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

1st July, 1995

Abstract

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 $\varepsilon^{-2}(n\,|G|)^{\mathrm{O}(\log n)}$, 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.

*The authors are supported by Esprit Working Group No. 7097, "RAND." Address for correspondence: Department of Computer Science, University of Edinburgh, The King's Buildings, Edinburgh EH9 3JZ, United Kingdom.

1 Problem specification and history

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 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 and the size of the grammar used to specify L. This problem has been considered by many authors (for example, Mairson [7]), who have proposed efficient solutions based on dynamic programming, but always restricted to the special case of unambiguous grammars. No polynomial time algorithm has been proposed for general context-free grammars.

Let G be a context-free grammar generating the language $L = L(G) \subseteq \Sigma^*$, and n a positive integer. The n-slice of L is just the subset $L \cap \Sigma^n$ containing all words in L of length n. The problem of determining the size of the n-slice of $L(G)$ is #P-complete [1], and remains so even when the grammar G is restricted to be regular. (The latter claim can be established by reduction from #DNF [3].) 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.

Kannan, Sweedyk and Mahaney [3] recently presented a "quasi-polynomial-time" (i.e., with running time $\exp(\mathrm{polylog}(|G|, n))$) algorithm for the case of a regular grammar G. In this article we extend their result to arbitrary context free grammars G. In fact, we operate within the more general setting of "$\{\cup, \cdot\}$-programs" which contains context free grammars as a special case. (See Theorem 1 for an exact statement of the main result.) The main techniques we use are Karp-Luby sampling from a union of sets [5] as extended by Kannan et al. [3], and depth reduction of algebraic programs, a technique due to Valiant, Skyum, Berkowitz and Rackoff [10].

2 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 $\Pi = (u_i \in \Omega : 0 \le i \le C - 1)$ such that, for all $0 \le i \le C - 1$, either $u_i \in I$, or $u_i = u_j \circ u_k$, where $0 \le j, k < i$ 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 $\Delta = (\mathbb{Z}[X], X, \{+, \times\})$, where X is a finite set of indeterminates, $\mathbb{Z}[X]$ 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 $\times$.

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 two. Similarly, our $\{+, \times\}$-programs will typically have addition operations with unbounded fan-in, and multiplication operations of fan-in two.

Define an *almost uniform sampler* (for languages) to be a randomised procedure that takes as input a $\{\cup, \cdot\}$-program for (or other description of) a language $L \subseteq \Sigma^*$, and a tolerance $0 < \varepsilon < 1$, and produces as output a word $Y \in L$ (a random variable) such that $(1+\varepsilon)^{-1}|L|^{-1} \le \Pr(Y = y) \le (1+\varepsilon)|L|^{-1}$ for all $y \in L$. Also define a *randomised approximation scheme* (for languages) to be a randomised procedure that takes input as above, and produces as output a number $\widehat{L}$ (a random variable) such that $(1+\varepsilon)^{-1}|L| \le \widehat{L} \le (1+\varepsilon)|L|$ with probability at least $\frac{3}{4}$. Our main result may now be stated.

Theorem 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 the same time complexity. The union operations in program Π may have unbounded fan-in, but the concatenation operations must all have fan-in two.*

The proof of Theorem 1 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 [4, 5] introduced a simple and elegant approach to estimating the size of—and, as a by-product, for 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. Kannan, Sweedyk and Mahaney [3] iterated the Karp-Luby method to obtain an almost uniform sampling procedure for languages computed by general $\{\cup, \cdot\}$-programs. Restated in the language of this article, the essential content of the result of Kannan et al. is the following.

Theorem 2 *There is an almost uniform sampler of time complexity* $\mathrm{O}(\varepsilon^{-2} n \times C^{\Lambda+3} \log C)$ *for producing samples from a language* $L \subseteq \Sigma^n$*, 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 the same time complexity. The union operations in program* Π *may have unbounded fan-in, but the concatenation operations must all have fan-in two.*

With a view to making the article self-contained, 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* $\operatorname{supp} p$ of polynomial $p \in \mathbb{Z}[X]$ as the set of all monomials occurring in p.

