RANKING AND UNRANKING LEFT SZILARD LANGUAGES

Erkki Mäkinen

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF TAMPERE
REPORT A-1997-2

UNIVERSITY OF TAMPERE
DEPARTMENT OF COMPUTER SCIENCE
SERIES OF PUBLICATIONS A
A-1997-2, JANUARY 1997

RANKING AND UNRANKING LEFT SZILARD LANGUAGES

Erkki Mäkinen

University of Tampere
Department of Computer Science
P.O. Box 607
FIN-33101 Tampere, Finland

ISBN 951-44-4108-7
ISSN 0783-6910

RANKING AND UNRANKING LEFT SZILARD LANGUAGES

Erkki Mäkinen
Department of Computer Science, University of Tampere,
P.O. Box 607, FIN-33101 Tampere, Finland

E-mail: em@cs.uta.fi

Abstract. We give efficient ranking and unranking algorithms for left Szilard languages of context-free grammars. If $O(n^2)$ time and space preprocessing is allowed then each ranking operation is possible in linear time. Unranking takes time $O(n \log n)$. These algorithms imply similar algorithms for context-free languages generated by arbitrary unambiguous context-free grammars.

Key Words and Phrases: context-free grammar, left Szilard language, ranking, compression of program files, random generation of words.

CR Categories: F.4.2, F.4.3, F.2.2, G.2.1, E.4

1. Introduction

Ranking and unranking are fundamental combinatorial algorithms. This paper concerns ranking and unranking algorithms for context-free languages. It is known [2] that there is a polynomial time ranking algorithm for a context-free language given by an unambiguous context-free grammar. Our aim here is to sharpen this result in the case of left Szilard languages, i.e. in the case of languages consisting of the leftmost derivations of context-free grammars. We are able to show that if $O(n^2)$ time and space preprocessing phase is allowed then a ranking operation can be performed in linear time while unranking takes time $O(n \log n)$. Throughout the paper, we use the unit-cost model for time and space. Hence, we suppose that it is possible to multiply arbitrary integers in constant time and to store an arbitrary integer in one memory cell. All time and space bounds are given as a function on the length of words in left Szilard languages. The numbers of productions and nonterminals are always considered as constants.

There are two obvious applications of ranking and unranking algorithms for left Szilard languages: random generation of words over a given context-free language and compression of program files. Random generation of words over a context-free language is used in testing parsers [5], or in more theoretically oriented applications such as studying formulas in the propositional calculus or the degree of ambiguity of a context-free grammar [5]. For recent results concerning random generation of words in an ambiguous context-free grammar, see [9]. Applications of random generation of words in computational biology are mentioned in [3]. For compression of program files, see e.g. [1,7]. Parallel algorithms for ranking context-free languages are studied in [6, 8].

2. Preliminaries

If not otherwise stated we follow the notations and definitions of [4]. Let $G = (V,\Sigma,P,S)$ be a context-free grammar (hereafter simply "a grammar") whose productions are uniquely labelled by the symbols of an alphabet C. If a production $A\to\alpha$ is associated with the label ρ we write $\rho{:}A\to\alpha$. If a sequence $\rho_1 \ldots \rho_n = \omega$ of labelled productions is applied in a leftmost derivation $\beta \Rightarrow^* \gamma$, we write $\beta \Rightarrow^{\omega} \gamma$. We consider leftmost derivations only and omit the normal subscript of $\Rightarrow$ indicating leftmost derivation. The *left Szilard language* Szl(G) of G is defined as

$$\mathrm{Szl}(G) = \{\ \omega \in C^* \mid S \Rightarrow^{\omega} w,\ w \in \Sigma^*\ \}\ [10\text{-}12].$$

If β is a string over V, then $\eta(\beta)$ denotes the string obtained from β by deleting all terminal symbols. The length of β is denoted by len(β). Given a context-free grammar G, the grammar generating Szl(G) can be obtained by replacing each production $\rho{:}A\to\alpha$ in P by the production $A\to\rho\eta(\alpha)$ [10]. The grammar obtained has the property that each production has a unique terminal symbol in the beginning of its right hand side. The grammar obtained is always unambiguous. We have a one-to-one correspondence between productions in the original grammar G, the labels indicating the productions, and the productions in the grammar generating Szl(G). In the sequel we make use of these one-to-one correspondences and feel free to choose the structure that suits best the present discussion.

