ELSEVIER
Theoretical Computer Science 155 (1996) 425–438
Theoretical Computer Science

Note

A note on binary grammatical codes of trees

Andrzej Ehrenfeucht[a], Paulien ten Pas[b], Grzegorz Rozenberg[a,b,*]

[a] *Department of Computer Science, University of Colorado at Boulder, Boulder, CO 80309, USA*
[b] *Department of Computer Science, Leiden University, P.O. Box 9512, 2300 RA Leiden, The Netherlands*

Received November 1994; revised March 1995
Communicated by A. Salomaa

Abstract

Grammatical codes of trees provide a way to encode ordered trees into strings over a finite alphabet in such a way that the length of each code-word is precisely the number of leaves of the coded tree. Such codes are grammatical because they result by applying production rules of a grammar G to a tree t which becomes then a derivation tree t' in G and the yeild of this derivation tree t' becomes the code-word for t. Grammatical codes were investigated in [2, 3], see also [1].

In this note we present two topics related to binary grammatical codes.

The first topic (see Section 2) is grammatical codes of binary trees with a *minimal* code alphabet. It is shown that the only binary codes that are minimal in this sense are the so-called "strict" binary codes (as considered in [3]).

The second topic (see Section 3) concerns the extension of binary grammatical codes to grammatical codes for trees of arbitrary degree. We make comparisons between classes of codes obtained in this way and the classes from [2, 3].

In Section 1 we recall (from [2, 3]) some notions and results concerning grammatical codes.

1. Grammatical codes of trees

In this note, by a tree we mean a nonempty rooted directed ordered tree without chains (i.e., each inner node of t has at least two direct descendants). Hence, a tree t is a pair $(V, \mathcal{O})$, where V is the set of nodes of t, and $\mathcal{O}$ is a function on the inner nodes of t that assigns to each inner node the sequence of its children. We use $nd(t)$ to denote the set of nodes V, $in(t)$ to denote the set of inner nodes of t, and $leaf(t)$ to denote the set of leaves of t. The *frontier of* t is the sequence of all leaves of t ordered according to $\mathcal{O}$.

* Corresponding author. Email: rozenber@wi.leidenuniv.nl.

SSDI 0304-3975(95)00093-3

A *binary tree* is a tree in which each internal node has exactly two direct descendants—hence here binary trees are *full* binary trees. A *node-labeled tree* t is a pair (t', η), where t' is a tree and $\eta: nd(t') \to \Sigma$ is a mapping, with Σ an alphabet. We say that t' is the *underlying tree of* t, denoted by $und(t)$. The notation and terminology concerning $und(t)$ carries over to t. Also, $yield(t) = \eta(v_1) \cdots \eta(v_n) \in \Sigma^+$, where $v_1 \cdots v_n$ is the frontier of t. An *inner-labeled tree* is a pair (t, η) where t is a tree and η is a mapping defined on the *inner* nodes of t.

We do not distinguish between isomorphic trees. Trees $t = (V, \mathcal{O})$ and $t' = (V', \mathcal{O}')$ are *isomorphic* if there is a bijection $\delta: V \to V'$ such that for each $v \in in(t)$, $\mathcal{O}(v) = (v_1, \ldots, v_n)$ iff $\mathcal{O}'(\delta(v)) = (\delta(v_1), \ldots, \delta(v_n))$. The set of all (chain-free) trees modulo isomorphism is denoted by T; the set of all binary trees modulo isomorphism is denoted by T_b; for $n \geqslant 1$, $T_b(n)$ denotes the set of all binary trees with n leaves modulo isomorphism.

By a *code* we mean a mapping $\varphi: T \to \Sigma^+$ that is injective and *length-preserving*, i.e., for each $t \in T$, $|\varphi(t)| = \#leaf(t)$. Analogously, a *binary code* is a mapping $\varphi: T_b \to \Sigma^+$ that is injective and length-preserving. A word $\varphi(t)$ in Σ^+ with $t \in T$ (or $t \in T_b$ in the binary case) is called a *code-word*; the set of all code-words of φ is denoted by $ran(\varphi)$.

A *OS system* is like a context-free grammar, except that it does not have terminal symbols, and it may have infinitely many productions. Formally, a OS system G is a triple (Σ, P, σ), where Σ is the (finite) alphabet of G, P is the (possibly infinite) set of productions of the form $a \to x$, with $a \in \Sigma$ and $x \in \Sigma^+$, $|x| \geqslant 2$, and $\sigma \in \Sigma$ is the axiom. The OS system G generates words in the usual way, as follows. One derivation step amounts to the substitution of a word x for an occurrence of a letter a, where $a \to x$ is a production in P. If a word $v \in \Sigma^+$ is obtained from $w \in \Sigma^+$ by a finite number of consecutive derivation steps, then we say that v is *derived from* w. The words generated by the OS system G are then the words derived from the axiom σ. We use L_G to denote $\{b \in \Sigma \mid a \to by \in P \text{ for some } y \in \Sigma^+ \text{ and } a \in \Sigma\}$, M_G to denote $\{b \in \Sigma \mid a \to xby \in P \text{ for some } x, y \in \Sigma^+ \text{ and } a \in \Sigma\}$, and R_G to denote $\{b \in \Sigma \mid a \to xb \in P \text{ for some } x \in \Sigma^+ \text{ and } a \in \Sigma\}$.

