BINARY TREE CODE WORDS AS CONTEXT-FREE LANGUAGES

Erkki Mäkinen

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF TAMPERE

REPORT A-1998-2

UNIVERSITY OF TAMPERE
DEPARTMENT OF COMPUTER SCIENCE
SERIES OF PUBLICATIONS A
A-1998-2, JANUARY 1998

BINARY TREE CODE WORDS AS CONTEXT-FREE LANGUAGES

Erkki Mäkinen

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

ISBN 951-44-4300-4
ISSN 0783-6910

Binary tree code words as context-free languages

Erkki Mäkinen
em@cs.uta.fi

Abstract

Given a binary tree coding system, the set of valid code words of all binary trees can be considered as a context-free language over the alphabet used in the coding system. The complexity of the language obtained varies from a coding system to another.

Xiang, Tang and Ushijima have recently proved some properties of such languages. We show that their results can be more easily proved by noticing the form of the generating grammar in question. Namely, in the most simplest cases the language obtained is a left Szilard language, a very simple deterministic language. Moreover, we prove some new results concerning the "binary tree languages".

1 Introduction

Different binary tree coding systems are introduced (for a survey, see [1]) and compared [2, 3, 4, 5] in the literature. Given a coding system, the set of valid code words forms a language over the alphabet used in the code words.

Xiang, Tang and Ushijima [5] have recently studied the properties of the languages generated by the following two context-free grammars

$$G_1 : S \to aSS, S \to bS, S \to cS, S \to d$$

and

$$G_2 : S \to aSS, S \to b.$$

G_1 produces valid code words in a system where a node with two children is labeled with a, a node with only a left (resp. right) child with b (resp. c), and a leaf with d. The code word is then obtained by reading the labels in preorder. Variants of this coding system are studied also by Korsh [6] and Bapiraju and Bapeswara Rao [7].

The coding system related to G_2 makes difference only between internal nodes and leaves. The former are labeled with a's and the latter with b's. Again, the labels are read in preorder. The code words obtained are often called *Zaks' sequences*, since they are introduced by Zaks in [8].

In what follows we study G_1 and G_2 and other context-free grammars generating valid binary tree code words. Xiang, Tang and Ushijima [5] have proved several theorems concerning the languages $L(G_1)$ and $L(G_2)$ generated by G_1 and G_2. We show that these results follow directly from the fact that G_1 and G_2 are so called left Szilard grammars obeying a certain strict form of context-free determinism.

We also consider other binary tree coding systems and the corresponding languages of their valid code words.

2 Preliminaries

When concerning context-free grammars and languages we mainly follow the notations and definitions of [9]. Similarly, we use the standard tree terminology [10].

Let $G = (V, \Sigma, P, S)$ be a context-free grammar (hereafter simply "grammar") whose productions are uniquely labeled by the symbols of an alphabet C. If a production $A \to \alpha$ is associated with the label ϕ we write $\phi : A \to \alpha$. The production labeled with ϕ is called the ϕ-production. If a sequence σ of labeled productions is applied in a leftmost derivation $\beta \Rightarrow^* \gamma$, we write $\beta \Rightarrow^\sigma \gamma$. Notice that we consider leftmost derivations only and omit the normal subscript indicating leftmost derivations. The left Szilard language $Szl(G)$ of G is defined as

$$Szl(G) = \{\sigma \in C^* \mid S \Rightarrow^\sigma w, w \in \Sigma^*\}[11].$$

We consider reduced [9] grammars only; i.e. grammars in which each nonterminal and terminal symbol appears in some terminal derivation from the start symbol to a terminal string. A production is called *terminating* if there is no nonterminals in its right hand side. Otherwise, a production is *continuing*. If w is a word over an alphabet Σ and a is a symbol in Σ then $a(w)$ stands for the number of a's in w. The empty word is denoted by λ.

Given a grammar G, a grammar generating $Szl(G)$ can be obtained by replacing each production $\phi : A \rightarrow \alpha$ in P by the production $A \rightarrow \phi\eta(\alpha)$, where η is a homomorphism erasing terminal symbols. The grammar obtained has the property that each production has a unique terminal symbol in the beginning of its right hand side and this is the only terminal in the right hand side. We refer such grammars to as *left Szilard grammars.* A left Szilard grammar 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 left Szilard grammar generating $Szl(G)$.

3 Left Szilard binary tree languages

In this section we first reformulate and reprove some result by Xiang, Tang and Ushijima [5]. Our proofs are based on the fact that G_1 and G_2 are both left Szilard grammars. Moreover, we give some new results concerning $L(G_1)$ and $L(G_2)$.

Proposition 3.1 *1. G_1 is unambiguous (Theorem 1 in [5]).*

2. A string w is a word in $L(G)$ if and only if $d(w) = a(w) + 1$. A string w' is a suffix of a word w in $L(G_1)$ if and only if $d(w') \geq a(w') + 1$ (Theorem 3 in [5]).