For the sake of notational simplicity, we assume that context-free grammars are in Chomsky normal form (CNF), so that all productions are of the form $A\to BC$ or $A\to a$, where A, B, and C are nonterminals, and a is a terminal. The productions having A in their left hand side are called *A-productions*. We say that a production of the form $A\to a$ is *terminating*; the other productions are *continuing*. Given a word ω in Szl(G), the corresponding word in L(G) is obtained by applying a homomorphism which maps the labels of continuing productions to the empty word and the labels of terminating productions to the terminal appearing in the production. So, if ρ:A→a is a production we map ρ to a.

If G is in CNF then a word w in L(G) with len(w) = n and the corresponding word ω in Szl(G) with length(ω) = m have 2n - 1 = m.

We consider the case where the length of words is fixed. This restriction does not affect to the generality of our study, since we can always easily count the words whose length is smaller than the current fixed length. Given a word ω of length n in Szl(G), the operation rank(ω) returns the rank of ω among the words of length n in Szl(G). On the other hand, unrank(x) returns the word ω with len(ω) = n in Szl(G) having rank x, provided that such a word exists.

3. Preprocessing

We determine, for each nonterminal A, an order on the set of A-productions. It does not matter whether we determine the order in the original grammar G, in the grammar generating Szl(G), or in the set of labels C. For a terminating production ρ:A→a, define pre(ρ) to be the number of terminating A-productions preceding ρ in the order of productions.

The words of length n in Szl(G) are now ordered according to the shape of their derivation trees and to the order of the productions as follows: on the upper level the words are ordered according to their derivations trees such that those having smaller left subtrees become first; words with derivation trees of the same shape are ordered according to the order of the productions used. The same ordering rule is then recursively used in the left and right subtrees. Notice that the order of derivations imposed is not lexicographical. Ranking and unranking the words in Szl(G) according to the lexicographical order seems to be much more complicated problem than the one we tackle here.

In what follows we say that a production S→AB in derivation $S \Rightarrow AB \Rightarrow^+ vw$, where v and w are terminal strings and $A \Rightarrow^+ v$ and $B \Rightarrow^+ w$, is *(i,j)-split* if len(v) = i and len(w) = j. For obvious reasons, (i,j)-split is not defined for terminating productions.

In order to be able to efficiently perform ranking and unranking operations, we need a preprocessing phase. Let $\|A\|_n$ denote the number of derivations from A to a terminal string of length n. For each production A→BC and for each length k, $0 < k \leq n$, we calculate the products $\|B\|_i * \|C\|_{k-i}$, $0 < i < n$. These numbers are cumulatively stored in table $Split_{k,i,A}$ such that for each production ρ:A→BC the entry $Split_{k,i,A}[\rho]$ contains the number of leftmost derivations producing a terminal string of length k and beginning with a (i,j)-split A-production preceding ρ in the order of productions. Moreover, we store in $Split_{k,A}[i]$, for each k and i and for each nonterminal A, the number of all derivations producing a terminal string of length k and beginning with a (s,t)-split production where $s + t = k$ and $0 < s < i$. So, $Split_{k,A}[i]$ holds the number of all derivations from A to a terminal string of length k with the property the left subtree of the derivation tree produces a string of length at most $s = i - 1$ and the right subtree produces a string of length at least $t = k - i + 1$.

The preprocessing described is clearly possible in $O(n^2)$ time and its results need $O(n^2)$ space.