A OS system $G = (\Sigma, P, \sigma)$ is *backwards deterministic* if $a \to x \in P$ and $a' \to x \in P$ imply that $a = a'$. It is *semi-determinsitic* if for each $n \geqslant 2$ and for each $a \in \Sigma$ there is exactly one production $a \to x$ with $|x| = n$. If G is a semi-deterministic OS system, then each tree t is the underlying tree of exactly one derivation tree of G; this derivation tree is denoted by $t[G]$. A OS system G is *unambiguous* if for each w generated by G, there is a unique derivation tree of w.

A code $\varphi: T \to \Sigma^+$ is *grammatical* if there is a semi-deterministic OS system $G = (\Sigma, P, \sigma)$ such that for each $t \in T$, $\varphi(t) = yield(t[G])$. Note that if such a OS system exists, then it is unique – we say then that G *determines* φ. In what follows, we shall not distinguish a grammatical code $\varphi: T \to \Sigma^+$ from the OS system $G = (\Sigma, P, \sigma)$ determining φ, and we use φ for the mapping on T as well as for the OS system. Clearly (see [3]), a semi-deterministic OS system φ determines a code in the above-mentioned way iff φ is unambiguous.

In [2, 3], grammatical codes were investigated that had a property stronger than unambiguity, the so-called "unique origin property". A OS system $\varphi = (\Sigma, P, \sigma)$ has the *unique origin property* if for each $x \in \Sigma^+$ there is a unique $y \in \Sigma^+$ such that if x is derived from some $y' \in \Sigma^+$, then y' is derived from y, and the derivation forest of x from y is unique. OS systems with the unique origin property were characterized in [3] using the notion of nonoverlapping right-hand sides (Proposition 1.1). In general, we say that words $x, y \in \Sigma^+$ (where possibly $x = y$) are *overlapping* if there exist $v \in \Sigma^+$, $u_1, u_2, w_1, w_2 \in \Sigma^*$ such that $x = u_1 v w_1$, $y = u_2 v w_2$, $u_1 w_1 u_2 w_2 \neq \Lambda$, and $u_i w_j = \Lambda$ for some $i, j \in \{1, 2\}$.

Proposition 1.1. *A semi-deterministic OS system φ has the unique origin property iff it is backwards deterministic, and for all right-hand sides x, y of φ, x and y are not overlapping.*

Thus, every semi-deterministic and backwards deterministic OS system with nonoverlapping right-hand sides is unambiguous, and hence a grammatical code. Such grammatical codes are called "nonoverlapping codes". We recall some types of grammatical codes introduced in [2, 3] which are subclasses of the nonoverlapping codes.

Definition 1.2. (1) A grammatical code $\varphi = (\Sigma, P, \sigma)$ is *nonoverlapping* if it is backwards deterministic, and for all productions $a \to x$ and $b \to y$, x and y are not overlapping.

(2) A grammatical code $\varphi = (\Sigma, P, \sigma)$ is *marked* if it is backwards determinstic and $\{L_\varphi, M_\varphi, R_\varphi\}$ is a partition of Σ.

(3) A grammatical code $\varphi = (\Sigma, P, \sigma)$ is *strict* if it is backwards deterministic and $\{L_\varphi, M_\varphi, R_\varphi\}$ is a partition of Σ such that $\# M_\varphi = 1$, and either $\# L_\varphi = 2$ and $\# R_\varphi = 3$, or $\# L_\varphi = 3$ and $\# R_\varphi = 2$.

Clearly, each strict code is marked, and each marked code is nonoverlapping. It was shown in [3] that nonoverlapping and marked codes have an alphabet of at least 6 letters, and that each marked code with 6 letters is strict. In [2, 3] it was assumed that the axiom was in M_φ, but here we omit this restriction.

We have similar results in the case of binary codes. A *binary OS system* is a OS system such that for each production $a \to x$, $|x| = 2$; it is *deterministic* if for each $a \in \Sigma$ there is exactly one production $a \to x$ in P. A binary code is *grammatical* if it is determined by a deterministic binary OS system. Binary grammatical codes have the *unique origin property* if they are backwards deterministic, and no right-hand sides are overlapping. In the binary case, the fact that right-hand sides are not overlapping trivially implies that $\{L_\varphi, R_\varphi\}$ is a partition of Σ; hence, in the binary case the notions of marked code and nonoverlapping code coincide.

Definition 1.3. (1) A binary grammatical code $\varphi = (\Sigma, P, \sigma)$ is *marked* if it is backwards deterministic and $\{L_\varphi, R_\varphi\}$ is a partition of Σ.

(2) A binary grammatical code $\varphi = (\Sigma, P, \sigma)$ is *strict* if it is backwards deterministic and $\{L_\varphi, R_\varphi\}$ is a partition of Σ such that $\#L_\varphi = 2$ and $\#R_\varphi = 2$.

It was shown in [3] that each marked binary code has an alphabet of at least 4 letters, and that a binary code is strict iff it is marked and has 4 letters. As a matter of fact there are 24 distinct nonisomorphic strict binary codes (see [3]). Strict codes can easily be decoded, see [3, 1].

Example 1.4. (1) Consider the OS system $\varphi = (\{\ell_1, \ell_2, \ell_3, m, r_1, r_2\}, P, \ell_1)$, where P consists of the following productions (for each $k \geqslant 1$):

$$\ell_1 \to \ell_2 m^k r_1, \quad m \to \ell_3 m^k r_1,$$

$$\ell_2 \to \ell_2 m^k r_2, \quad r_1 \to \ell_1 m^k r_1,$$

$$\ell_3 \to \ell_3 m^k r_2, \quad r_2 \to \ell_1 m^k r_2.$$

φ is a strict code.

(2) Consider the binary OS system $\varphi = (\{a, b, c, d\}, P, a)$ where P consits of the productions $a \to bc$, $b \to bd$, $c \to ac$, and $d \to ad$. Then φ is a strict binary code.

