Communications in Information and Systems
Vol. 2, No. 1, pp. 29-52, June 2002

STRUCTURED GRAMMAR-BASED CODES FOR UNIVERSAL LOSSLESS DATA COMPRESSION*

JOHN KIEFFER† AND EN-HUI YANG‡

Abstract. A grammar-based code losslessly compresses each finite-alphabet data string x by compressing a context-free grammar G_x which represents x in the sense that the language of G_x is $\{x\}$. In an earlier paper, we showed that if the grammar G_x is a type of grammar called irreducible grammar for every data string x, then the resulting grammar-based code has maximal redundancy/sample $O(\log \log n / \log n)$ for n data samples. To further reduce the maximal redundancy/sample, in the present paper, we first decompose a context-free grammar into its structure and its data content, then encode the data content conditional on the structure, and finally replace the irreducible grammar condition with a mild condition on the structures of all grammars used to represent distinct data strings of a fixed length. The resulting grammar-based codes are called *structured grammar-based codes*. We prove a coding theorem which shows that a structured grammar-based code has maximal redundancy/sample $O(1/\log n)$ provided that a weak regular structure condition is satisfied.

Keywords. lossless data compression, universal codes, redundancy, context-free grammars, grammar-based codes

1. Introduction.

Universal lossless source coding first arose in the late 1960's, becoming systematized with Davisson's seminal 1973 paper [2]. A universal lossless code is a lossless source code which is asymptotically optimal for all finite-state information sources on a given finite alphabet. The most famous class of universal lossless codes consists of the Lempel-Ziv codes (LZ77 [13], LZ78 [14], and their many variants). In the 20+ years since the Lempel-Ziv codes were introduced, other classes of universal lossless codes have been proposed and analyzed. One of these classes is the class of grammar-based codes [3].

A grammar-based code can be thought of as a type of transform code. It losslessly encodes/decodes a finite-alphabet data string x according to the following four stages:

- *Analysis Stage:* A context-free grammar G consisting of a set of production rules is found which represents x in the sense that the language $L(G)$ of the grammar G is $\{x\}$, where the language $L(G)$ is simply the set of all sequences derived from G.
- *Encoding Stage:* A binary codeword $B(G)$ from which the grammar G can

*Invited paper; Received on February 19, 2002, accepted for publication on June 6, 2002. This work was supported by National Science Foundation Grants NCR-9508282 and CCR-9902081, by the Natural Sciences and Engineering Research Council of Canada under Grants RGPIN203035-98 and RGPIN203035-02, by the Premier's Research Excellence Award, and by the Canada Research Chairs Program.

†Department of Electrical & Computer Engineering, University of Minnesota, Room 4-174 EE/CSci Bldg., 200 Union Street SE, Minneapolis, MN 55455, USA. E-mail: kieffer@ece.umn.edu

‡Department of Electrical & Computer Engineering, University of Waterloo, Waterloo, Ontario, CA N2L 3G1. E-mail: ehyang@bbcr.uwaterloo.ca

be reconstructed is formed and transmitted to the decoder.

- *Decoding Stage:* The grammar G is reconstructed from the codeword $B(G)$.
- *Synthesis Stage:* The data string x is "grown" from the production rules of G.

The most important of these four stages is the Analysis Stage, because the other three stages are determined from it. In the paper [3], the grammar representing the data string in the Analysis Stage was taken to be an *irreducible grammar*, a type of grammar that had been used by previous workers in the applications of context-free grammars. (Irreducible grammars are discussed thoroughly in [3]; the reader does not need to understand what an irreducible grammar is for the purposes of this paper.) Grammar-based codes based upon the use of irreducible grammars in the Analysis Stage were shown in [3] to exhibit maximal redundancy/sample $O(\log\log n/\log n)$, where the parameter n is the number of data samples.

Since the paper [3] appeared, the natural question has been whether a different type of grammar could be used in the Analysis Stage, which would bring about grammar-based codes with maximal redundancy/sample $O(1/\log n)$. This question is of interest because

(i) the Lempel-Ziv codes (possibly the most popular class of universal lossless source codes) have not yet been shown to exhibit maximal redundancy/sample $O(1/\log n)$, although LZ78 is known to have maximal redundancy/sample $O(\log\log n/\log n)$ [6], and

(ii) the maximal redundancy/sample of the context-tree weighting algorithm (CTW)[10], [11] and the prediction by partial match algorithm (PPM) [1] is bounded below by a positive constant for all large n, let alone any convergence rate.

(The notion of the maximal redundancy/sample is much stronger than the usual definition of redundancy against the so-called tree sources[9], [10], [11]. In terms of the latter weak definition of redundancy, CTW is known to have $O(\log n/n)$ redundancy/sample.) The present paper settles this question in the affirmative.

Our approach is to use *structured grammars* in the Analysis Stage. The precise details concerning the notion of a structured grammar shall be presented later in this paper. In this introduction, we can give an intuitive feeling for this concept. Roughly speaking, a context-free grammar G representing x can be decomposed into two parts: the structure of G and the data content of G. The structure of G is related to the *derivation tree* of G—a tree via which the data string x can be grown from the production rules of G. If x has n entries, then the derivation tree of G will have n leaf vertices labeled with the entries of x from left to right (the internal vertices of the derivation tree are labeled with variables of G). The data content of G is uniquely determined by the structure of G and the data string x itself. In [3], [12], the grammar G is encoded without decomposing G into its structure and data content. The grammar encoding methods presented in [3], [12] can be applied to any context-free grammar. In particular, the binary codewords $B(G)$ for all context-free grammars G form a prefix set. However, this generality does indicate that there is room to improve as far as the compression of x is concerned. Since different grammars can represent the same string and different grammars representing different strings can have the same structure, the improvement of compression efficiency of x can be made if we first decompose G into its structure and data content, then encode its structure, and finally encode its data content conditional on its structure. By imposing certain mild conditions on the structure of each grammar used in the Analysis Stage to represent a data string of a given length, the encoding of the structure is either free or has a negligible overhead, and the dominating term is the encoding of the data content conditional on the structure, thereby improving the compression performance and reducing the redundancy from $O(\log\log n/\log n)$ to $O(1/\log n)$. The resulting grammar-based codes with the new encoding method are called the *structured grammar-based codes.* In a special case, one may require that all grammars used in the Analysis Stage to represent all distinct data strings of a given length have the same structure.

