \cdot\}$-programs, with $\cup$ replacing $+$, and $\cdot$ replacing $\times$. Since all words in a slice of

a language have the same length, programs computing these languages can be homogeneous, hence the motivation for looking at homogeneous programs.

The above definitions are standard, but we find it convenient to generalise the notion of $\{\cup, \cdot\}$-program somewhat, so that arbitrary unions of previously computed values may be formed in a single operation, though concatenation is still restricted to pairs of values. We say that the union operation has unbounded *fan-in*, while the concatenation operation has fan-in 2. Similarly, our $\{+, \times\}$-programs will typically have addition operations with unbounded fan-in, and multiplication operations of fan-in 2.

Following [10], we make the following definitions:

DEFINITION 3.1. An almost uniform sampler (for languages) is a randomised procedure that takes as input a $\{\cup, \cdot\}$-program for (or other description of) a language $L \subseteq \Sigma^*$, and a tolerance $\varepsilon > 0$, and produces as output a word $Y \in L$ (a random variable) such that $(1+\varepsilon)^{-1} |L|^{-1} \leqslant \Pr(Y = y) \leqslant (1+\varepsilon) |L|^{-1}$ for all $y \in L$.

DEFINITION 3.2. A randomized approximation scheme (for languages) is a randomised procedure that takes input as above, and produces as output a number $\hat{L}$ (a random variable) such that $(1+\varepsilon)^{-1} |L| \leqslant \hat{L} \leqslant (1+\varepsilon) |L|$ with probability at least $\frac{3}{4}$.

Our main result may now be stated.

THEOREM 3.1. *There is an almost uniform sampler of time complexity* $\varepsilon^{-2} \times (nC)^{O(\log n)}$ *for producing samples from a language* $L \subseteq \Sigma^n$, *where* L *is presented as a* $\{\cup, \cdot\}$*-program* Π *of size* C. *There is also a randomised approximation scheme with the same input specification and time complexity for estimating the size of the language. The union operations in program* Π *may have unbounded fan-in, but the concatenation operations must all have fan-in two.*

The remainder of this section presents a proof of this theorem. The proof rests on two ideas: Monte-Carlo sampling from a union of sets, and depth compression of circuits. We examine these in turn.

Karp and Luby [12, 13] introduced a simple and elegant approach to estimating the size of—and, as a by-product, uniformly sampling from—the union of a collection of finite sets, all of whose sizes are known in advance. A direct application of their technique yields a uniform sampling procedure for languages computed by $\{\cup, \cdot\}$-programs Π with two alternations of union and concatenation: thus Π consists of a block of unions, followed by a block of concatenations, followed by another block of unions. We will prove the following iterated version of the Karp–Luby method for obtaining almost uniform sampling procedure for languages computed by general $\{\cup, \cdot\}$-programs.

THEOREM 3.2. *There is an almost uniform sampler of time complexity* $O(\varepsilon^{-2} n C^{\Lambda+3} \log C)$ *for producing samples from a language* $L \subseteq \Omega^n$, *where* L *is presented as a homogeneous* $\{\cup, \cdot\}$*-program* Π *of size* C *and depth* Λ. *There is also a randomised approximation scheme with the same input specification and time complexity for estimating the size of the language. The union operations in program* Π *may have unbounded fan-in, but the concatenation operations must all have fan-in* 2.

A proof of this theorem is presented in the next section. Define a *monomial* over a set X of indeterminates to be a product of elements from X. (Thus a polynomial is a weighted sum of monomials.) Also define the *support* supp p of polynomial $p \in [X]$ to be the set of all monomials occurring in p. We say that a polynomial is *homogeneous* if every monomial in its support has the same degree and *multilinear* if the degree of each indeterminate in each monomial in the support is at most 1.

COROLLARY 3.3. *There is an almost uniform sampler that has time complexity* $O(\varepsilon^{-2} n C^{\Lambda+3} \log C)$ *for producing samples from the support* supp p *of a homogeneous multilinear polynomial* p *of degree* n, *where* p *is presented as a* $\{+, \times\}$*-program* Π *of size* C *and depth* Λ. *There is also a randomized approximation scheme with the same input specification and time complexity for estimating the size of the support. The addition operations in program* Π *may have unbounded fan-in, but the multiplications must all have fan-in* 2.