2. Binary grammatical codes with 4 letters

To code binary trees (in a length-preserving manner) one needs at least 4 letters, since the number of binary trees with n leaves is greater than 3^n for sufficiently large n (see, e.g., [4]). As shown in [3] minimal binary codes (i.e., binary codes with an alphabet of 4 letters) exist, e.g., strict binary codes are such codes. Here we will show (Theorem 2.7) that strict codes are the only binary codes that are minimal and grammatical.

It is well-known (see, e.g., [4]) that the number b_n of binary trees with n leaves is "close to 4^n". More precisely,

$$b_n = \frac{4^n}{\sqrt{\pi} n^{3/2}} (1 + \mathcal{O}(1/n)). \tag{1}$$

We will use the following consequence of this fact.

Lemma 2.1. *For each α with $0 < \alpha < 4$, there exists a natural number $N_1 > 0$ such that for each $n \geqslant N_1$, $b_n > \alpha^n$.*

Proof. Since $\alpha < 4$, it follows that there exists a natural number $N_1 > 0$ such that for each $n \geqslant N_1$.

$$\alpha^n \frac{n^{3/2}}{4^n} = \frac{n^{3/2}}{(4/\alpha)^n} < \frac{1}{\sqrt{\pi}}.$$

Combining this with (1) we obtain that for each $n \geqslant N_1$,

$$b_n > \frac{4^n}{\sqrt{\pi n^{3/2}}} > \alpha^n. \qquad \square$$

Lemma 2.1 implies that, given a binary code with 4 letters, every word over its alphabet occurs in some code-word – this is shown in the next lemma. In fact, we prove something stronger: every word occurs in some code-word at a position which is a multiple of its length plus one.

For $w, x \in \Sigma^+$ and $1 \leqslant k \leqslant |w|$, we say that x is a *k-segment of* w if $|x| = k$ and $w = uxv$ with $|u|$ a multiple of k.

Lemma 2.2. *Let $\varphi: T_b \to \Sigma^+$ be a binary code with $\#\Sigma = 4$. For each $x \in \Sigma^+$ there exists a $t \in T_b$ such that x is a $|x|$-segment of $\varphi(t)$.*

Proof. Let $x \in \Sigma^+$ with $|x| = k$. Assume to the contrary that for all $t \in T_b$, x is not a k-segment of $\varphi(t)$. Then

$$ran(\varphi) \subseteq (\Sigma^k - \{x\})^* \left(\bigcup_{j=0}^{k-1} \Sigma^j \right).$$

In particular, if n is a multiple of k, then

$$\#(ran(\varphi|_{T_b(n)})) \leqslant (4^k - 1)^{n/k}.$$

Since φ is injective, $b_n = \#T_b(n) = \#(ran(\varphi|_{T_b(n)}))$. We conclude that $b_n \leqslant (4^k - 1)^{n/k}$ for each n that is a multiple of k.

However, by Lemma 2.1 with $\alpha = (4^k - 1)^{1/k}$, we have that for sufficiently large n, $b_n > \alpha^n = (4^k - 1)^{n/k}$; a contradiction.

Consequently, there is a $t \in T_b$ such that x is a k-segment of $\varphi(t)$. $\square$

To prove the main theorem of this section (Theorem 2.7), we need a particular implication of Lemma 2.2. Given two distinct words x and y of the same length, we wish to construct a "context" for them, i.e., a pair of code-words which differ only in the occurrences of x and y. Lemma 2.4, which follows easily from Lemma 2.2, states that this is possible.

Let $x, y \in \Sigma^+$ be such that $|x| = |y|$ and $x \neq y$. We define a mapping $h_{(x,y)}: \Sigma^+ \to \Sigma^+$ that replaces every $|x|$-segment x of w by y. Formally, if $w = v_1 v_2 \ldots v_n u \in \Sigma^+$, with $n \geqslant 0$, $|v_i| = |x|$ for $i = 1, \ldots, n$, and $|u| < |x|$, then $h_{(x,y)}(w) = v_1' \ldots v_n' u$, where for $i = 1, \ldots, n$, $v_i' = v_i$ if $v_i \neq x$, and $v_i' = y$ otherwise.

Definition 2.3. Let $\varphi: T_b \to \Sigma^+$ be a binary code with $\#\Sigma = 4$. For $n \geqslant 1$, binary trees $t_1, t_2 \in T_b(n)$, and $x, y \in \Sigma^+$ with $x \neq y$ and $|x| = |y|$, t_1 and t_2 are (x, y)-*related* if $h_{(x,y)}(\varphi(t_1)) = h_{(x,y)}(\varphi(t_2))$.

Hence, t_1 and t_2 are (x, y)-related if $\varphi(t_1)$ and $\varphi(t_2)$ differ only in some k-segments which equal x in one of these words and y in the other one.

Lemma 2.4. *Let $\varphi: T_b \to \Sigma^+$ be a binary code with $\#\Sigma = 4$. For each pair $x, y \in \Sigma^+$ with $|x| = |y|$ and $x \neq y$, there exist $n \geqslant 1$ and $t_1, t_2 \in T_b(n)$ with $t_1 \neq t_2$ such that t_1 and t_2 are (x, y)-related.*

Proof. Let $x, y \in \Sigma^+$ be such that $|x| = |y|$ and $x \neq y$. Assume to the contrary that for every n, and for all $t_1, t_2 \in T_b(n)$, if $t_1 \neq t_2$, then t_1 and t_2 are not (x, y)-related. This implies that the mapping $h_{(x,y)} \circ \varphi$ is injective (and hence a binary code). However, by the definition of $h_{(x,y)}$, for every $t \in T_b$, x is not a $|x|$-segment of $h_{(x,y)}(\varphi(t))$, which contradicts Lemma 2.2. □

The next two lemmas yield the proof of the main theorem.