3. A string $w' = w_i w_{i+1} \dots w_n$ is a suffix of a word $w = w_1 \dots w_i \dots w_n$ in $L(G_1)$ if and only if

$$1 \leq d(w') - a(w') \leq i$$

(Lemma 1 of [5]).

4. For each proper prefix w' of a word in $L(G_1)$ we have $a(w') \geq d(w')$ (Theorem 5 of [5]).

5. Each word in $L(G)$ ends up with a symbol d (Corollary 3 of Theorem 3 in [5]).

Proof

1. All left Szilard grammars are unambiguous.

2. Each terminal derivation in G_1 starts from S and ends up with a string with no appearances of S. Productions $S \to bS$ and $S \to cS$ do not change the number of S's, while $S \to aSS$ increases the number and $S \to d$ decreases the number by one. Hence, in each terminal derivation there must be one production $S \to d$ more than $S \to aSS$ to rewrite all S's. The one-to-one correspondence between productions and terminal symbols implies the claim concerning words in $L(G)$.

   The claim concerning suffixes follows from the fact that each proper suffix corresponds to a derivation from some sentential form to a terminal string. In order the suffix to be proper the sentential form in question must contain at least one appearance of a nonterminal. Since S is the only nonterminal in G_1, we can repeat the reasoning above. In fact, if a sentential form contains k appearances of S then we have $d(w') = a(w') + k$ for the corresponding suffix w'.

   Again, since productions $S \to bS$ and $S \to cS$ do not change the number of nonterminals, w' is a suffix of any word $w = vw'$ where

$$a(v) - d(v) + 1 = d(w') - a(w').$$

3. The lower bound follows from Item 2. The upper bound follows from the fact that the correspondng prefix $w_1 \ldots w_{i-1}$ is produced by a derivation of length $i - 1$ where each step can increase the number of nonterminals by 1. Hence, the derivation producing w' starts from a sentential form containing at most i nonterminals. The upper bound is reached when $S \to d$ is applied i times to such a sentential form. (The derivation related to the upper bound is $S \Rightarrow \ldots \Rightarrow a^{i-1}S^i \Rightarrow \ldots \Rightarrow a^{i-1}d^i$.)

4. This is obvious from the grammatical point of view. If the claim does not hold, we have no nonterminals left, and we cannot continue the derivation. Hence, w' could not be a proper prefix. (Again, the number of b's and c's is irrelevant.)

5. $S \to d$ is the only terminating production in G_1. Each terminal derivation ends up with an application of a terminating production.

$\square$

The other results concerning $L(G_1)$ given in [5] could be proved in a similar manner.

Similar to Item 2 of Proposition 3.1 we can prove the following

Proposition 3.2 *[5] If w' is a suffix of a word in $L(G_2)$, we have*

$$b(w') - a(w') \geq 1.$$

Usually this result is given in the form

Proposition 3.3 *If w' is a proper prefix of a word in $L(G_2)$, we have*

$$a(w') - b(w') \geq 0.$$

Proposition 3.3 is known as the *dominating property* of Zaks' sequences [8].

The rest of this section is devoted to some new results concerning G_1 and G_2 and the languages generated by them.

Proposition 3.4 *Each word w, $w \neq d$, in $L(G_1)$ can be written in the form $w = xyz$, $y \neq \lambda$, such that all words xy^iz, $i = 0, 1, \ldots$, are in $L(G_1)$.*

Proof If w has a subword u in $\{b, c\}^+$, we can choose $y = u$. Otherwise, w must contain the subword ad, and we can set $y = ad$. (The subword ad is produced by the subderivation $S \Rightarrow aSS \Rightarrow adS$ which can always be repeated arbitrary number of times.) $\square$

Actually, since S is the only nonterminal in G_1, we can insert subwords $(ad)^i$, $i = 1, 2, \ldots$, and u, $u \in \{b, d\}^+$, in any word w in $L(G_1)$ and to obtain a new word in $L(G_1)$. The only restriction is that the subword cannot be inserted to the rear of w.

This observation has a natural interpretation: inserting ad corresponds to an expansion of an edge to contain a node with two children so that the left subtree has one node, and inserting u corresponds to an expansion of an edge to contain nodes with one child.

A result similar to Proposition 3.4 holds also for G_2

Proposition 3.5 *Each word w, $w \neq b$, in $L(G_2)$ can be written in the form $w = xyz$, $y \neq \lambda$, such that all words xy^iz, $i = 0, 1, \ldots$, are in $L(G_2)$.*

Proof We can always choose $y = ab$. $\square$

4 Other coding systems

The set of binary trees on n nodes is known to be in one-to-one correspondence with well-formed bracket sequences with n pairs of brackets. Hence, such a bracket sequence can be considered as a binary tree code word. The language related to this coding system can be generated by the grammar

$$G_3 : S \to [S]S, S \to \lambda.$$

Since $L(G_3)$ is not prefix-free, it cannot be a left Szilard language. ($L(G_3)$ is deterministic, but not strict deterministic in the sense of [9].)