Corollary 3 *There is an almost uniform sampler of time complexity* $\mathrm{O}(\varepsilon^{-2} \times nC^{\Lambda+3} \log C)$ *for producing samples from the support* $\operatorname{supp} p$ *of a homogeneous, multilinear polynomial* p *of degree* n*, where* p *is presented as a* $\{+, \times\}$*-program* Π *of size* C *and depth* Λ*. The addition operations in program* Π *may have unbounded fan-in, but the multiplications must all have fan-in two.*

Proof Let $\Pi = (v_0, \dots, v_{C-1})$ be an $\{+, \times\}$-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. These facts imply $|\operatorname{supp} v_i| = |\operatorname{supp} v_j| \times |\operatorname{supp} v_k|$, whenever $v_i = v_j \times v_k$. Moreover, it is clear that $\operatorname{supp} v_i = \operatorname{supp} v_j \cup \operatorname{supp} v_k$, whenever $v_i = v_j + v_k$. Now apply Theorem 2. □

Observe that the time-complexity of the almost uniform sampler guaranteed by Theorem 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 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, Skyum, Berkowitz and Rackoff [10] 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 a arithmetic program that is simultaneously of small size and small depth. The following is a special case of their result.

Proposition 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* $\mathrm{O}(C^2)$ *and depth* $\mathrm{O}(\log d)$ *that computes* p*.*

Proof See [10]. 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 [10] 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. $\square$

The construction of Valiant et al. is not directly applicable 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.

Let Σ be a finite alphabet, and n a natural number. Introduce a set $X = X(\Sigma, n) = \{x_{t\sigma} : 0 \leq t \leq n-1 \text{ and } \sigma \in \Sigma\}$ of indeterminates, and define the mapping $\pi : \Sigma^n \to \mathbb{Z}[X]$ by $\pi(\alpha) = \prod_{t=0}^{n-1} x_{ta_t}$, for all words $\alpha = a_0 \dots a_{n-1} \in \Sigma^n$. 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 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 $p \in \mathbb{Z}[X]$ such that* $\operatorname{supp} p = \operatorname{supp} \pi(L)$.

Proof Let the $\{\cup, \cdot\}$-program in the statement of the lemma be $\Pi = (u_0, \dots, u_{C-1})$. 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 $0 \leq i \leq C-1$ and $0 \leq r \leq n - \deg u_i$, define $v_i^{(r)} \in \mathbb{Z}[X]$ 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 $s = \deg u_j$.

An easy induction on i establishes that $\operatorname{supp}(v_i^{(r)}) = \operatorname{supp}(\pi^{(r)}(u_i))$, for all $v_i^{(r)}$, where $\pi^{(r)}(a_0 \dots a_{s-1}) = \prod_{t=0}^{s-1} x_{r+t,a_t}$ 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 $\pi^{(0)}(L) = \pi(L)$. $\square$

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

Proof Combine Proposition 4 and Lemma 5. $\square$

We immediately have the required:

Proof of Theorem 1 Combine Corollaries 3 and 6. Note that the time to construct the small depth program of Corollary 6 is negligible in relation to the time taken to obtain a sample from it using Theorem 2. $\square$

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 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 $\varepsilon^{-2}(n\,|G|)^{\mathrm{O}(\log n)}$. 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 in which (roughly) each language computed by Π contains all words of some length l generated by some non-terminal in G. $\square$

3 The sampling algorithm and its analysis

As promised, we now present and analyse an almost uniform sampler satisfying the conditions of Theorem 2.

Consider an $\{\cup, \cdot\}$-program $\Pi = (u_i : 0 \le i \le C-1)$, of size C and depth Λ, computing a language $L \in \Sigma^n$. 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 for 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}. In estimating the size of u_i, the algorithm requires random samples from u_j for $0 \le j < i$: 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.