Proof. Let $\Pi = (v_0, ..., v_{C-1})$ be a $\{+, \times\}$-program computing a homogeneous, multilinear polynomial p. Assume that Π is minimal, so that if any v_i is deleted from the sequence then Π is no longer a program computing p. By induction on decreasing i, every v_i is multilinear and homogeneous. (Thus any program π computing a homogeneous, multilinear polynomial can be transformed into a minimal program computing the same polynomial, in which all polynomials computed intermediately are also homogeneous and multilinear.) These facts imply that $|\text{supp } v_i| = |\text{supp } v_j| \times |\text{supp } v_k|$ whenever $v_i = v_j \times v_k$. Moreover, it is clear that $\text{supp } v_i = \text{supp } v_j \cup \text{supp } v_k$ whenever $v_i = v_j + v_k$. The proof of Theorem 3.2 is based on just these two properties of the two operations involved and hence can be adapted to prove the corollary. ∎

Observe that the time-complexity of the almost uniform sampler guaranteed by Theorem 3.2 increases rapidly with the depth of the program. Unfortunately, a $\{\cup, \cdot\}$-program derived directly from a context-free grammar will in general have large depth, since the grammar may allow very unbalanced parse trees. Before applying Theorem 3.2, then, it is necessary to compress the program so that it has relatively small depth. The technology for achieving this compression in the case of arithmetic programs has been available for over a decade. In a concise and beautiful note, Valiant *et al.* [21] showed that if p is a polynomial of small degree that is computed by a arithmetic program of small size, then p can be computed by an arithmetic program that is simultaneously of small size and small depth. The following is a special case of their result.

PROPOSITION 3.4. *Suppose there is a homogeneous* $\{+, \times\}$*-program of size* C *that computes a polynomial* p *of degree* d. *Then there is a homogeneous* $\{+, \times\}$*-program* (*with unbounded fan-in additions*) *of size* $O(C^2)$ *and depth* $O(\log d)$ *that computes* p.

Proof. See [21]. The result there is stated for Δ-programs, where $\Delta = (K[X], X \cup K, \{+, \times\})$ and K is a field. However, it is easily checked that the proof does not introduce any new scalars (i.e., elements of K). Note that in [21] the addition operations are assumed to be binary; allowing unbounded fan-in additions reduces

the size of the compressed circuit by a factor of C and the depth by a factor of $\log C$, since the balanced binary trees of additions may be replaced by single summations. ▮

The construction of Valiant *et al.* does not directly apply to $\{\cup, \cdot\}$-programs, as it relies on commutativity of multiplication. We circumvent this problem by establishing a connection between $\{\cup, \cdot\}$-programs and $\{+, \times\}$-programs which involves encoding languages as polynomials. A similar idea was used in [1].

Let Σ be a finite alphabet and n a natural number. Introduce a set $X = X(\Sigma, n) = \{x_{t\sigma}: 0 \leqslant t \leqslant n-1 \text{ and } \sigma \in \Sigma\}$ of indeterminates, and define the mapping $\pi: \Sigma^n \to [X]$ by $\pi(\alpha) = \prod_{t=0}^{n-1} x_{ta_t}$, for all words $\alpha = a_0, ..., a_{n-1} \in \Sigma^n$. Extend the domain of π to the set of all languages $L \subseteq \Sigma^n$ by defining $\pi(L) = \sum_{\alpha \in L} \pi(\alpha)$. It is possible to simulate a $\{\cup, \cdot\}$-program computing a language $L \in \Sigma^n$ by a $\{+, \times\}$-program computing a polynomial of degree n over $X = X(\Sigma, n)$.

LEMMA 3.5. *Suppose there is a $\{\cup, \cdot\}$-program of size C computing language $L \subseteq \Sigma^n$. Then there is a homogeneous $\{+, \times\}$-program of size Nc and degree n computing a polynomial $p \in [X]$ such that* $\operatorname{supp} p = \operatorname{supp} \pi(L)$.

Proof. Let the $\{\cup, \cdot\}$-program in the statement of the lemma be $\Pi = (u_0, ..., u_{C-1})$. Assume that Π is minimal, so that if any u_i is deleted from the sequence then Π is no longer a program computing L. Then, by induction on decreasing i, the program Π is homogeneous. For each $0 \leqslant i \leqslant C-1$ and $0 \leqslant r \leqslant n - \deg u_i$, define $v_i^{(r)} \in [X]$ as follows:

(i) if $u_i = \{\sigma\}$ with $\sigma \in \Sigma$ then $v_i^{(r)} = x_{r\sigma}$;

(ii) if $u_i = u_j \cup u_k$ then $v_i^{(r)} = v_j^{(r)} + v_k^{(r)}$;

(iii) if $u_i = u_j \cdot u_k$ then $v_i^{(r)} = v_j^{(r)} \times v_k^{(r+s)}$, where $s = \deg u_j$.

In transforming a $\{\cup, \cdot\}$-program into a $\{+, \times\}$-program we are replacing concatenation by the commutative operation of multiplication. In order to preserve the positional information present in words we introduce a basis of indeterminates $(x_{r\sigma})$ that describe not only the symbol but also the position at which the symbol occurs. Homogeneity and the structure of $\{\cup, \cdot\}$-programs allows us to write the above position-constrained $\{+, \times\}$-program statements that are equivalent to the corresponding $\{\cup, \cdot\}$-program statements.