1.1. Terminology. We present terminology and notation to be used throughout the rest of the paper. Since we have to start somewhere, we assume that the reader has had some previous exposure to the concept of a context-free grammar. (Those readers who work in pattern recognition, machine intelligence, image processing, or many other areas will already have some familiarity with this concept. Readers having less familiarity are encouraged to consult the paper [3].)

- $\mathcal{S}^+$ denotes the set of all strings $s_1s_2\cdots s_k$ in which $s_1,\cdots,s_k$ are $1\le k<\infty$ entries from set $\mathcal{S}$.
- $*$ denotes the concatenation operation in $\mathcal{S}^+$.
- $|\mathcal{S}|$, $card(\mathcal{S})$ denote cardinality of set $\mathcal{S}$.
- $|x|$ denotes the length of string x.
- $V(G)$ denotes the set of variables of a grammar G.
- $L(G)$ denotes the language of a grammar G.
- $|G|$ denotes the total number of elements appearing in the right members of the production rules of a grammar G.
- $L(v|T)$ denotes the number of leaf vertices of a tree T which are equal or subordinate to vertex v of T.
- All logarithms are to base 2.

2. Bracketed Expressions and Their Trees. A context-free grammar G is said to represent a string x if $L(G)=\{x\}$. In the next section, we shall derive the grammars that we shall use to represent the data strings that we wish to compress. The most convenient way for us to derive these grammars will be through the use of fully bracketized expressions. This section is devoted to the presentation of useful material on fully bracketized expressions and the trees associated with them. As shown below, fully bracketized expressions are related to multilevel refined parsing.

Let $\mathcal{S}$ be a finite set. The set $Br(\mathcal{S})$ of fully bracketized expressions over $\mathcal{S}$ is defined by

$$Br(\mathcal{S}) \triangleq Br_{at}(\mathcal{S}) \cup Br_{nat}(\mathcal{S}), \tag{2.1}$$

where

- $Br_{at}(\mathcal{S}) = \{[s] : s \in \mathcal{S}\}$; and
- $Br_{nat}(\mathcal{S})$ is the smallest subset U of $(\mathcal{S} \cup \{[,]\})^+$ satisfying the property that $[s_1 s_2 \cdots s_k] \in U$ whenever $k \geq 2$ and $s_1, s_2, \cdots, s_k$ are members of $U \cup \mathcal{S}$.

The members of $Br(\mathcal{S})$ shall be called $\mathcal{S}$-expressions, the members of $Br_{at}(\mathcal{S})$ shall be called atomic $\mathcal{S}$-expressions, and the members of $Br_{nat}(\mathcal{S})$ shall be called nonatomic $\mathcal{S}$-expressions. For each left bracket in an $\mathcal{S}$-expression, there is a right bracket that is paired with it. If an $\mathcal{S}$-expression σ consists of $n + k$ entries, with n of the entries belonging to $\mathcal{S}$ and k of the entries being brackets, then there corresponds to σ a rooted tree $T(\sigma)$ having exactly n leaf vertices and $k/2$ internal vertices; the tree $T(\sigma)$ is uniquely characterized by the following properties:

(i): There is a one-to-one correspondence between the n $\mathcal{S}$-filled positions in σ and the n leaf vertices of $T(\sigma)$. Letting $x_1, x_2, \cdots, x_n$ denote the left-to-right entries of σ which belong to $\mathcal{S}$, the correspondence is made clear by labeling the i-th left-to-right leaf vertex of $T(\sigma)$ with x_i $(i = 1, 2, \cdots, n)$.

(ii): There is a one-to-one correspondence between the $k/2$ left-right bracket pairs in σ and the $k/2$ internal vertices of $T(\sigma)$. The vertex in $T(\sigma)$ corresponding to a given left-right bracket pair in σ is the vertex whose leaf vertex successors correspond, according to *(i)*, to the $\mathcal{S}$-filled positions in σ that lie between the left bracket and paired right bracket.

For each atomic $\mathcal{S}$-expression, the corresponding tree is trivial, consisting of root vertex and one leaf vertex. For each nonatomic $\mathcal{S}$-expression σ, the tree $T(\sigma)$ has at least two children for each internal vertex (see Fig. 1); conversely, every finite rooted tree which carries a label from $\mathcal{S}$ on each leaf vertex and which has two or more children for each internal vertex corresponds to a unique nonatomic $\mathcal{S}$-expression.