We use δ to bound the error in the size estimate, i.e., if $\widehat{A}$ is an estimate[1] for the size of a set A, then $\mathrm{e}^{-\delta}|A| \le \widehat{A} \le \mathrm{e}^{\delta}|A|$. We use the expression "$\widehat{A}$ is a δ-approximation to $|A|$" to denote this fact. We also use ε to bound the variation of a probability distribution from uniform, where the variation is specified in terms of the *relative pointwise distance*. This means that if $\widehat{D}_A$ denotes the distribution on a set A, then $\mathrm{e}^{-\varepsilon}|A|^{-1} \le \widehat{D}_A(x) \le \mathrm{e}^{\varepsilon}|A|^{-1}$, for every x in A. We use the expression "$\widehat{D}_A$ is ε-uniform" to denote this fact. The sampling/estimation algorithm is now described.

3.1 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 $\widehat{A}$ and $\widehat{B}$ denote the size estimates for A and B respectively, and let $\widehat{D}_A$ and $\widehat{D}_B$ denote the distributions on A and B induced by the algorithm. Note that the estimates $\widehat{A}$ and $\widehat{B}$ 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 $\widehat{S} = \widehat{A}\widehat{B}$.

If we assume that $\widehat{A}$ and $\widehat{B}$ are δ-approximations to $|A|$ and $|B|$, respectively, and that $\widehat{D}_A$ and $\widehat{D}_B$ are both ε-uniform, then it is easy to see that $\widehat{S}$ is a 2δ-approximation to $|A| \times |B|$ and that $\widehat{D}_S$ is 2ε-uniform.

3.2 Union

Suppose S is a set represented by a union operation, so that $S = \bigcup_{A \in \mathcal{A}} A$ for some finite collection $\mathcal{A}$ of sets.[2] Note that in our application, $|\mathcal{A}| \le C$. As in the case of concatenation, we let $\widehat{A}$ denote the size estimate for set A, and $\widehat{D}_A$ the induced distribution. Let $\widehat{M} = \sum_{A \in \mathcal{A}} \widehat{A}$. We assume, for all $A \in \mathcal{A}$, that $\widehat{A}$ is a δ-approximation to $|A|$, and that the distribution $\widehat{D}_A$ is ε-uniform.

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

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

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

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 denote by $\mathcal{B} = \{A \in \mathcal{A} : A \ni x\}$ the collection of $m(x)$ sets in $\mathcal{A}$ 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,

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

It is clear that $s(x)$ is a $(2\delta+\varepsilon)$-approximation to the ideal quantity (i.e., the success probability for x if all empirical size estimates $\widehat{A}$ were exact, and all distributions $\widehat{D}_A$ 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 $\widehat{D}_S(x)$, 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 $A \in \mathcal{A}$ with probability $\widehat{A}/\widehat{M}$.
  2. Produce a sample x from A according to $\widehat{D}_A$.
  3. Output $m(x)^{-1}$. (Recall that x belongs to $m(x)$ of the sets in $\mathcal{A}$.)

  The above experiment defines a random variable, say Z, that takes values in the set $\{i^{-1} : 1 \le i \le |\mathcal{A}|\}$. 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 $\widehat{\mu}$). We finally output $\widehat{\mu}\widehat{M}$ as our estimate for $|S|$.

  Note that

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

  which implies that

  $$\mu\widehat{M} = \sum_{A \in \mathcal{A}} \widehat{A} \sum_{x \in A} \widehat{D}_A(x) \frac{1}{m(x)}.$$

  In the ideal (no error) setting, the expected value of Z is just $|S|/\sum_{A \in \mathcal{A}} |A|$, and it is clear from the above that the value $\mu\widehat{M}$ is an $(\varepsilon+\delta)$-approximation to $|S|$. By performing enough trials of the above experiment, we try to ensure that the computed value $\widehat{\mu}$ 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 [2] (also see Corollary 5.2 (a) in [6]) that shows that the probability that $\widehat{\mu}$ does not satisfy $(1+\gamma)^{-1}\mu \le \widehat{\mu} \le (1+\gamma)\mu$

(and, *a fortiori*, that $\widehat{\mu}$ 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 $|\mathcal{A}|^{-1}$, which represents the case that all sets in $\mathcal{A}$ 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 $C \geq 3$.