4. Ranking

We first consider how to determine the rank of a given word ω having len(ω) = n among all words in Szl(G) of length n. We start by determining the splits used in the derivation. (Recall that ω is a leftmost derivation in G.) This is possible by storing the characters of ω (labels of productions) in a stack as long as further characters (labels of terminating productions) later expose the splits applied. The following example clarifies this process.

Example 1. Consider a grammar G with productions

ο:S→AB
π:S→BB
ρ:A→AA
σ:A→AB
τ:B→BA
ν:A→a
φ:B→b,

and word οσνφττφνν in Szl(G). The corresponding derivation is S ⇒ AB ⇒ ABB ⇒ aBB ⇒ abB ⇒ abBA ⇒ abBAA ⇒ abbAA ⇒ abbaA ⇒ abbaa.

We first push an instance of productions ο:S→AS and σ:A→AB onto a stack. The following two characters ν and φ indicate that terminating productions are applied to both nonterminals in the right hand side of the top element (A→AB) of the stack. This shows that the top element is a (1,1)-split production. When a split is found, the production is popped off from the stack. Popping a (i,j)-split production from the stack indicates that the new top element is a (i+j,k)-split for some k. Reading the rest of the word shows that the total sequence of splits is ο(2,3)σ(1,1)νφτ(2,1)τ(1,1)φνν. ☒

The sequence of splits can be found in linear time. (Each character of the input word causes at most one element to be once pushed and popped.)

When the sequence of splits is known, we can continue the ranking process by using the precomputed Split tables. Example 2 continues the sample ranking started in Example 1.

Example 2. We are looking for the rank of οσνφττφνν among all words of length 9 in Szl(G). Figure 1 shows the Split tables needed in the present example. We obey the order of productions given in Example 1, i.e. ο < π and ρ < σ (other order relations are irrelevant).

$\text{Split}_{5,S}$

1	0
2	24
3	33
4	43

$\text{Split}_{3,B}$

1	0
2	2

$\text{Split}_{2,1,A}$

ρ	0
σ	1

Figure 1. The Split tables consulted in Example 2.

The first character ο corresponds to a (2,3)-split production. Entry $\text{Split}_{5,S}[2] = 24$ gives the number of left derivations beginning with (1,4)-split productions. All these derivations have a rank smaller than our sample word. Since ο < π, there is no continuing S-production preceding ο in the order of productions. The second character σ corresponds to a (1,1)-split production. Since ρ < σ, we consult table $\text{Split}_{2,1,A}$. Entry $\text{Split}_{2,1,A}[\sigma] = 1$ shows that there is one derivation from A to a terminal string of length 2 before the subderivation in the sample word. This subderivation can be continued with any of the subderivations from the sibling of A (which is B) to a terminal string of length 3. There are three such derivations from B. So far, we have found 24 + 3 derivations having rank smaller than that of our sample word.

The next two characters (ν and ϕ) correspond to terminal productions. Both A and B have only one terminal production and therefore terminal productions do not increase the rank. Then we have a (2,1)-split production τ:B→BA. Entry $Split_{3,B}[2]$ shows that there are two derivations from B to a terminal word of length 3 beginning with a (1,2)-split production. Further characters do not increase the rank. (There is only one possibility to complete a derivation of length 9 from the sentential form abBAA obtained by the prefix $\omicron\sigma\nu\phi\tau$.). Hence, there are 24 + 3 + 2 = 29 derivations of length 9 before our sample word and rank($\omicron\sigma\nu\phi\tau\tau\phi\nu\nu$) = 30. ☒

We can now write the algorithm used in Examples 1 and 2 as follows:

Algorithm Ranking
Input: A grammar G and a word ω, len(ω) = n, in Szl(G).
Output: rank(ω).
Method:
1. Find the sequence of splits used in the derivation corresponding to ω;
2. rank ← 1;
3. **for** p ← 1 **to** n **do**
    if the pth character in ω corresponds to a continuing (i,j)-split production ρ:A→BC
    then **if** A is in the root or in a right child in the derivation tree
        then rank ← rank + $Split_{i+j,A}[i]$ + $Split_{i+j,i,A}[\rho]$
        else {A is in a left child in the derivation tree and the (p-1)th character in ω corresponds to a (s,t)-split production D→AE, where s = i + j}
            rank ← rank + $Split_{i+j,A}[i]$ + $Split_{i+j,i,A}[\rho] * \|E\|_t$
    else { ρ corresponds to a terminating production }
        if A is in the root or in a right child in the derivation tree
        then rank ← rank + pre(ρ)
        else {A is in a left child in the derivation tree and the (p-1)th character in ω corresponds to a (s,t)-split production D→AE}
            rank ← rank + pre(ρ) $* \|E\|_t$;

As mentioned earlier, step 1 of the algorithm takes linear time. Step 3 contains a constant number of condition tests, table look-ups and arithmetic operations per a character of the input word. Hence, the above algorithm runs in linear time.

We can modify our ranking algorithm so that the preprocessing consists only of calculating $\|A\|_k$'s for each k and A. The rest of the values can be then determined during the run of the algorithm in time $O(n^2)$.

Theorem 1. Ranking a left Szilard language is possible in linear time with $O(n^2)$ time and space preprocessing or in time $O(n^2)$ with linear time and space preprocessing.

5. Unranking

As well as in ranking, the Split tables can be used also in unranking. Again, we start with an example.

Example 3. We perform operation unrank(58) in the case of the grammar given in Example 1 considering the words of length 9. First, we consult table $Split_{5,S}$ and look for the greatest value not exceeding 58. The right entry is $Split_{5,S}[4] = 43$. We next consult table $Split_{5,4,S}$. We are looking for the greatest entry not exceeding 58 - 43 = 15. The right entry is $Split_{5,4,S}[o] = 0$. This tells us that we must start with o:S→ AB.

$Split_{4,A}$

1	0
2	10
3	16

$Split_{5,4,S}$

ο	0
π	30

$Split_{4,2,A}$

ρ	0
σ	4

Figure 2. The Split tables consulted in Example 3 (excluding $Split_{5,S}$ which is shown in Figure 1.).

Because of entry $Split_{4,A}[2] = 10$ we continue with a (2,2)-split production. From $Split_{4,2,A}$ we look for the greatest entry not exceeding 15 - 10. (Notice that $\|B\|_1 = 1$.) The right entry is $Split_{4,2,A}[\sigma] = 4$. If we complete the derivation by always choosing the first possible production, we end up with a word ω in Szl(G) having 43 + 10 + 4 = 57 preceding derivations. Hence, its rank is 58. So, we complete the derivation to be οσρνντϕνϕ. ⊠

In the following algorithm we use the phrases "find the right split", "find the right continuing production", and "find the right terminating production" in the sense demonstrated in Example 3: we are looking for the greatest table entry not exceeding the argument of the search. It must also be noticed that if the nonterminal in question is in a left child in the derivation tree, we must multiply the argument by the number of derivations to a terminal string of appropiate length from the nonterminal in the corresponding right child. We must also take care of the special cases where a cumulative table contains the same value in its two (or more) consecutive elements. We must always choose the first one of these.

Algorithm Unranking

Input: A grammar G, length n and an integer r.

Output: A word ω such that rank(ω) = r among words of length n in Szl(G) .