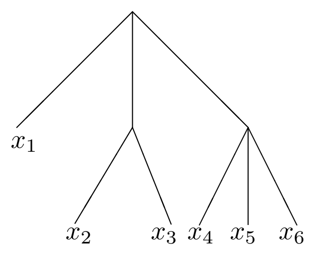


FIG. 1. *Tree $T(\sigma)$ for the expression $\sigma = [x_1[x_2x_3][x_4x_5x_6]]$.*

For each positive integer n, let $Br^n(\mathcal{S})$ be the set of all $\mathcal{S}$-expressions which have exactly n $\mathcal{S}$-filled positions. Let $\{s(n) : n = 1, 2, \cdots\}$ be the sequence

$$s(n) = \begin{cases} 1, & n = 1 \\ n^{-1} \sum_{k=0}^{n-2} \binom{2n-k-2}{n-1}\binom{n-2}{k}, & n \geq 2 \end{cases}$$

The numbers $s(n)$ are called little Schröder numbers [7] [8]. The little Schröder numbers are important to us here because

$$|Br^n(\{0\})| = s(n), \quad n \geq 1 \tag{2.2}$$

$$|Br^n(\mathcal{S})| = |\mathcal{S}|^n s(n), \quad n \geq 1 \tag{2.3}$$

The relation (2.2) is well known [8, Ch. 6]; the relation (2.3) becomes clear by noticing that each entry of a $\{0\}$-expression where 0 occurs can be regarded as a placeholder for an element of $\mathcal{S}$. The first few little Schröder numbers are given in Table 1.

Table 1
Little Schröder Numbers

n	$s(n)$	n	$s(n)$
1	1	6	197
2	1	7	903
3	3	8	4279
4	11	9	20793
5	45	10	103049

Example 1: The $s(4) = 11$ expressions in $Br^4(\{0\})$ are seen to be [[00][00]], [0000], [0[00]0], [[00]00], [00[00]], [[000]0], [0[000]], [[0[00]]0], [[[00]0]0], [0[[00]0]], [0[0[00]]].

A *subexpression* of an $\mathcal{S}$-expression σ is defined to be any $\mathcal{S}$-expression which occurs as a substring of σ. Let σ be a fixed $\mathcal{S}$-expression. The occurrences of subexpressions in σ are in one-to-one correspondence with the internal vertices of $T(\sigma)$, since each subexpression begins with a left bracket and ends with the matching right bracket. For example, the occurrence of $[x_4x_5x_6]$ as a subexpression of $\sigma = [x_1[x_2x_3][x_4x_5x_6]]$ corresponds to the vertex of $T(\sigma)$ in Fig. 1 whose children are labeled x_4, x_5, x_6.

3. Representational Grammars. Let $\mathcal{A}$ be a fixed finite alphabet. We shall call the members of $\mathcal{A}^+$ $\mathcal{A}$-strings. Our goal in this paper is to efficiently compress each $\mathcal{A}$-string via the grammar-based approach. In order that we may do this, we put forth in this section a set $\mathcal{G}(rep)$ of context-free grammars called *representational grammars* which satisfy

(i): The language $L(G)$ of each grammar $G \in \mathcal{G}(rep)$ consists of a unique string, and that string is an $\mathcal{A}$-string.

(ii): For each $\mathcal{A}$-string x, there is at least one grammar $G \in \mathcal{G}(rep)$ which represents x (in the sense that $L(G) = \{x\}$).

In the paper [3], we used context-free grammars which we called admissible grammars to represent data strings in the sense of (i)-(ii) above. Simply put, an admissible grammar is any context-free grammar G for which the language generated by G is a singleton. The class of representational grammars introduced in this section is a proper subclass of the class of admissible grammars. Since writing [3], we have come to realize that the class of representational grammars is more suitable for data compression purposes than the class of admissible grammars.

A context-free grammar G is uniquely specified by defining the following four entities:

(i): The set $V(G)$ of variables of G.
(ii): The start variable of G.
(iii): The set of terminal symbols of G.
(iv): The set of production rules of G.

Let σ be a fixed (but arbitrary) $\mathcal{A}$-expression. In the rest of this paragraph, we describe how to build from σ a unique context-free grammar $G_{rep}(\sigma)$. Recall from the end of Sec. 2 how a subexpression of σ corresponds to each internal vertex of the tree $T(\sigma)$. Traverse each internal vertex of $T(\sigma)$ in the top-down left-to-right order (i.e., the breadth-first order), appending the subexpression corresponding to that vertex to a list if that subexpression has not appeared previously in the list (start with the empty list). Let

$$\sigma_0, \sigma_1, \cdots, \sigma_t$$

be the final list of distinct subexpressions of σ after all of the internal vertices of $T(\sigma)$ have been traversed. The set of variables of $G_{rep}(\sigma)$ is

$$V(G_{rep}(\sigma)) = \{A_0, A_1, A_2, \cdots, A_t\},$$

where each A_i is an abstract symbol not belonging to the data alphabet $\mathcal{A}$. The start variable of $G_{rep}(\sigma)$ is A_0. The set of terminal symbols of $G_{rep}(\sigma)$ is the set consisting of those symbols in $\mathcal{A}$ that appear in σ. For each $i = 0, 1, \cdots, t$, there is exactly one production rule

$$A_i \rightarrow \alpha_1 \alpha_2 \cdots \alpha_k \tag{3.1}$$

of $G_{rep}(\sigma)$ whose left member is A_i, obtained as follows. First, form the unique factorization

$$\sigma_i = [s_1 s_2 \cdots s_k],$$

in which $s_1, s_2, \cdots, s_k$ are members of $\mathcal{A} \cup Br_{nat}(\mathcal{A})$. If $s_i \in \mathcal{A}$, then α_i in (3.1) is taken to be s_i. If $s_i \in Br_{nat}(\mathcal{A})$, then α_i is taken to be A_j, where j is the unique integer such that $\sigma_j = s_i$.

We now formally define the set of representational grammars by

$$\mathcal{G}(rep) \overset{\Delta}{=} \{G_{rep}(\sigma) : \sigma \in Br(\mathcal{A})\}$$

We list properties of representational grammars easily deduced from the construction given in the preceding paragraph.

Properties of Representational Grammars.