An easy induction on i establishes that $\operatorname{supp}(v_i^{(r)}) = \operatorname{supp}(\pi^{(r)}(u_i))$, for all $v_i^{(r)}$, where $\pi^{(r)}(a_0 \cdots a_{s-1}) = \prod_{t=0}^{s-1} x_{r+t, a_t}$ and $\pi^{(r)}(L) = \sum_{\alpha \in L} \pi^{(r)}(\alpha)$. Let Π' be the $\{+, \times\}$-program obtained by arranging the polynomials $\{v_i^{(r)}\}$ in lexicographic order of the index pairs (i, r). Then Π' computes a polynomial p with $\operatorname{supp} p = \operatorname{supp}(\pi^{(0)}(L))$ and has size bounded by nC. But $\pi^{(0)}(L) = \pi(L)$. ▮

All the components of the sampling procedure are now in place. Suppose we are given a language L specified by a $\{\cup, \cdot\}$-program. The overall strategy is to simulate the $\{\cup, \cdot\}$-program using a $\{+, \times\}$-program, compress the depth of the $\{+, \times\}$-program using the method of Valiant *et al.*, and use iterated Karp–Luby sampling to obtain a word from L that is close to uniform.

COROLLARY 3.6. *Suppose there is a* $\{\cup, \cdot\}$*-program of size* C *computing a language* $L \in \Sigma^n$*. Then there is a* $\{+, \times\}$*-program of size* $O((nC)^2)$*, depth* $O(\log n)$ *and degree* n *computing a polynomial* $p \in [X]$ *such that* $\text{supp } p = \text{supp } \pi(L)$.

Proof. Apply Lemma 3.5 followed by Proposition 3.4.

We immediately have the required:

Proof of Theorem 3.1. Apply Corollary 3.6 followed by Corollary 3.3. Note that the time to construct the small depth program of Corollary 3.6 is negligible in relation to the time taken to obtain a sample from it using Theorem 3.2. ∎

The result highlighted in the abstract follows easily. Let $|G|$ denote any reasonable measure of the size of grammar G, for example the total number of symbols required to write down all the productions in the grammar. (The precise choice of encoding is immaterial to the result.)

COROLLARY 3.7. *There is an almost uniform sampler for the n-slice of a language* $L(G)$ *generated by a context-free grammar* G*, which runs in time* $\varepsilon^{-2}(n\,|G|)^{O(\log n)}$*. There is a also a randomised approximation scheme with the same time complexity.*

Proof. There is an efficient translation of G (in Chomsky normal form) into a $\{\cup, \cdot\}$-program Π which is sketched below. For each non-terminal A in the grammar G and for each length $\ell \leqslant n$, let A_ℓ denote a variable in Π computing the language consisting of all words of length ℓ generated by A. A production in G of the form $A \to BC$ is translated into statements of the form $A_\ell \to B_i C_j$ for each possible value of ℓ and each pair (i, j) such that $i + j = \ell$.

4. THE SAMPLING ALGORITHM AND ITS ANALYSIS

As promised, we now present and analyse an almost uniform sampler satisfying the conditions of Theorem 3.2.

Consider a homogeneous $\{\cup, \cdot\}$-program $\Pi = (u_i : 0 \leqslant i \leqslant C-1)$, of size C and depth Λ, computing a language $L \subseteq \Sigma^n$. Note that union operations are assumed to have unbounded (bounded only by the size C of the program) fan-in. In order to apply the Karp–Luby sampling strategy, we need to obtain estimates of the sizes of the languages computed at each u_i. These sizes are computed "bottom-up," starting at u_0 and working through to u_{C-1}. Note that since u_0 computes a singleton-set, size estimation and sampling are trivial for u_0. In estimating the size of u_i, the algorithm requires random samples from u_j for $0 \leqslant j \leqslant i$: these are generated "top-down" by a recursive procedure using the sizes already computed. Thus the sampling and counting procedures are inextricably linked, and we describe them both together.

DEFINITION 4.1. Suppose $\hat{A}$ is an estimate[2] for the size of a set A, and $\hat{D}_A$ is a distribution on A. For any $\delta, \varepsilon > 0$, we say that $\hat{A}$ is a δ-approximation to $|A|$ if

[2] Note that we always use a *hat* to indicate that the quantity in question is empirically determined and hence approximate.

$e^{-\delta}|A| \leqslant \hat{A} \leqslant e^{\delta}|A|$, and that $\hat{D}_A$ is ε-uniform if $e^{-\varepsilon}|A|^{-1} \leqslant \hat{D}_A(x) \leqslant e^{\varepsilon}|A|^{-1}$ for every x in A (i.e., if $\hat{D}_A$ is within "relative pointwise distance" ε of uniform).

The sampling/estimation algorithm is now described.