Method

1. rank ← r; p ← 0; current_length ← n; current_nonterminal ← S;
2. **while** rank > 1 **do begin**

   p ← p + 1;

   if current_length > 1 { we are looking for a continuing production }

   then begin

   find the right split (i,j) for current_nonterminal A and current_length k by consulting table $Split_{k,A}$;

   rank ← rank - $Split_{k,A}[i]$;

   find the right continuing A-production ρ by consulting table $Split_{k,i,A}$;

   if A is in the root or in a right child in the derivation tree

   then rank ← rank - $Split_{i+j,i,A}[\rho]$

   else {A is in a left child in the derivation tree and the (p-1)th character in ω corresponds to a (s,t)-split production D→AE where s = i + j}

   rank ← rank - $Split_{i+j,i,A}[\rho] * \|E\|_t$

   end { then }

   else { current_length = 1 } **begin**

   find the right terminating A-production ρ by studying the order of productions;

   if A is in the root or in a right child in the derivation tree

   then rank ← rank - pre(ρ)

   else {A is in a left child in the derivation tree and the (p-1)th character in w corresponds to a (s,t)-split production D→AE}

   rank ← rank + pre(ρ) $* \|E\|_t$;

   end { else }

   augment the leftmost derivation found so far by applying ρ;

   store the found splits with the current sentential form;

   current_nonterminal ← the leftmost nonterminal in the current sentential form;

   current_length ← the split value related to the new current_nonterminal;

   end; {while}
3. **if** (p < n) **and** (rank = 1)

   then complete the output word to be of length n by choosing the first possible productions (obeying the splits found);

When consulting $Split_{k,A}$ we do a binary search on k elements. In the worst case, the splits are (1,n-1), (1,n-2), ..., and so on. This gives us time bound $\sum_{i=1}^{n} \log (i - 1) = \Theta(n \log n)$. Other parts of the algorithm can be performed in linear time.

Theorem 2. Unranking a left Szilard language is possible in time $O(n \log n)$ with $O(n^2)$ time and space preprocessing.

Given an unambiguous grammar G, the words in L(G) can be unranked by using the algorithm above. Indeed, the one-to-one correspondence between words in L(G) and the leftmost derivations (words in Szl(G)) allows us to use algorithm Unranking without any changes. Unfortunately, the situation is not so simple with algorithm Ranking because we have to parse the given word before we can determine its rank. In general, parsing a word in G is a more difficult operation than ranking a word in Szl(G). Hence, in the latter case, the time needed depends on the efficiency of parsing.

Acknowledgements. This work was supported by the Academy of Finland.

References

[1] Robert D.Cameron, Source encoding using syntactic information source models. *IEEE Trans. Inf. Theor.* **IT-34** (1988), 843-850.

[2] Andrew V.Goldberg and Michael Sipser, Compression and ranking, *Proc. 17th Annual ACM Symp. on Theory of Computing*, 1985, 440-448.

[3] Vivek Gore, Mark Jerrum, Kannan Sampath, Z. Sweedyk, and Steve Mahaney, A quasi-polynomial-time algorithm for sampling words from a context-free language. Manuscript, July 1995.

[4] M.A. Harrison, *Introduction to Formal Language Theory*. Addison-Wesley, 1978.

[5] Timothy Hickey and Jacques Cohen, Uniform random generation of strings in a context-free grammar. *SIAM J. Comput.* **12** (1983), 645-655.

[6] Dung T. Huynh, The complexity of ranking simple languages. *Math. Syst. Theory* **23** (1990), 1-19.

[7] Jyrki Katajainen, Martti Penttonen and Jukka Teuhola, Syntax-directed compression of program files. *Softw. Pract. Exper.* **16**, 3 (1986), 269-276.

[8] Klaus-Jörn Lange, Peter Rossmanith and Wojciech Rytter, Parallel recognition and ranking of context-free languages, *Lecture Notes in Computer Science* **629** (1992), 24-36.

[9] Harry G. Mairson, Generating words in a context-free language uniformly at random. *Inf. Process. Lett.* **49** (1994), 95-99.

[10] Erkki Mäkinen, On context-free derivations. *Acta Universitatis Tamperensis* **198**, 1985.

[11] Etsuro Moriya, Associate languages and derivational complexity of formal grammars and languages. *Inform. Control* **22** (1973), 139-162.

[12] Martti Penttonen, On derivation languages corresponding to context-free grammars. *Acta Inform.* **3** (1974), 285-291.