Prop 1: For each representational grammar G, there is exactly one $\mathcal{A}$-expression σ such that $G = G_{rep}(\sigma)$.

Prop 2: The language of a representational grammar $G_{rep}(\sigma)$ is $\{x\}$, where x is the $\mathcal{A}$-string consisting of those entries of σ belonging to $\mathcal{A}$ (taken left-to-right).

Prop 3: The unique derivation tree of a representational grammar $G_{rep}(\sigma)$ yields $T(\sigma)$ when the labels on the internal vertices are removed.

Prop 4: For each positive integer n and each $\mathcal{A}$-string x of length n, there are exactly $s(n)$ representational grammars which represent x.

Example 2: Let $\mathcal{A} = \{0, 1\}$, and we pick

$$\sigma = [[[01][01]][[11][01]][[11][01]]] \tag{3.2}$$

as an $\mathcal{A}$-expression for which we construct the grammar $G_{rep}(\sigma)$. There are five subexpressions of σ, to which we assign variables as follows:

$$\begin{aligned}
A_0 &\leftrightarrow [[[01][01]][[11][01]][[11][01]]] \\
A_1 &\leftrightarrow [[01][01]] \\
A_2 &\leftrightarrow [[11][01]] \\
A_3 &\leftrightarrow [01] \\
A_4 &\leftrightarrow [11]
\end{aligned} \tag{3.3}$$

The production rules of our grammar $G_{rep}(\sigma)$ are then seen to be:

$$\begin{aligned}
A_0 &\rightarrow A_1A_2A_2 \\
A_1 &\rightarrow A_3A_3 \\
A_2 &\rightarrow A_4A_3 \\
A_3 &\rightarrow 01 \\
A_4 &\rightarrow 11
\end{aligned} \tag{3.4}$$

The derivation tree of $G_{rep}(\sigma)$ is given in Fig. 2. From the labels on the leaf vertices of the derivation tree, we see that

$$L(G_{rep}(\sigma)) = \{010111011101\}.$$

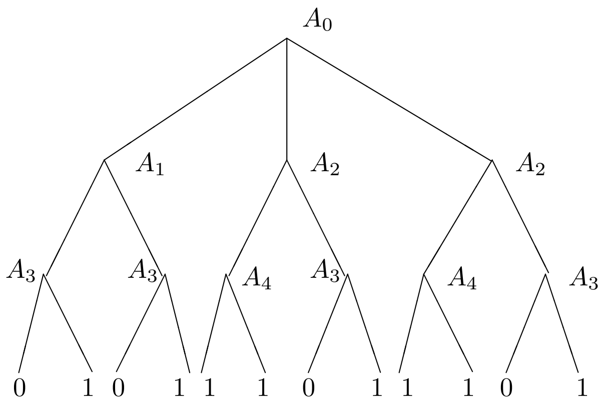


FIG. 2. *Derivation tree of $G_{rep}(\sigma)$ in Example 2.*

Removing the labels on the internal vertices of the derivation tree, the tree $T(\sigma)$ can be seen to result.

Example 3: For an atomic $\mathcal{A}$-expression $[a]$, the grammar $G_{rep}([a])$ has only one production rule $A_0 \to a$.

4. Structure Grammars. We put forth in this section a set of context-free grammars $\mathcal{G}(str)$ called *structure grammars.* We shall see that there is a useful interplay between the grammars in $\mathcal{G}(rep)$ and the grammars in $\mathcal{G}(str)$.

In the previous section, we explained how to form the grammar $G_{rep}(\sigma)$ for any $\mathcal{A}$-expression σ. By the same technique, we form a unique grammar corresponding to each $\{0\}$-expression, the only difference being that we denote the set of variables of the resulting grammar by

$$\{U_0, U_1, \cdots, U_t\}$$

instead of

$$\{A_0, A_1, \cdots, A_t\},$$

where the U_i's are special symbols reserved for denoting variables of structure grammars. The grammar corresponding to $\sigma \in Br(\{0\})$ formed in this way shall be denoted $G_{str}(\sigma)$. The set of structure grammars can now be formally defined as

$$\mathcal{G}(str) = \{G_{str}(\sigma) : \sigma \in Br(\{0\})\}.$$

If σ is an $\mathcal{A}$-expression, and σ' is the $\{0\}$-expression that arises by changing to 0 each entry of σ which belongs to $\mathcal{A}$, then we henceforth write $\sigma \to \sigma'$ to denote this fact. If $G = G_{rep}(\sigma)$ is a representational grammar, then we define G^* to be the grammar $G_{str}(\sigma')$ for which $\sigma \to \sigma'$. We call G^* the *structure grammar of* G. Given any representational grammar G, there is a natural mapping $\phi_G : V(G) \cup \mathcal{A} \to V(G^*) \cup \{0\}$ defined as follows. Let σ, σ' be the bracketed expressions such that $G = G_{rep}(\sigma)$

and $G^* = G_{str}(\sigma')$. Let $A_i \leftrightarrow \sigma_i$ be the correspondence between variables A_i of G and subexpressions σ_i of σ that was used in defining the grammar G, and let $U_j \leftrightarrow \sigma'_j$ be the correspondence between variables U_j of G^* and subexpressions σ'_j of σ' that was used in defining the grammar G^*. If $A_i \in V(G)$, we define $\phi_G(A_i) = U_j$, where $U_j \in V(G^*)$ is the variable of G^* such that $\sigma_i \to \sigma'_j$. If $a \in \mathcal{A}$, we define $\phi_G(a) = 0$.

The proof of the following simple lemma is omitted.