Lemma 2.5. *Each binary grammatical code $\varphi = (\Sigma, P, \sigma)$ with $\#\Sigma = 4$ is backwards deterministic.*

Proof. Let $\varphi = (\Sigma, P, \sigma)$ be a binary grammatical code with $\#\Sigma = 4$, and assume to the contrary that φ has productions $a \to pq$ and $b \to pq$, with $a \neq b$. By Lemma 2.4 there exist $n \geqslant 1$ and $t_1, t_2 \in T_b(n)$ with $t_1 \neq t_2$ such that t_1 and t_2 are (a, b)-related. Now for each j, if the jth letter of $\varphi(t_1)$ differs from the jth letter of $\varphi(t_2)$ (which implies that these letters are a and b), then add two direct descendants to the jth leaf of t_1 and to the jth leaf of t_2. For the so obtained trees t_1' and t_2' we have $t_1' \neq t_2'$ and, since a and b have the same right-hand side in P, we have $\varphi(t_1') = \varphi(t_2')$, which contradicts the injectivity of φ. □

Lemma 2.6. *For each binary grammatical code $\varphi = (\Sigma, P, \sigma)$ with $\#\Sigma = 4$, $\{L_\varphi, R_\varphi\}$ is a partition of Σ.*

Proof. Let $\varphi = (\Sigma, P, \sigma)$ be a binary grammatical code with $\#\Sigma = 4$. It is sufficient to show that $L_\varphi \cap R_\varphi = \emptyset$. Assume to the contrary that there is a $q \in L_\varphi \cap R_\varphi$. Hence, φ has productions $a \to pq$ and $b \to qr$. Consider the words ar and pb.

Suppose first that $ar \neq pb$. By Lemma 2.4, there exist $n \geqslant 1$ and $t_1, t_2 \in T_b(n)$ with $t_1 \neq t_2$ such that t_1 and t_2 are (ar, pb)-related. Let $w_1 = \varphi(t_1)$, and $w_2 = \varphi(t_2)$. For every odd j, $1 \leqslant j < n$, do the following: if $w_1(j)w_1(j+1) = ar$ and $w_2(j)w_2(j+1) = pb$, then add two direct descendants to the jth leaf of t_1, and add two direct descendants to the $(j+1)$st leaf of t_2; if $w_1(j)w_1(j+1) = pb$ and $w_2(j)w_2(j+1) = ar$, then do the same with the roles of t_1 and t_2 interchanged. Let t_1' be the tree obtained in this way from t_1, and let t_2' be the tree obtained in this way from t_2. Since $w_1 \neq w_2$, there is a leaf of one of these trees, say the jth leaf, that has been added in the above construction as the right child of a node with lable a. But then

in the other tree the jth leaf is a left child, and thus $t'_1 \neq t'_2$. Also, since t_1 and t_2 are (ar, pb)-related, it follows that $\varphi(t'_1) = \varphi(t'_2)$. This contradicts the injectivity of φ.

In the case that $ar = pb$, we start with a tree $t \in T_b$ such that ar occurs in $\varphi(t)$ (by Lemma 2.2 such a tree exists), and construct two trees out of t. Let j be such that $\varphi(t) = warz$, with $|w| = j - 1$. The first tree is obtained by adding two descendants to the jth leaf of t, and the second tree is obtained by adding two descendants to the $(j + 1)$st leaf of t. Clearly, these trees are different, but they get the same code-word. Hence, also in this case we obtain a contradiction with the injectivity of φ.

Consequently, $L_\varphi \cap R_\varphi = \emptyset$. □

Theorem 2.7. *Each binary grammatical code with an alphabet of* 4 *letters is a strict binary code.*

Proof. By Lemmas 2.5 and 2.6 each binary grammatical code with 4 letters is marked, and hence it is strict. □

For binary grammatical codes with larger alphabets, Lemma 2.6 no longer holds, as will be shown in Example 2.8. Note that if a left leaf is followed by a right leaf in the frontier of a tree $t \in T_b$, then these leaves are the two children of one node; we call such a pair of leaves a *complete pair of* t.

Example 2.8. Let φ be a binary OS system with productions $a \to ae$, $b \to bd$, $c \to ac$, $d \to ed$, and $e \to bc$, and some arbitrary axiom. Clearly, this OS system is deterministic and backwards deterministic, but not marked. Since the right-hand sides ae and ed overlap, φ does not have the unique origin property. However, we will show that φ *is* unambiguous. Hence φ is a grammatical code.

Let $n \geqslant 1$, and let $t_1, t_2 \in T_b(n)$ be such that $\varphi(t_1) = \varphi(t_2)$. We will show by induction on n that $t_1[\varphi] = t_2[\varphi]$ (and hence $t_1 = t_2$). If $n = 1$, then trivially $t_1 = t_2$. Now suppose that the claim holds for all $n \leqslant k$, for some $k \geqslant 1$. Let $n = k + 1$, and let $t_1, t_2 \in T_b(n)$. We look for a subword in $\varphi(t_1)$ that labels a complete pair in $t_1[\varphi]$ as well as in $t_2[\varphi]$. Then we can apply the induction hypothesis after removing from each tree the complete pair, and conclude that $t_1[\varphi] = t_2[\varphi]$.