4.1. *Concatenation*

Suppose S is a set represented by a concatenation operation, where $S = A \cdot B$, and that we are about to estimate the size of S. (Note that the concatenation operation has fan-in 2.) Let $\hat{A}$ and $\hat{B}$ denote the size estimates for A and B respectively, and let $\hat{D}_A$ and $\hat{D}_B$ denote the distributions on A and B induced by the algorithm. Note that the estimates $\hat{A}$ and $\hat{B}$ will already be available by the time we come to estimate the size of S.

- To sample from S, we sample from A (denoted x), sample from B (denoted y), and output xy.
- To estimate the size of S, we simply let $\hat{S} = \hat{A}\hat{B}$.

If we assume that $\hat{A}$ and $\hat{B}$ are δ-approximations to $|A|$ and $|B|$, respectively, and that $\hat{D}_A$ and $\hat{D}_B$ are both ε-uniform, then it is easy to see that $\hat{S}$ is a 2δ-approximation to $|A| \times |B|$ and that $\hat{D}_S$ is 2ε-uniform. (For example, if both $\hat{A}$ and $\hat{B}$ are underestimates by factors of $e^{-\delta}$, then $\hat{S}$ is an underestimate by a factor of $e^{-2\delta}$. Similar analysis of extreme deviations of $\hat{S}$ and $\hat{D}_S$ yields the results.)

4.2. *Union*

Suppose S is a set represented by a union operation, so that $S = \bigcup_{A \in \mathscr{A}} A$ for some finite collection $\mathscr{A}$ of sets.[3] Note that in our application, $|\mathscr{A}| \leqslant C$. As in the case of concatenation, we let $\hat{A}$ denote the size estimate for set A, and $\hat{D}_A$ the induced distribution. Let $\hat{M} = \sum_{A \in \mathscr{A}} \hat{A}$. We assume, for all $A \in \mathscr{A}$, that $\hat{A}$ is a δ-approximation to $|A|$, and that the distribution $\hat{D}_A$ is ε-uniform.

- To sample from S, we do the following:

1. Choose a set $A \in \mathscr{A}$ with probability $\hat{A}/\hat{M}$.
2. Produce a sample x from A according to $\hat{D}_A$.
3. Denote by $m(x)$ the number of sets in $\mathscr{A}$ containing x, i.e., $m(x) = |\{A \in \mathscr{A} : A \ni x\}|$. Output x with probability $m(x)^{-1}$.

Since the above steps might not always produce an output, they are repeated until an output is produced. We call one iteration of the above steps a *trial*.

Suppose $x \in S$, and let $\mathscr{B}$ denote the collection $\{A \in \mathscr{A} : A \ni x\}$ of $m(x)$ sets in $\mathscr{A}$ containing x. Let $s(x)$ denote the success probability for x, i.e., the probability that x is output in a given trial in the sampling procedure. Then

$$s(x) = \sum_{A \in \mathscr{B}} \frac{\hat{A}}{\hat{M}} \hat{D}_A(x) \frac{1}{m(x)}.$$

[3] The collection $\mathscr{A}$ is actually a multiset, as we cannot guarantee that we detect redundant computations in the program Π in polynomial time.

It is clear that $s(x)$ is a $(2\delta+\varepsilon)$-approximation to the ideal quantity $\hat{M}^{-1}$ (i.e., the success probability for x if all empirical size estimates $\hat{A}$ were exact, and all distributions $\hat{D}_A$ were uniform). We let r denote the probability that a trial fails to produce an output. Then the probability that $x \in S$ is eventually output is given by $s(x)+s(x)\,r+s(x)\,r^2+\cdots = s(x)/(1-r)$. Note that $1-r=\sum_{y\in S} s(y)$, so $1-r$ is also a $(2\delta+\varepsilon)$-approximation to the ideal. Hence $\hat{D}_S(x)$, which is the probability that x is eventually output by the sampling procedure, is $(4\delta+2\varepsilon)$-uniform.

- To estimate $|S|$, we use the following experiment:

1. Choose a set $A \in \mathscr{A}$ with probability $\hat{A}/\hat{M}$.
2. Produce a sample x from A according to $\hat{D}_A$.
3. Output $m(x)^{-1}$. (Recall that x belongs to $m(x)$ sets in $\mathscr{A}$.)

The above experiment defines a random variable, say Z, that takes values in the set $\{i^{-1}: 1 \leqslant i \leqslant |\mathscr{A}|\}$ we estimate μ, the expected value of Z by performing t independent trials of the above experiment and computing the mean of the output values (denoted $\hat{\mu}$). We finally output $\hat{\mu}\hat{M}$ as our estimate for $|S|$.