LEMMA 1. *Let G be any representational grammar. If*

$$A_i \to \alpha_1\alpha_2\cdots\alpha_k$$

is any production rule of G, then

$$\phi_G(A_i) \to \phi_G(\alpha_1)\phi_G(\alpha_2)\cdots\phi_G(\alpha_k)$$

is a production rule of G^*. Conversely, every production rule of G^* is mapped onto by at least one production rule of G in this way.

Example 4: Let G be the representational grammar of Example 2. From the correspondences (3.3), we see that G^* must have three variables U_0, U_1, U_2 with correspondences

$$\begin{aligned} U_0 &\leftrightarrow [[[00][00]][[00][00]][[00][00]]] \\ U_1 &\leftrightarrow [[00][00]] \\ U_2 &\leftrightarrow [00]. \end{aligned} \tag{4.1}$$

(Simply change every 1 to 0 on the right sides of (3.3) and eliminate duplications.) The mapping ϕ_G must map the set $\{A_0, A_1, A_2, A_3, A_4, 0, 1\}$ onto the set $\{U_0, U_1, U_2, 0\}$. We see that ϕ_G is specified by

$$\begin{aligned} \phi_G(A_0) &= U_0 \\ \phi_G(A_1) &= U_1 \\ \phi_G(A_2) &= U_1 \\ \phi_G(A_3) &= U_2 \\ \phi_G(A_4) &= U_2 \\ \phi_G(0) &= 0 \\ \phi_G(1) &= 0 \end{aligned}$$

by seeing how the correspondences (4.1) arose from the correspondences (3.3). Applying the mapping ϕ_G to the production rules of G in (3.4), we automatically obtain the following production rules of G^* via Lemma 1:

$$\begin{aligned} &U_0 \to U_1U_1U_1 \\ &U_1 \to U_2U_2 \\ &U_2 \to 00 \end{aligned}$$

The structure grammar G^* of G represents the structural information of G. From the above, it follows that one can easily get G^* from G through the mapping ϕ_G. On the other hand, if x is the data string represented by G, then one can uniquely determine G from G^* and x. In the next section, we will exploit this relationship to define the unnormalized conditional entropy $H(G|G^*)$ of the representational grammar G given its structure grammar G^* and conditionally encode G given G^*. If encoder and decoder know the grammar G^*, it will be possible for the encoder to encode the grammar G for perfect recovery by the decoder using approximately $H(G|G^*)$ code bits.

To conclude this section, we present a structure grammar concept that will be useful to us later on. Let G be any structure grammar. We define the *spreading factor* $\beta(G)$ of G as follows. Letting T be the derivation tree of G, $\beta(G)$ is the largest ratio $L(v|T)/L(v'|T)$ as (v, v') ranges through all parent-child vertex pairs of T. The spreading factor $\beta(G)$ measures, to some degree, how children from an internal vertex are spread.

Example 5: Let G be the structure grammar presented in Example 4. Its derivation tree T, stripped of all labels, coincides with the tree in Fig. 2 when it is stripped of all labels. When v is the root of T and v' is any of the children of v, we see that $L(v|T) = 12$ and $L(v'|T) = 4$. The parent-child pair (v, v') yields the largest ratio $L(v|T)/L(v'|T)$; hence $\beta(G) = 12/4 = 3$.

The following result is proved in Appendix A.

THEOREM 1. *Let n be any integer ≥ 2 and let G be any representational grammar which represents an $\mathcal{A}$-string of length n. Then*

$$|G| \leq 72\beta(G^*)^2(|\mathcal{A}|+2)^2 \log(|\mathcal{A}|+2)\left(\frac{n}{\log n}\right).$$

5. Conditional Grammar Encoding.

Our first task in this section is to define the unnormalized conditional entropy $H(G|G^*)$ of any representational grammar G given its structure grammar G^*.

DEFINITION: Let $S = s_1 s_2 \cdots s_k$ be any nonempty string of finite length over any alphabet. We define the (unnormalized) entropy of the string S by

$$H(S) \triangleq \sum_{i=1}^{k} -\log p(s_i),$$

where for any $s \in \{s_1, \cdots, s_k\}$,

$$p(s) = k^{-1} card(\{1 \leq i \leq k : s_i = s\}).$$

If S is the empty string, define $H(S) = 0$. Let $U = u_1 u_2 \cdots u_k$ be any string of the same length as S. Let $\mathcal{U} = \{u_1, \cdots, u_k\}$, and for each $u \in \mathcal{U}$, let $S(u)$ be the substring obtained from S by removing each entry s_i of S for which $u_i \neq u$. We define the unnormalized conditional entropy of S given U as

$$H(S|U) \triangleq \sum_{u \in \mathcal{U}} H(S(u)).$$

From information theory, it is easy to see that $H(S|U) \leq H(S)$.

Let G be a fixed (but arbitrary) representational grammar. We are now ready to define $H(G|G^*)$. First, letting $V(G) = \{A_0, A_1, \cdots, A_t\}$, form the string

$$\alpha(A_0) * \alpha(A_1) * \cdots * \alpha(A_t), \tag{5.1}$$

where $\alpha(A_i)$ denotes the string in $(V(G) \cup \mathcal{A})^+$ which is the right member of the production rule of G whose left member is A_i. Let $\omega_G = \omega_1\omega_2 \cdots \omega_k$ be the string in $(V(G) \cup \mathcal{A})^+$ formed from the string (5.1) by striking from this string the first left-to-right appearance of each variable in $V(G)$. Then the unnormalized conditional entropy $H(G|G^*)$ of G given its structure grammar G^* is defined as

$$H(G|G^*) \triangleq H(\omega_1\omega_2 \cdots \omega_k | \phi_G(\omega_1)\phi_G(\omega_2) \cdots \phi_G(\omega_k)).$$