Note that in any code-word $\varphi(t)$, the letters a and b label left leaves in $t[\varphi]$, and c and d label right leaves. Hence, if $\varphi(t_1)$ contains a subword of the form ac, bc, or bd, then this subword labels a complete pair in both $t_1[\varphi]$ and $t_2[\varphi]$. We claim that if $\varphi(t_1)$ contains a subword ae, then this also labels a complete pair, because the corresponding occurrence of e cannot label a left leaf.

To see this, assume to the contrary that e labels a left leaf in $t[\varphi]$. Then the parent v of this leaf has label d. Let t'_1 be the tree obtained from t_1 by removing all nodes below v. Then the subword ad occurs in $\varphi(t'_1)$, and it labels a left and a right leaf, i.e., a complete pair in $t'_1[\varphi]$. This contradicts the fact that ad is not a right-hand side of φ.

A symmetric argument applies for every occurrence of ed: if e labels a right leaf, then its parent has label a, contradicting the fact that ad cannot label a complete pair. □

It follows from Theorem 2.7 and Proposition 1.1 that every binary OS system with 4 letters that is semi-deterministic and unambiguous has the unique origin property. It might be interesting to see whether in coding arbitrary chain-free trees the situation is similar, i.e., whether the fact that a grammatical code is minimal (using 6 letters) implies that it has the unique origin property, meaning that it is a nonoverlapping code.

3. Extensions of binary codes

One way to obtain grammatical codes for arbitrary trees is to extend a given binary grammatical code. The idea is based on the following translation of arbitrary trees into binary trees. What happens is that each inner node together with its direct descendants is refined into a binary subtree, where additionally a labeling of the inner nodes denotes whether a node comes from the original tree or is constructed in the refinement.

Definition 3.1. Let $t = (V, \mathcal{O})$ be a tree. Let $v \in in(t)$ with $\mathcal{O}(v) = (u_1, \ldots, u_n)$, $n \geqslant 2$. Define a new set of nodes $W_v = \{v^{(2)}, \ldots, v^{(n-1)}\}$, and corresponding ordering functions $\mathcal{O}_v$ on $\{v\} \cup W_v$ as follows.

If $n > 2$, then

$$\mathcal{O}_v(v) = (u_1, v^{(2)}),$$

$$\mathcal{O}_v(v^{(i)}) = (u_i, v^{(i+1)}) \quad \text{for } i = 2, \ldots, n-2 \text{ and}$$

$$\mathcal{O}_v(v^{(n-1)}) = (u_{n-1}, u_n).$$

If $n = 2$, then

$$\mathcal{O}_v(v) = \mathcal{O}(v) = (u_1, u_2).$$

Then the *binary refinement of* t, denoted by $bin(t)$, is the inner-labeled binary tree $(V \cup \bigcup_{v \in in(t)} W_v, \bigcup_{v \in in(t)} \mathcal{O}_v, \eta)$ where η is defined by $\eta(v) = 1$ if $v \notin V$ and $\eta(v) = 0$ if $v \in V$.

Example 3.2. Fig. 1 gives an example of a tree t and its binary refinement $bin(t)$.

Note that the mapping *bin* that assigns to every $t \in T$ the binary refinement $bin(t)$ is injective.

For a tree t of arbitrary degree, if $bin(t) = (t', \eta)$, then the binary tree t' can be coded by means of a binary grammatical code φ. Moreover, in order to mark which nodes have label 1 in $bin(t)$, we adapt the node-labeling of $t'[\varphi]$ as follows: if v is a node in $bin(t)$ with label 1, then for each node (other than v) on the path from v to the leaf that is the leftmost descendant of v, its label a in $t'[\varphi]$ is changed into $\hat{a}$. The yield of this

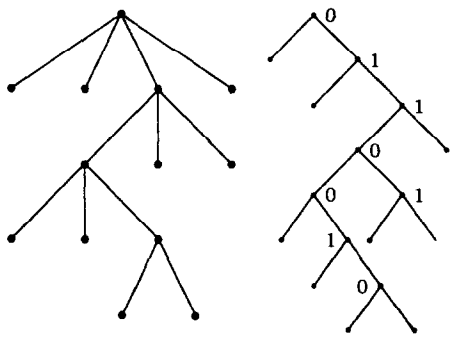


Fig. 1. A tree t and its binary refinement $bin(t)$.

tree will be the code-word for t. Since only right children can have label 1 in $bin(t)$, $bin(t)$ can be recovered from this yield as follows: by removing the hats $\varphi(t')$ is obtained, and hence t'; now if the ith letter of the yield has a hat, then the lowest ancestor of the ith leaf of t' that is a right child gets label 1.

Since *bin* is injective, this construction indeed gives a code for T, in the sense that different trees get different code-words. Moreover, it can be defined by the following OS system, which means that this code is grammatical.

Definition 3.3. Let φ be a binary grammatical code. The *extension of* φ is the OS system $\varphi_{ext} = (\Sigma_{ext}, P_{ext}, \sigma)$ where $\Sigma_{ext} = \Sigma \cup \{\hat{\ell} \mid \ell \in L_\varphi\}$, and

$$P_{ext} = \{a \to \ell_1\hat{\ell}_2 \ldots \hat{\ell}_n r_n \mid n \geqslant 1, a \to \ell_1 r_1, r_1 \to \ell_2 r_2, \ldots, r_{n-1} \to \ell_n r_n \in P\}$$
$$\cup \{\hat{a} \to \hat{\ell}_1\hat{\ell}_2 \ldots \hat{\ell}_n r_n \mid n \geqslant 1, a \in L_\varphi, a \to \ell_1 r_1, r_1 \to \ell_2 r_2, \ldots, r_{n-1} \to \ell_n r_n \in P\}.$$

Theorem 3.4. *For each binary grammatical code φ, the extension φ_{ext} is a grammatical code for the set of chain-free trees.*

Proof. Let $\varphi = (\Sigma, P, \sigma)$ be a binary code, and let $\varphi_{ext} = (\Sigma_{ext}, P_{ext}, \sigma)$ be the extension of φ. By the construction of φ_{ext}, and since φ is deterministic, it follows that for each $n \geqslant 2$, and each $a \in \Sigma_{ext}$, there is exactly one production $a \to w$ in P with $|w| = n$. Hence, φ_{ext} is semi-deterministic.