Note that

$$\mu = \sum_{A\in\mathscr{A}} \frac{\hat{A}}{\hat{M}} \sum_{x\in A} \hat{D}_A(x)\,\frac{1}{m(x)},$$

which implies that

$$\mu\hat{M} = \sum_{A\in\mathscr{A}} \hat{A} \sum_{x\in A} \hat{D}_A(x)\,\frac{1}{m(x)}.$$

The expected value of Z is just $|S|/\sum_{A\in\mathscr{A}}|A|$ in the ideal (no error) setting, and it is clear that the value $\mu\hat{M}$ is an $(\varepsilon+\delta)$-approximation to $|S|$. By performing enough trials of the above experiment, we try to ensure that the computed value $\hat{\mu}$ is a γ-approximation to μ, for some appropriately selected γ, and hence our estimate for $|S|$ is a $(\varepsilon+\delta+\gamma)$-approximation. We use a result by Hoeffding [9] (also see Corollary 5.2 (a) in [16]) that shows that the probability that $\hat{\mu}$ does not satisfy $(1+\gamma)^{-1}\mu \leqslant \hat{\mu} \leqslant (1+\gamma)\mu$ (and, *a fortiori*, that $\hat{\mu}$ is a not a γ-approximation to μ) is bounded by $2\exp(-2t\mu^2\gamma^2)$. The least value that μ can have is $|\mathscr{A}|^{-1}$, which represents the case where all sets in $\mathscr{A}$ are equal. Hence, if we choose

$$t = \gamma^{-2}C^2 \ln C, \tag{1}$$

the above failure probability is at most $2C^{-2}$, where C, we recall, is the size of the program. For future reference, observe that, with probability at least $\frac{3}{4}$, all C size estimates made during the processing of program Π are $(\varepsilon+\delta+\gamma)$-approximations to the true values, provided $C \geqslant 3$.

4.3. *Overall Error Analysis*

Our strategy is to select the sampling-error parameter $\gamma = \gamma(l)$ to be a function of the level l in the program, and compute upper bounds on $\varepsilon = \varepsilon(l)$ and $\delta = \delta(l)$ by

induction on level. As one would expect, accuracy degrades rapidly as the level increases, so $\varepsilon(l)$ and $\delta(l)$ are exponential functions of l. Clearly, at the lowest level, we can sample as well as estimate exactly, so $\varepsilon(0) = \delta(0) = 0$. From the above discussion, for $l \geqslant 0$,

$$\varepsilon(l+1) \leqslant \max\{2\varepsilon(l),\ 2\varepsilon(l) + 4\delta(l)\}$$

and

$$\delta(l+1) \leqslant \max\{2\delta(l),\ \varepsilon(l) + \delta(l) + \gamma(l)\}.$$

Set $\gamma(l) = c4^l$, for some constant c to be chosen presently. By induction on level l, we obtain $\varepsilon(l) \leqslant c4^l$ and $\delta(l) \leqslant 2c4^l$. Setting $c = \varepsilon 4^{-(\Lambda+1)}$, where ε is the specified tolerance on sizes and distributions at the highest level Λ, we have $\varepsilon(\Lambda), \delta(\Lambda) \leqslant \varepsilon/2$. Note that for this choice of c,

$$\gamma(l) = \varepsilon 4^{-(\Lambda - l + 1)}. \tag{2}$$

Provided all the sampling errors that arise during the execution of the algorithm are within the bounds we have set—which event occurs with probability at least $\frac{3}{4}$ as we saw in the previous subsection—we shall have succeeded in computing an $(\varepsilon/2)$-approximation to $|L|$, the size of the language computed by Π; in the process, we also obtain $(\varepsilon/2)$-uniform samples from L. Thus the estimation and sampling procedure presented in this section satisfies the conditions laid down for an almost uniform sampler and for a randomised approximation scheme.

4.4. *Analysis of Running Time*

As in the case of the error analysis, we work in terms of the levels of the program. Let $T_C(l)(T_S(l))$ denote the expected time required to estimate the size of a set (produce a sample from a set) at level l, maximised over all computations at that level. Clearly, $T_C(0)$ and $T_S(0)$ are both bounded by constants .

Let us first bound $T_S(l+1)$. Producing a sample from a concatenation operation at level $l+1$, involves obtaining two samples from level l (or lower) and then concatenating them: this takes time $2T_S(l) + O(1)$. Producing a sample from a union operation involves obtaining a number of samples from lower levels and computing, for each sample x, its multiplicity $m(x)$. (A certain amount of arithmetic must also be performed, but this work is negligible in comparison.) The expected number of samples required is at most C, and the time to compute $m(x)$, by dynamic programming, is $O(Nc)$. (Recall that a program $\Pi = (u_i)$ consists of a sequence of languages u_i. For each i in turn, compute the set of subwords of x that are contained in language u_i; since the program is homogeneous, this set of subwords may be conveniently represented by a bit-vector of length n.) For each operation in the program in sequence, compute the subwords of x that are contained in it. Thus,

$$T_S(l+1) \leqslant C[\,T_S(l) + O(nC)\,]. \tag{3}$$

Solving this recurrence yields $T_S(l) = O(nC^{l+1})$.