Example 6: Let G be the representational grammar introduced in Example 2. We compute $H(G|G^*)$. From the production rules (3.4), we see that

$$\omega_G = A_2A_3A_30111. \tag{5.2}$$

We can apply the mapping ϕ_G derived in Example 4 to each term on the right side of (5.2), from which we conclude that

$$H(G|G^*) = H(A_2A_3A_30111|U_1U_2U_20000) \approx 3.25.$$

Remark. The quantity $H(G) \triangleq H(\omega_G)$ was defined in [3] as the definition of the unnormalized entropy of a representational grammar G. It was shown in [3] that without being decomposed into its structure G^* and its data content, G can be encoded by a prefix code into a codeword of roughly $H(G)$ bits. However, this encoding method is not efficient because

(a) given a data sequence of length n, it follows from (2.2) and Section 4 that there are $s(n)$ distinct representational grammars representing x and yet in the Analysis Stage one needs just one grammar for each distinct string x; and

(b) grammars representing distinct strings may have the same structure grammar.

As a result, a better way is to conditionally encode G given G^*. The following theorem says that given G^*, G can be losslessly encoded into a codeword of roughly $H(G|G^*)$ bits. By imposing some mild condition on G^*, the encoding of G^* is either free or has a negligible overhead. Since $H(G|G^*) \leq H(G)$, the compression performance is improved, as shown in the next section. This is the essence of structured grammar-based coding.

Theorem 2. *Let G' be any structure grammar. There are binary strings $\{B(G|G') : G \in \mathcal{G}(rep),\ \ G^* = G'\}$ such that*

- *For each representational grammar G whose structure grammar is G',*

$$|B(G|G')| \leq H(G|G') + 4|G| + |\mathcal{A}|.$$

- *For each pair of distinct representational grammars G_1, G_2 whose structure grammar is G', the string $B(G_1|G')$ is not a prefix of the string $B(G_2|G')$.*

Proof of Theorem 2. Let G' be a fixed structure grammar. Let G be any representational grammar such that $G^* = G'$. Let

$$V(G) = \{A_0, A_1, \cdots, A_K\}$$

$$V(G') = \{U_0, U_1, \cdots, U_J\}$$

We show how to construct a binary codeword $B(G|G')$ such that G will be recoverable from $B(G|G')$ and G'. By superimposing the derivation tree of G over the derivation tree of G', a tree T is obtained such that each of its vertices carries a label from $V(G) \cup \mathcal{A}$ (which we call the G-label of the vertex) and also carries a label from $V(G') \cup \{0\}$ (which we call the G'-label of the vertex). The tree T has the following properties

(1): The root vertex of T carries G-label A_0 and G'-label U_0.
(2): If the G-label of a vertex of T is A_i and the G-labels of its children are $a_1, a_2, ..., a_r$ from left to right, then $A_i \to a_1 a_2 \cdots a_r$ is a production rule of G.
(3): If the G'-label of a vertex of T is U_j and the G'-labels of its children are $b_1, b_2, \cdots, b_s$, then $U_j \to b_1 b_2 \cdots b_s$ is a production rule of G'.

We now prune the tree T according to the following procedure:
Step 1 Traverse the internal vertices of T in the breadth-first order.
Step 2 If the G-label of the currently traversed internal vertex appears before, cut the subtree rooted at this internal vertex except this internal vertex itself.
Step 3 Continue to traverse the remaining internal vertices of the pruned tree in the breadth-first order.
Step 4 Repeat Steps 2 and 3 until all the remaining internal vertices are traversed.
(The above procedure is illustrated in Example 7 after this proof.) After pruning T, one eventually obtains a tree $T(G|G')$ such that (i) $T(G|G')$ possesses exactly $|G|$ edges and $|V(G)|$ internal vertices; (ii) the G-labels on the internal vertices of $T(G|G')$ are the variables in $V(G)$. Let B_1 be a binary codeword of length $2|G|$ from which the tree $T(G'|G)$, without labels, can be recovered (an internal vertex of $T(G'|G)$ with k children will generate $2k$ consecutive bits in B_1—k bits to tell how many children the vertex has followed by k bits to tell which of these children are internal vertices and which are leaf vertices). The codeword B_1 will appear at the beginning of the codeword $B(G|G')$. After the decoder has recovered the tree $T(G|G')$ without labels from B_1, it can then insert the G-labels on the internal vertices of $T(G|G')$ (because

of the numbering convention with which the variables in $V(G)$ were generated), and it can also insert the G'-labels on all of the vertices of $T(G|G')$ (since the decoder knows G'). At this point, the decoder will not know the G-labels on the leaf vertices of $T(G|G')$ (once these are known, G is determined). However, the decoder can determine the mapping ϕ_G from $V(G) \cup \mathcal{A}$ into $V(G') \cup \{0\}$ (by seeing what label from $V(G')$ is on each internal vertex of $T(G|G')$). Let B_2 be a binary codeword of length $|\mathcal{A}| + |G|$ which gives the frequency of each member of $V(G) \cup \mathcal{A}$ in the right members of the production rules of G (the first $|\mathcal{A}|$ bits of B_2 identify those members of $\mathcal{A}$ which are terminal symbols of G; in the remaining $|G|$ bits, each frequency f of a symbol lying in the right members of the production rules of G is represented by f bits). The codeword B_2 appears right after the codeword B_1 at the beginning of $B(G|G')$. For each $U \in V(G') \cup \{0\}$, let $S(U)$ be the (possibly empty) sequence of G-labels that appear on the leaf vertices of $T(G|G')$ whose G'-label is U. From the mapping ϕ_G that the decoder learns from B_1 and the frequencies that the decoder learns from B_2, the decoder will then know how long each sequence $S(U)$ is, what the alphabet of each $S(U)$ is, and what is the frequency with which each member of the alphabet of $S(U)$ appears in $S(U)$. Consequently, for each $U \in V(G') \cup \{0\}$ there is a binary codeword $B(U)$ of length $\lceil H(S(U)) \rceil$ which will allow the decoder to recover $S(U)$. Once the decoder knows each $S(U)$, the decoder will know the complete G-labeling of $T(G|G')$, and therefore will know G. Our argument has shown that we can take