Let $t \in T$ be a tree, and let $bin(t) = (t', \eta)$. Let $\alpha : nd(t') \to \Sigma_{ext}$ be the adapted node-labeling as described above. It follows from the definitions of $bin(t)$ and of φ_{ext} that $t[\varphi_{ext}] = (t, \alpha|_{nd(t)})$. This implies that $bin(t)$, and hence the original tree t, can be uniquely determined from $yield(t[\varphi_{ext}])$. Hence φ_{ext} is unambiguous.

Consequently, φ_{ext} is a grammatical code for T. □

Example 3.5. Consider the strict binary code $\varphi = (\{a, b, c, d\}, P, a)$ from Example 1.4(2), with production set $a \to bc$, $b \to bd$, $c \to ac$, $d \to ad$. Let φ_{ext} be the

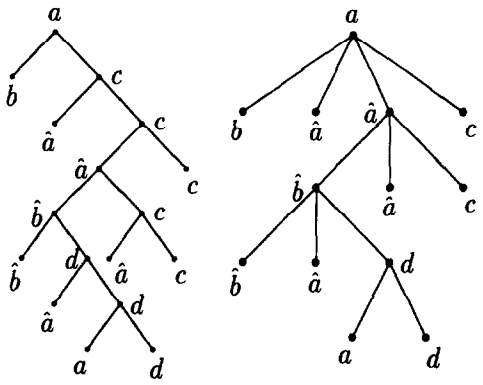


Fig. 2. *und*(*bin*(*t*)) with adapted node-labeling and $t[\varphi_{ext}]$.

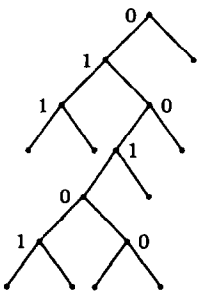


Fig. 3. A symmetric way of refining t.

extension of φ. Then the productions of φ_{ext} are (for each $k \geqslant 1$)

$$a \to b\hat{a}^k c, \qquad d \to a\hat{a}^k d,$$

$$b \to b\hat{a}^k d, \qquad \hat{a} \to \hat{b}\hat{a}^k c,$$

$$c \to a\hat{a}^k c, \qquad \hat{b} \to \hat{b}\hat{a}^k d.$$

Note that φ_{ext} is a strict code; in fact it is (a renaming of) the strict code given in Example 1.4(1).

Fig. 2 gives the adapted node-labeling α for the binary refinement from Fig. 1, and $t[\varphi_{ext}]$.

Remark 3.6. In Definition 3.3, one could also refine the nodes of degree larger than 2 in a "leftmost" way, as done in Fig. 3 for the tree t of Fig. 1. Then an extension code is obtained where copies of right letters are added; symmetric results hold for these extensions.

Remark 3.7. In [3] a way of coding node-labeled trees was discussed where the underlying tree is coded with a marked code, and the node-labels are stored in the leaves by use of a so-called "direction function". Extension codes closely correspond to such a way of coding the binary refinements of T (which are inner-labeled binary trees), provided that we extend *marked* binary codes. More precisely, let $\varphi = (\Sigma, P, \sigma)$ be a marked binary code, let $\psi : \Sigma \to \{left, right\}$ be the direction function defined by $\psi(a) = left$ if $a \in R_\varphi$ and $\psi(a) = right$ if $a \in L_\varphi$, and consider the code word w of $bin(t)$ given by φ and ψ (as described in [3]). Then each label 1 ends up in a left leaf of $bin(t)$, and for $a \in \Sigma$, $\hat{a}$ and a in $\varphi_{ext}(t)$ correspond to letters $(a, 1)$ and $(a, 0)$, respectively, in w.

So far, we have imposed no restrictions on the form of φ. The following theorem states that nonoverlapping (i.e., marked) binary codes extend to nonoverlapping codes

for T. Moreover, a condition is given that characterizes those binary codes that extend to marked codes. We use LL_φ to denote $\{b \in L_\varphi \mid$ there exists $a \in L_\varphi$ such that $a \to bc$ in $\varphi\}$, and LR_φ to denote $\{b \in L_\varphi \mid$ there exists $a \in R_\varphi$ such that $a \to bc$ in $\varphi\}$.

Theorem 3.8. *Let φ be a binary grammatical code, and let φ_{ext} be the extension of φ.*

(1) *φ is a marked binary code iff φ_{ext} is a nonoverlapping code.*

(2) *φ is backwards deterministic and $LL_\varphi \cap LR_\varphi = \emptyset$ iff φ_{ext} is a marked code.*

Proof. Let $\varphi = (\Sigma, P, \sigma)$ be a binary code, and let $\varphi_{ext} = (\Sigma_{ext}, P_{ext}, \sigma)$ be the extension of φ.