3.3 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 $l \geq 0$,

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

and

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

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

3.4 Analysis of running time

As in the case of the error analysis, we work in terms of the levels of the program. Let $T_{\mathrm{C}}(l)$ ($T_{\mathrm{S}}(l)$) 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, $T_{\mathrm{C}}(0)$ and $T_{\mathrm{S}}(0)$ are both bounded by constants.

Let us first bound $T_{\mathrm{S}}(l+1)$. 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 $2T_{\mathrm{S}}(l)+\mathrm{O}(1)$. 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 $\mathrm{O}(nC)$. Thus,

$$T_{\mathrm{S}}(l+1) \le C\Big[T_{\mathrm{S}}(l)+\mathrm{O}(nC)\Big].$$

Solving this recurrence yields $T_{\mathrm{S}}(l)=\mathrm{O}(nC^{l+1})$.

We now bound the estimation time $T_{\mathrm{C}}(l+1)$. 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,

$$T_{\mathrm{C}}(l+1) \le t\Big[T_{\mathrm{S}}(l)+\mathrm{O}(nC)\Big],$$

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

$$T_{\mathrm{C}}(l)=\mathrm{O}\left(\varepsilon^{-2}n\,16^{\Lambda-l+1}C^{l+2}\log C\right).$$

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

$$T_{\mathrm{C}}(\Lambda)=\mathrm{O}(\varepsilon^{-2}nC^{\Lambda+2}\log C)$$

The same bound holds also for $T_{\mathrm{S}}(\Lambda)$. The total execution time is at most C times this, establishing Theorem 2.

4 Remarks and open problems

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 [8, Thm 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.

In principle the dependence of the time complexity of the sampling procedure on ε could be reduced from from ε^{-2} to $\log\varepsilon^{-1}$, using Markov chain simulation techniques described by Sinclair in [9, Cor. 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 $\widehat{A}$ are just numbers, whereas it seems that the distributions $\widehat{D}_A$ must be represented as (the code for) functions, or "objects" in more fashionable parlance. We have not been able to make this approach work.

References

[1] Michael R. Garey and David S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, CA, 1979, p. 176.

[2] W. Hoeffding, Probability inequalities for sums of bounded random variables, *Journal of American Statistical Association* **58** (1963), pp. 13–30.

[3] Sampath Kannan, Z. Sweedyk and Steve Mahaney, Counting and random generation of strings in regular languages, *Proceedings of the 6th ACM-SIAM Symposium on Discrete Algorithms*, 1995, pp. 551–557.

[4] Richard M. Karp and Michael Luby, Monte-Carlo algorithms for enumeration and reliability problems, *Proceedings of the 24th IEEE Symposium on Foundations of Computer Science*, Computer Society Press, 1983, pp. 56–64.

[5] Richard M. Karp, Michael Luby and Neal Madras, Monte-Carlo approximation algorithms for enumeration problems, *Journal of Algorithms* **10** (1989), pp. 429–448.

[6] Colin McDiarmid, On the method of bounded differences, *London Mathematical Society Lecture Note Series* **141**, Cambridge University Press, 1989, pp. 148–188.

[7] Harry Mairson, Generating words in a context-free language uniformly at random, *Information Processing Letters* **49** (1994), pp. 95–99.

[8] Noam Nisan, Lower bounds for non-commutative computation (Extended Abstract), *Proceedings of the 23rd ACM Symposium on Theory of Computing*, ACM Press 1991, pp. 410–418.

[9] A. Sinclair, *Algorithms for Random Generation and Counting: A Markov Chain Approach*, Birkhäuser, Boston, 1993.

[10] L. G. Valiant, S. Skyum, S. Berkowitz and C. Rackoff, Fast parallel computation on polynomials using few processors, *SIAM Journal on Computing* **12** (1983), pp. 641–644.