To obtain Zaks' sequences or the words in $L(G_2)$, we can perform the following algorithm: label all internal nodes by a, label all leaves by b, and read the labels in preorder. Reading the labels in *level-by level order* (from top to bottom and form left to right) we obtain different kind of code words. Level-by-level code words are used e.g. in [12].

The set of all level-by-level code words coincides with $L(G_1)$. The only difference between the two methods is the order in which nodes are handled. In the corresponding grammars this determines the rewriting order of nonterminals in sentential forms. In the method related to G_1 this order is depth-first, while level-by level code words use "first-in-first-out" order or breadth-first order. In general, breadth-first order is not possible in normal context-free grammars. However, our grammars contain only one nonterminal (S). Hence, it does not make any difference whether we use depth-first or breadth-first order of nonterminal rewriting. (Breadth-first restriction in context-free derivations is studied e.g. in [13, 14].)

So far, we have restricted ourselves to coding systems using a fixed alphabet. Most of the coding systems introduced in the literature use integers from interval $[1..n]$ when coding binary trees on n nodes. As an example of such coding systems we shortly discuss the *left distance method* [15]. In this coding method the code item related to a node equals the node's distance from the left arm (i.e., the path from the root following the left child pointers). The code items are read in preorder in order to obtain the whole code word. The valid code words in the left distance method have a simple characterization: $(x_0, x_1, \ldots, x_{n-1})$ is a valid code if and only if $x_0 = 0$ and $0 \le x_i \le x_{i-1} + 1$, for $i = 1, \ldots, n-1$. Based on this characterization, we can define a grammar generating the code words for binary trees on at most

n nodes. The language generated is finite, and hence, a regular grammar is sufficient. The productions are essentially of the form

$$X_{i,j} \to a_0 X_{i+1,0}, X_{i,j} \to a_1 X_{i+1,1}, \ldots, X_{i,j} \to a_{j+1} X_{i+1,j+1},$$

where the subscripts (i, j) of a nonterminal indicate the position in the code word (i), and the value of the code item in the ith position (j). On the right hand sides, the position is incremented $(i + 1)$, and the possible value for the next code item is on the interval $[0..j + 1]$.

5 Conclusion

We have studied binary tree code words as context-free languages. We have shown that properties of these languages can be more easily proved by noticing the derivational structure of the corresponding context-free grammar.

References

[1] Mäkinen, E. (1991a) A survey on binary tree codings. *Comp. J.*, **34**, 438–443.

[2] Mäkinen, E. (1991b) Efficient generation of rotational-admissible codewords for binary trees. *Comp. J.*, **34**, 379.

[3] Mäkinen, E. (1992) A note on graftings, rotations, and distances in binary trees. *EATCS Bull.*, **46**, 146–148.

[4] Lucas, J.M., Roelants van Baronaigien, D. and Ruskey, F. (1993) On rotations and the generation of binary trees. *J. Algorithms*, **15**, 343–366.

[5] Xiang, L., Tang, C. and Kazuo Ushijima (1997) Grammar-oriented enumeration of binary trees. *Comp. J.*, **40**, 278–291.

[6] Korsh, J.F. (1993) Counting and randomly generating binary trees. *Inf. Process. Lett.*, **45**, 291–294.

[7] Bapiraju, V. and Bapesraju Rao, V.V. (1994) Enumeration of binary trees. *Inf. Process. Lett.*, **51**, 125–127.

[8] Zaks, S. (1980) Lexicographic generation of ordered trees. *Theoret. Comput. Sci.*, **10**, 63–82.

[9] Harrison, M.A. (1978) *Introduction to Formal Language Theory.* Addison-Wesley, Reading.

[10] Knuth, D.E. (1997) *The Art of Computer Programming. Vol 1, Fundamental Algorithms. Third Edition.* Addison-Wesley, Reading.

[11] Mäkinen, E. (1985) On context-free derivations. *Acta Universitatis Tamperensis*, **197**.

[12] Lee, C.C., Lee, D.T. and Wong, C.K. (1983) Generating binary trees of bounded height. *Acta Inf.*, **23**,529–544.

[13] Cherubini, A., Citrini, C., Crespi-Reghizzi, S. and Mandrioli, D. (1990) Breadth and depth grammars and deque automata. *Intern. J. Found. Computer Science*, **1**, 219–232.

[14] E. Mäkinen (1991), A hierarchy of context-free derivations. *Fundam. Inf.*, **14**, 255–259.

[15] E. Mäkinen (1987), Left distance binary tree representations. *BIT*, **27**, 163–169.