(1) Firstly, assume that φ is marked. It follows from the backwards determinism of φ that φ_{ext} is backwards deterministic. We show now that the right-hand sides of φ_{ext} do not overlap. From the fact that L_φ and R_φ are disjoint and that every right-hand side of φ_{ext} is in $(L_\varphi \cup L_{ext}) \cdot L_{ext}^* \cdot R_\varphi$, where $L_{ext} = \{\hat{\ell} \mid \ell \in L_\varphi\}$ we obtain that the only possibility for overlapping right-hand sides is that one right-hand side is a suffix of another right-hand side. More precisely, there is a production of the form $a \to \ell_1 \hat{\ell}_2 \ldots \hat{\ell}_n r_n$ or $\hat{a} \to \hat{\ell}_1 \hat{\ell}_2 \ldots \hat{\ell}_n r_n$, with $a, \ell_1, \ldots, \ell_n, r_n \in \Sigma$, and a production of the form $\hat{b} \to \hat{\ell}_j \ldots \hat{\ell}_n r_n$, with $2 \leqslant j \leqslant n$, and $b \in L_\varphi$. This implies the existence of productions $a \to \ell_1 r_1, r_1 \to \ell_2 r_2, \ldots, r_{n-2} \to \ell_{n-1} r_{n-1}, r_{n-1} \to \ell_n r_n$ in P as well as productions $b \to \ell_j r'_j, r'_j \to \ell_{j+1} r'_{j+1}, \ldots, r'_{n-2} \to \ell_{n-1} r'_{n-1}, r'_{n-1} \to \ell_n r_n$. Since φ is backwards deterministic, it follows that $r_{n-1} = r'_{n-1}$, and hence also that $r_{n-2} = r'_{n-2}, \ldots, r_j = r'_j$, and $r_{j-1} = b$. However, $r_{j-1} = b$ contradicts the fact that $b \in L_\varphi$. Hence, no right-hand sides of P_{ext} overlap. Consequently, φ_{ext} is a nonoverlapping code.

Conversely, if φ_{ext} is nonoverlapping, then, since all productions of φ are also productions of φ_{ext}, L_φ and R_φ are disjoint and φ is backwards deterministic – hence φ is marked.

(2) By the construction of φ_{ext}, $L_{\varphi_{ext}} = L_\varphi \cup \{\hat{\ell} \mid \ell \in LL_\varphi\}$, $M_{\varphi_{ext}} = \{\hat{\ell} \mid \ell \in LR_\varphi\}$, and $R_{\varphi_{ext}} = R_\varphi$. Note that disjointness of LL_φ and LR_φ implies disjointness of L_φ and R_φ. Consequently, $L_{\varphi_{ext}}$, $M_{\varphi_{ext}}$ and $R_{\varphi_{ext}}$ are mutually disjoint iff LL_φ and LR_φ are disjoint. By (1), if φ_{ext} is marked, then φ is backwards deterministic. Hence, φ satisfies the given condition iff φ_{ext} is marked. □

Theorem 3.8 implies that φ is a strict binary code iff φ_{ext} is a nonoverlapping code with 6 letters. These minimal nonoverlapping codes obtained by extending strict codes are not necessarily strict, see, e.g., the code from Example 3.9(1). Note that strict codes obtained as extensions have 3 left letters; using the symmetric extension of Remark 3.6 one may obtain strict codes with 3 right letters. It is impossible that both symmetric versions of the extension of a strict code are strict. More precisely, of the 24 strict binary codes 16 codes have extensions that are not strict; of the other 8 codes 4 codes yield so-called 'insertive' (see [2]) strict extensions according to Definition 3.3, and 4 codes yield 'insertive' strict extensions following a symmetric definition (see Remark 3.6). In fact, the 4 binary codes giving strict extensions according to Definition 3.3 have the productions of Example 3.5 and one of the 4 letters as axiom. The

production set of the obtained extensions gives, with the right choice of the axiom, one of the two strongly recursive dependent codes with 3 left letters which were discussed in [2].

Example 3.9. (1) Consider the strict binary code $\varphi = (\{a, b, c, d\}, P, a)$ where P consists of the productions $a \to ac$, $b \to bd$, $c \to bc$, and $d \to ad$, and let φ_{ext} be the extension of φ. Then the productions of φ_{ext} are (for each $k \geqslant 1$):

$$a \to a\hat{b}^k c, \qquad d \to a\hat{a}^k d,$$

$$b \to b\hat{a}^k d, \qquad \hat{a} \to \hat{a}\hat{b}^k c,$$

$$c \to b\hat{b}^k c, \qquad \hat{b} \to \hat{b}\hat{a}^k d.$$

Hence φ_{ext} is a nonoverlapping code that is not strict.

(2) Consider the nonoverlapping code (taken from [3]) $\varphi = (\{\ell_1, \ell_2, \ell_3, r_1, r_2, r_3\}, P, \ell_1)$ where P consists of the productions, for $k \geqslant 1$,

$$\ell_1 \to \ell_1 r_3^k r_1, \qquad r_1 \to \ell_2 r_1^k r_3,$$

$$\ell_2 \to \ell_1 r_3^k r_2, \qquad r_2 \to \ell_3 r_2^k r_1,$$

$$\ell_3 \to \ell_2 r_1^k r_2, \qquad r_3 \to \ell_3 r_2^k r_3.$$

This code is an example of a nonoverlapping code that is not an extension code. This is easily seen by the fact that each (left- or right-) extension of a binary strict code introduces at most two letters that can occur as "middle" letters.