We now bound the estimation time $T_C(l+1)$. At a concatenation operation, all that needs to be done is to multiply two previously computed values. At a union operation, we need to perform t trials of the experiment that defines the random variable Z. Each trial is dominated by the time to obtain a sample x from level l (or lower), and to compute its multiplicity $m(x)$. Thus,

$$T_C(l+1) \leqslant t[\, T_S(l) + O(Nc)\,], \tag{4}$$

which, combined with estimates (1) and (2) yields

$$T_C(l) = O(\varepsilon^{-2} n 16^{\Lambda - l} C^{l+2} \log C).$$

This expression is maximised at $l = \Lambda$, when

$$T_C(\Lambda) = O(\varepsilon^{-2} n C^{\Lambda+2} \log C)$$

The same bound holds also for $T_S(\Lambda)$. The total execution time is at most C times this, establishing Theorem 3.2.

5. FORMULAS AND REGULAR EXPRESSIONS

It is an open question whether there is a polynomial-time almost uniform sampler for languages specified by general $\{\cup, \cdot\}$-programs. However, we are able to exhibit such a sampler for the special case of $\{\cup, \cdot\}$-formulas, which correspond to regular expressions without the Kleene-star operator.

Define a *Λ-formula* to be a δ-program $\Pi = (u_i)$ in which each u_i is used at most once as an operand. Thus a $\{\cup, \cdot\}$-formula is essentially a regular expression without Kleene-star.[4] For consistency with the usual definition of regular expression, we insist that both concatenation and union operators have fan-in two. Formulas inherit the attributes *size* and *depth* and the property *homogeneous* from programs. Note that the size of a $\{\cup, \cdot\}$-formula is the number of symbols in the corresponding regular expression, discounting any parentheses.

The absence of shared subcomputations makes formulas apparently easier to deal with than programs, and we are able to improve the time complexity from "pseudo-polynomial" to truly polynomial.

THEOREM 5.1. *There is an almost uniform sampler that has time complexity $\varepsilon^{-2} C^{O(1)}$ for producing samples from a language $L \subseteq \Sigma^n$, where L is presented as a $\{\cup, \cdot\}$-formula of size C. There is also a randomized approximation scheme with the same input specification and time complexity.*

The exponent of C implicit in the O-notation is less than $13 \cdot 4$. (We do not aim here to minimise the exponent.)

As in the proof of Theorem 3.1, our approach is to transform the $\{\cup, \cdot\}$-formula into an equivalent $\{+, \times\}$-formula, with a view to applying known depth-compression techniques. For the compression step, we use, in place of Proposition 3.4,

[4] It differs only in having a defined order of evaluation.

a classical result of Brent *et al.*, [3], which allows us to work exclusively with operations of fan-in 2. For (bounded fan-in) formulas we have the following tightened version of Theorem 3.2.

THEOREM 5.2. *There is an almost uniform sampler that has time complexity* $O(\varepsilon^{-2}16^{\Lambda}C^2 \log C)$ *for producing samples from a language* $L \subseteq \Sigma^n$, *where* L *is presented as a homogeneous* $\{\cup, \cdot\}$*-formula of size* C *and depth* Λ. *There is also a randomised approximation scheme with the same input specification and time complexity*.

Proof. The improvement over Theorem 3.2 comes from restricting unions to have fan-in two. The analysis closely follows that presented in Section 4, and we indicate here only the minor changes that are necessary. In Subsection 4.2, the expected value μ of Z is now at least $\frac{1}{2}$, and hence the number t of trials (cf. Eq. (1)) may be reduced to

$$t = 4\gamma^{-2} \ln C = 4\varepsilon^{-2}16^{\Lambda - l + 1} \ln C. \tag{5}$$

For a similar reason, the expected number of samples required from level l to produce a sample at level $l+1$ is at most 2. Thus the recurrence governing $T_S(l)$ (cf. Eq. (3)) can be tightened to

$$T_S(l+1) \leqslant 2[T_S(l) + O(C)]. \tag{6}$$

Solving this recurrence yields $T_S(l) = O(C2^l)$. Note that the dynamic programming procedure for computing the multiplicity $m(x)$ now requires just a single bit for each program step instead of a bit-vector of length n, which explains the replacement of $O(nC)$ in Eq. (3) by $O(C)$ above. For a similar reason, inequality (4) may be tightened to

$$T_C(l+1) \leqslant t[T_S(l) + O(C)].$$

Substituting the revised estimates (6) and (5) for $T_S(l)$ and t into the last inequality yields

$$T_C(l) = O(\varepsilon^{-2}16^{\Lambda - l}2^l C \log C).$$

This expression is maximised at $l = 0$, when

$$T_C(0) = O(\varepsilon^{-2}16^{\Lambda} C \log C).$$

As before, the total execution time is at most C times this, verifying the claimed time complexity. ∎

COROLLARY 5.3. *There is an almost uniform sampler that has time complexity* $O(\varepsilon^{-2}16^{\Lambda - l}2^l C \log C)$ *for producing samples from the support* supp p *of a homogeneously multilinear polynomial* p *of degree* n, *where* p *is presented as a* $\{+, \times\}$*-formula of size* C *and depth* Λ.