$$B(G|G') = B_1 B_2 B(U_1) B(U_2) \cdots B(U_J) B(0).$$

The length of $B(G|G')$ is

$$3|G| + |\mathcal{A}| + \sum_{U \in V(G') \cup \{0\}} \lceil H(S(U)) \rceil. \tag{5.3}$$

The sum comprising the last term of (5.3) has no more than $|V(G')|$ nonzero terms; hence this last term is $\leq |V(G')| + H(G|G') \leq |G| + H(G|G')$. The conclusion of Theorem 2 is now established.

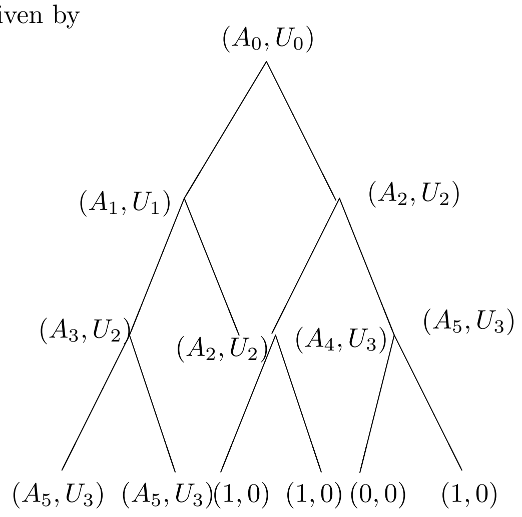
Example 7: Let G' be the structure grammar with production rules

$$\begin{aligned} U_0 &\to U_1 U_2 \\ U_1 &\to U_2 U_2 \\ U_2 &\to U_3 U_3 \\ U_3 &\to 00 \end{aligned}$$

and let G be the representational grammar with production rules

$$\begin{aligned} A_0 &\to A_1 A_2 \\ A_1 &\to A_3 A_2 \\ A_2 &\to A_4 A_5 \\ A_3 &\to A_5 A_5 \\ A_4 &\to 11 \\ A_5 &\to 01 \end{aligned} \tag{5.4}$$

The tree $T(G|G')$ is given by



where in the pair $(\bullet, \bullet)$ at each vertex, the first coordinate is the G-label and the second coordinate is the G'-label. (Note that the subtrees rooted at (A_2, U_2) of depth 2 and at (A_5, U_3) of depth 3 are deleted according to the pruning procedure mentioned in the proof of Theorem 2.) The codeword B_1 is

$$B_1 = 01 * 11 * 01 * 10 * 01 * 11 * 01 * 00 * 01 * 00 * 01 * 00$$

of length $24 = 2|G|$. The frequencies of A_1, A_2, A_3, A_4, A_5, 0, 1 in the right members of (5.4) are $1, 2, 1, 1, 3, 1, 3$, respectively. Thus,

$$B_2 = 11 * 1 * 01 * 1 * 1 * 001 * 1 * 001,$$

where the first two bits tell the decoder that both $0, 1$ in the alphabet $\mathcal{A} = \{0, 1\}$ are terminal symbols of G. The length of B_2 is $14 = |G| + |\mathcal{A}|$. The distinct G'-labels used on the leaf vertices of $T(G|G')$ are $U_2, U_3, 0$, and so the decoder can obtain the G-labels on these vertices from the sequences $S(U_2), S(U_3), S(0)$. These sequences are

$$\begin{aligned} S(U_2) &= A_2 \\ S(U_3) &= A_5 A_5 \\ S(0) &= 1101 \end{aligned}$$

From the G'-labels on the 7 leaf vertices of $T(G|G')$, the decoder determines that the sequences $S(U_2)$, $S(U_3)$, $S(0)$ are of lengths $1, 2, 4$, respectively. From the G'-labels on the internal vertices of $T(G|G')$, the decoder determines that the alphabets of $S(U_2), S(U_3)$ are subsets of $\{A_2, A_3\}, \{A_4, A_5\}$, respectively; the decoder knows that the alphabet of $S(0)$ is a subset of $\mathcal{A} = \{0, 1\}$. From the codeword B_2, the decoder determines that $S(U_2)$ consists of 1 A_2 and 0 A_3's and therefore $S(U_2) = A_2$; that $S(U_3)$ consists of 0 A_4's and 2 A_5's and therefore $S(U_3) = A_5A_5$; and that $S(0)$ consists of 3 zeroes and 1 one. Only $S(0)$ remains to be determined; since the alphabet of $S(0)$ is binary, and since the length of $S(0)$ coincides with $\lceil H(S(0))\rceil$, $S(0)$ can be transmitted to the decoder as is. Thus, in this example, we can take

$$B(G|G') = B_1B_2 1101,$$

of length 42.

6. Universal Coding Theorem. We embark upon the main section of the paper. A formal definition of the concept of structured grammar-based code is given. Redundancy bounds for a structured grammar-based code with respect to families of information sources are obtained.