(3) Consider the strict code $\varphi = (\{\ell_1, \ell_2, \ell_3, m, r_2, r_3\}, P, \ell_3)$ where P consists of the productions, for $k \geqslant 1$,

$$m \to \ell_1 m^k r_1, \qquad \ell_3 \to \ell_1 m^k r_2,$$

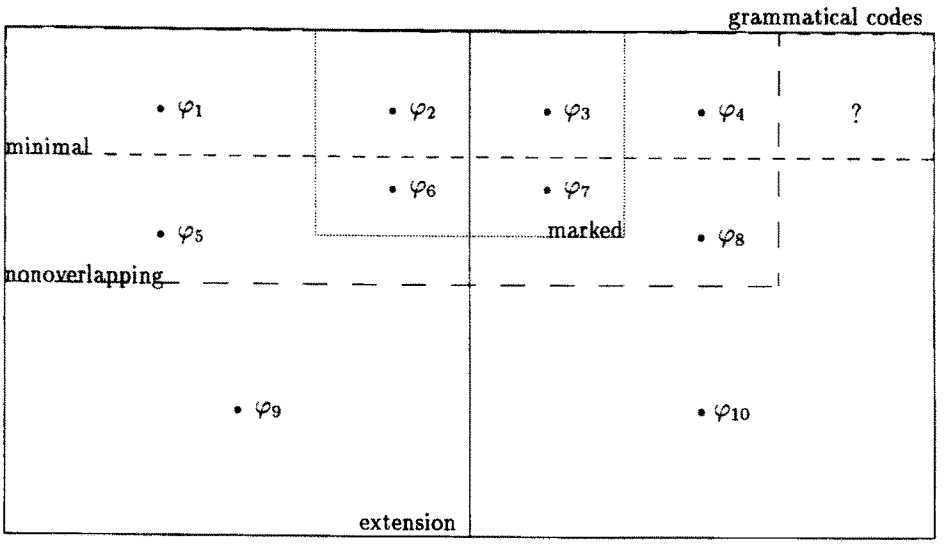
$$\ell_1 \to \ell_2 m^k r_1, \qquad r_1 \to \ell_2 m^k r_2,$$

$$\ell_2 \to \ell_3 m^k r_1, \qquad r_3 \to \ell_3 m^k r_2.$$

This code is an example of a strict code that is not an extension code. Note that if it would be an extension code, then it would be an extension introducing m and one of the ℓ_j as new letters. By Definition 3.3 this new letter ℓ_j would be ℓ_1, since $m \to \ell_1 r_1$ is a production of φ. But then also ℓ_2 would be a new letter, since $\ell_1 \to \ell_2 r_1$ is a production of φ; contradiction.

There is an easy way to construct codes that are not extensions, by choosing the axiom in such a way that it must be a new letter. However, intuitively, the axiom is not crucial to the nature of a grammatical code. Hence, we prefer to give examples of codes that are not extensions for more essential reasons.

Summarizing, we obtain the inclusion diagram of Fig. 4 for grammatical codes of chain-free trees. The question mark denotes that we do not know whether this area is



extensions of strict binary codes = minimal codes ∩ extension codes
extensions of marked binary codes = nonoverl. codes ∩ extension codes
strict codes = marked codes ∩ minimal codes

Fig. 4. Inclusion diagram.

empty or not (cf. the discussion at the end of Section 2). For the subclass of extension codes we *do* have that every minimal code is nonoverlapping: if φ_{ext} is an extension code with 6 letters, then it is obtained from a binary code φ of 4 letters; by Theorem 2.7 φ is strict, and hence, by Theorem 3.8, φ_{ext} is nonoverlapping.

For the sake of completeness we give representative codes corresponding with the diagram:

- φ_1 is the code φ_{ext} from Example 3.9(1),
- φ_2 is the code φ_{ext} from Example 3.5,
- φ_3 is the code φ from Example 3.9(3),
- φ_4 is the code from Example 3.9(2),
- φ_5 is the extension of the marked binary code with productions $a \to ad$, $b \to bd$, $c \to ce$, $d \to ae$, and $e \to be$,
- φ_6 is the extension of the marked binary code with productions $a \to ad$, $b \to bd$, $c \to be$, $d \to cd$, and $e \to ce$,
- φ_7 is the marked code with productions, for $k \geqslant 0$, $a \to ac^k e$, $b \to bc^k e$, $c \to ac^k f$, $d \to bc^k f$, $e \to ac^k g$, $f \to bc^k f$, and $g \to bc^k d$,
- φ_8 is the nonoverlapping code with productions, for $k \geqslant 0$, $a \to ad^k e$, $b \to bd^k e$, $c \to ad^k f$, $d \to ce^k d$, $e \to ce^k f$, $f \to bd^k g$, $g \to ad^k g$,
- φ_9 is the extension of the binary code of Example 2.8,
- φ_{10} is the code with productions, for $k \geqslant 0$, $a \to af^k e$, $b \to bf^k d$, $c \to af^k c$, $d \to ef^k d$, $e \to bf^k c$, $f \to af^k g$, $g \to bf^k g$.

The fact that φ_7, φ_8, and φ_{10} are not extensions can be shown by arguments similar to the ones used in Examples 3.9(2) and (3). The unambiguity of φ_{10} follows by reasoning as in Example 2.8.

References

[1] A. Ehrenfeucht, J. Engelfriet, P. ten Pas and G. Rozenberg, Grammatical codes of trees and terminally coded grammars, *Fund. Inform.*, to appear.
[2] A. Ehrenfeucht and G. Rozenberg, Grammatical codes of trees, *Discrete Appl. Math.* **32** (1991) 103–129.
[3] A. Ehrenfeucht, P. ten Pas and G. Rozenberg, Properties of grammatical codes of trees, *Theoret. Comput. Sci.* **125** (1994) 259–293.
[4] D.E. Knuth, *The Art of Computer Programming, Vol 1*: Fundamental Algorithms (Addison-Wesley, Reading, MA, 1973).