The proof is similar to the proof of Corollary 3.3. Restated in the language of this article, the result of Brent *et al.* [3] concerning depth reduction of arithmetic formulas is as follows.

PROPOSITION 5.4. *Suppose there is a homogeneous* $\{+, \times\}$*-formula of size* C *that computes a polynomial* p*. Then there is a homogeneous* $\{+, \times\}$*-formula of size* $O(C^{1.72})$ *and depth* $2.47 \log_2 C$ *that computes* p*.*

Proof. See [3]. As with Proposition 3.4, note that the construction of Brent *et al.* introduces no new scalars. ∎

We have all the ingredients for:

Proof of Theorem 5.1. Encode the postulated $\{\cup, \cdot\}$-formula as an $\{+, \times\}$-formula as in the proof of Lemma 3.5. (Note that on this occasion there is no increase in size, as we do not need to maintain $O(n)$ "translations" of each polynomial computed.) Now apply Proposition 5.4 and Corollary 5.3. ∎

The results of this section depend crucially on the input being a homogeneous regular expression. Given an arbitrary regular expression R with associated language L, we do not know if there is a *succinct*, homogeneous regular expression that generates precisely the n-slice of L. This is to be contrasted with the fact that given a context-free grammar, NFA, or DFA for a language L, we can find (respectively) succinct context-free grammars, NFAs, or DFAs for the n-slice of L.

6. REMARKS AND OPEN PROBLEMS

The proof of the #P-completeness of counting the n-slice of a regular language follows from an easy reduction from #DNF. Given a DNF formula f with terms $t_1, ..., t_k$, we construct k deterministic finite automata where the ith automaton accepts precisely those words in $\{0, 1\}^n$ which (when viewed as truth assignments) satisfy term t_i. We then create a nondeterministic finite automaton by creating a new start state and making ε-transitions from this start state to the start states of each of the k deterministic finite automata. Note that the use of nondeterminism is essential; otherwise the size of the n-slice could be easily computed using iterated matrix multiplication. However, the construction does work if the regular language is specified by a regular expression instead of a nondeterministic finite automaton.

The indirect approach taken in this article via $\{+, \times\}$-programs appears to be necessary, as there is no possibility of compressing the depth of $\{\cup, \cdot\}$-programs themselves. Indeed Nisan [17, Theorem 4] has demonstrated an $\Omega(n)$ lower bound on the depth of $\{\cup, \cdot\}$-programs for the palindrome language $\{ww^{\mathrm{R}} : w \in \{0, 1\}^n\}$ where w^{R} denotes the reversal of w. It is worthwhile mentioning here that depth compression techniques similar to ours have been developed before in other settings involving noncommutative operations. Allender and Jiao [2] (see also [14]) have shown that depth compression is possible for arithmetic circuits for the algebra $(\Sigma^*, \Sigma, \{\max, \cdot\})$, where max denotes the operation of taking the lexicographic maximum, and denotes concatenation.

In principle the dependence of the time complexity of the sampling procedure on ε could be reduced from ε^{-2} to $\log \varepsilon^{-1}$, using Markov chain simulation techniques described by Sinclair in [19, Corollary 4.9]. However, this extra computational layer, would reduce the practicality of the method to an even lower level.

The main and obvious open question is whether a truly polynomial-time algorithm exists for sampling from the n-slice of a context-free language, or for estimating its size. (The sampling and estimation problems are of equivalent complexity, to within polynomial factors.) A superficially appealing approach is to compute samples (as well as size estimates) in bottom-up fashion, using dynamic programming. However, the nature of the information that is flowing through the dynamic programming procedure now becomes rather elusive: the size estimates $\hat{A}$ are just numbers, whereas it seems that the distributions $\hat{D}_A$ must be represented as (the code for) functions, or "objects" in more fashionable parlance. We have not been able to make this approach work nor have we been successful with other approaches such as proving rapid mixing of a suitably defined Markov chain, or finding unbiased estimators with small standard deviations.

An incidental but interesting question concerns the relative expressive powers of various representations of regular and context-free languages. For example, we do not know if regular expressions are as succinct as NFAs; in fact we do not even know that any DFA can be converted into a regular expression without a super-polynomial blow-up in size. We saw in Section 5 that sampling and size estimation problems appear to be easier given a regular expression as input as opposed to a $\{\cup, \cdot\}$-program, but we do not know whether it is computationally advantageous for the input to be presented as an NFA.

Theorem 5.1 can be strengthened slightly as follows. The condition that $L \subseteq \Sigma^n$ (i.e., that the formula specifying the language L is homogeneous) allows us to (approximately) count the number of words in a language L obtained as the concatenation of two languages L_1 and L_2 whose (approximate) sizes are known. A more complicated relaxation of homogeneity (such as a requirement that each word in L is generated in exactly one way as a concatenation of a word in L_1 and a word in L_2) would also suffice. Characterising the class of formulas for which Theorem 5.1 applies remains an open problem.

Received January 1, 1996; final manuscript received November 8, 1996

REFERENCES

1. Allender, Eric, Bruschi, Danilo, and Pighizzini, Giovanni (1993), The complexity of computing maximal word functions, *Comput. Complexity* **3**, 368–391.
2. Allender, Eric, and Jiao, Jia (1993). Depth reduction for noncommutative arithmetic circuits (extended abstract) *in* "Proceedings of the 25th ACM Symposium on Theory of Computing," pp. 515–522, Assoc. Comput. Mach., New York.
3. Brent, Richard, Kuch, David, and Maruyama, Kigoshi (1973), The parallel evaluation of arithmetic expressions without division, *IEEE Trans. Comput.* **C-22**, 532–534.
4. Bar-Hillel, V., Perles, M., and Shamir, E. (1961), On formal properties of simple phrase structure grammars, *Z. Phonet. Sprachwiss. Kommunikationsforsch.* **14**, 143–172.

5. Garey, Michael R., and Johnson, David S. (1979), "Computers and Intractability: A Guide to the Theory of NP-Completeness," p. 176, Freeman, San Francisco.
6. Gore, Vivek, and Jerrum, Mark (1995), "A Quasi-polynomial-time Algorithm for Sampling Words from a Context-Free language," Report ECS-LFCS-95-326, Department of Computer Science, University of Edinburgh.
7. Ginsburg, S., and Ullian, J. S. (1966), Preservation of unambiguity and inherent ambiguity in context-free languages, *J. Assoc. Comput. Mach.* **13**, 62–88.
8. Hickey, Timothy, and Cohen, Jacques (1983), Uniform random generation of strings in a context-free language, *SIAM J. Comput.* **12**, 645–655.
9. Hoeffding, W. (1963), Probability inequalities for sums of bounded random variables, *J. Amer. Statist. Assoc.* **58**, 13–30.
10. Jerrum, M. R., Valliant, L. G., and Vazirani, V. V. (1986), Random generation of combinatorial structures from a uniform distribution, *Theoret. Comput. Sci.* **43**, 169–188.
11. Kannan, Sampath, Sweedyk, Z., and Mahaney, Steve (1995), Counting and random generation of strings in regular languages, *in* "Proceedings of the 6th ACM–SIAM Symposium on Discrete Algorithms," pp. 551–557.
12. Karp, Richard M., and Luby, Michael (1983), Monte-Carlo algorithms for enumeration and reliability problems, *in* "Proceedings of the 24th IEEE Symposium on Foundations of Computer Science," pp. 56–64, Comput. Soc. Press.
13. Karp, Richard M., Luby, Michael, and Madras, Neal (1989), Monte-Carlo approximation algorithms for enumeration problems, *J. Algorithms* **10**, 429–448.
14. Mahajan, M., and Vinay, V. (1994), Non-commutative computation, depth reduction and skew circuits, *in* "Proceedings of the Foundations of Software Technology and Theoretical Computer Science Symposium," pp. 48–59, Springer-Verlag, Berlin/New York.
15. Mairson, Harry (1994), Generating words in a context-free language uniformly at random, *Inform. Process. Lett.* **49**, 95–99.
16. McDiarmid, Colin (1989). On the method of bounded differences, *in* "London Mathematical Society Lecture Note Series," Vol. 141, pp. 148–188, Cambridge Univ. Press, Cambridge, UK.
17. Nisan, Noam (1991), Lower bounds for non-commutative computation (extended abstract), *in* "Proceedings of the 23rd ACM Symposium on Theory of Computing," pp. 410–418, Assoc. Comput. Mach., New York.
18. Searls, D. B. (1993), The computational linguistics of biological sequences, *in* "Artificial Intelligence and Molecular Biology" (L. Hunter, Ed.), pp. 47–120, AAAI Press.
19. Sinclair, Alistair (1993), "Algorithms for Random Generation and Counting: A Markov Chain Approach," Birkhäuser, Boston.
20. Smith, R. (1988), A finite state machine algorithm for finding restriction sites and other pattern matching applications, *Comput. Appl. Biosci.* **4**, 459–465.
21. Valliant, L. G., Skyum, S., Berkowitz, S., and Rackoff, C. (1983), Fast parallel computation on polynomials using few processors, *SIAM J. Comput.* **12**, 641–644.

Printed in Belgium