Information Sources. An *alphabet $\mathcal{A}$ information source* is defined to be any mapping $\mu : \mathcal{A}^+ \to [0, 1]$ such that

$$\begin{array}{rcl} 1 & = & \sum_{a\in\mathcal{A}} \mu(a) \\ \mu(\boldsymbol{x}) & = & \sum_{a\in\mathcal{A}} \mu(\boldsymbol{x}a), \;\; \boldsymbol{x} \in \mathcal{A}^+ \end{array}$$

That is, μ is a probability distribution induced by a random process with alphabet $\mathcal{A}$.

Finite-State Sources. Let k be a positive integer. An alphabet $\mathcal{A}$ information source μ is called a *k-th order finite-state source* if there is a set $\mathcal{S}$ of cardinality k, a symbol $s_0 \in \mathcal{S}$, and nonnegative real numbers $\{p(s, x|s') : s, s' \in \mathcal{S},\ x \in \mathcal{A}\}$ such that both of the following hold:

$$\sum_{s,x} p(s, x|s') = 1, \;\; s' \in \mathcal{S} \tag{6.1}$$

$$\mu(x_1x_2\cdots x_n) = \sum_{s_1,s_2,\cdots,s_n\in\mathcal{S}} \prod_{i=1}^{n} p(s_i, x_i|s_{i-1}), \;\; x_1x_2\cdots x_n \in \mathcal{A}^+. \tag{6.2}$$

We let Λ_k denote the family of all alphabet $\mathcal{A}$ k-th order finite-state sources. We call members of the set $\cup_k\Lambda_k$ *finite-state sources*. Note that finite-state sources defined here are much broader than tree sources considered in [10], [11], [9].

Lossless Source Codes. We define an *alphabet $\mathcal{A}$ lossless source code* to be a sequence of pairs $\mathcal{C} = \{(\epsilon_n, \delta_n) : n = 1, 2, \cdots\}$ in which

i) For each $n = 1, 2, \cdots$, ϵ_n is a mapping (called the n-th encoder mapping of the code $\mathcal{C}$) which maps each string $\boldsymbol{x}$ in $\mathcal{A}^n$ into a codeword $\epsilon_n(\boldsymbol{x}) \in \{0, 1\}^+$, and δ_n is the mapping (called the n-th decoder mapping of $\mathcal{C}$) which maps $\epsilon_n(\boldsymbol{x})$ back into $\boldsymbol{x}$; and

ii) for each $n = 1, 2, \cdots$, and each distinct pair of strings $\boldsymbol{x}_1, \boldsymbol{x}_2$ in $\mathcal{A}^n$, the codeword $\epsilon_n(\boldsymbol{x}_1)$ is not a prefix of the codeword $\epsilon_n(\boldsymbol{x}_2)$.

Structured Grammar-Based Codes. We define a *structure transform* to be a mapping $x \to G'_x$ from $\mathcal{A}^+$ into the set $\mathcal{G}(str)$ of structure grammars such that the structure grammar G'_x has $|x|$ leaf vertices in its derivation tree for every $\mathcal{A}$-string x. Fix an arbitrary structure transform $x \to G'_x$. For each positive integer n, let

$$\mathcal{G}^n \stackrel{\Delta}{=} \{G'_x : x \in \mathcal{A}^n\}.$$

Let $\mathcal{C} = \{(\epsilon_n, \delta_n) : n = 1, 2, \cdots\}$ be an alphabet $\mathcal{A}$ lossless source code. We call $\mathcal{C}$ a *structured grammar-based code induced by the structure transform* $x \to G'_x$ if ϵ_n and δ_n losslessly encode and decode respectively each $\mathcal{A}$-string x of length n according to the following four stages:

- *Analysis Stage:* Determine the structure grammar G'_x corresponding to x and then form the unique representational grammar G_x that represents x and has G'_x as its structure grammar.
- *Encoding Stage:* Encode the structure grammar G'_x, if needed, into a binary string $B(G'_x)$, and then conditionally encode G_x given G'_x into the binary string $B(G_x|G'_x)$, resulting in the total codeword $\epsilon_n(x) = B(G'_x)B(G_x|G'_x)$.
- *Decoding Stage:* Reconstruct G'_x using $B(G'_x)$ and G_x using $B(G_x|G'_x)$ and G'_x.
- *Synthesis Stage:* Recover x via propagation of the derivation tree of G_x.

The manner in which G'_x is encoded will depend in general on the underlying structure transform. As shown in Theorems 3 and 4, under certain mild conditions on the structure grammars in $\mathcal{G}^n$, the encoding of G'_x can be done quite naturally.

Redundancy Results. The type of redundancy we employ in this paper is *maximal redundancy/sample*. Let Λ be a family of alphabet $\mathcal{A}$ information sources. Let $\mathcal{C} = \{(\epsilon_n, \delta_n) : n = 1, 2, \ldots\}$ be an alphabet $\mathcal{A}$ lossless source code. The n-th order maximal redundancy/sample of $\mathcal{C}$ with respect to the family of sources Λ is the number

$$\mathrm{R}ed_n(\mathcal{C}, \Lambda) \stackrel{\Delta}{=} n^{-1} \max_{x \in \mathcal{A}^n} \left[|\epsilon_n(x)| - H(x|\Lambda) \right],$$

where $H(x|\Lambda)$ is defined by

$$H(x|\Lambda) \stackrel{\Delta}{=} \inf\{-\log \mu(x) : \mu \in \Lambda\}.$$

It should be pointed out that when the source class Λ is broad enough, the redundancy notion defined here is very strong. For instance, it can be shown that if $\mathcal{C}$ is CTW or PPM, then $\mathrm{R}ed_n(\mathcal{C}, \Lambda_k)$ is bounded below by a positive constant for large n.

Let us now impose some mild conditions on the underlying structure transform and analyze the redundancy of the resulting structured grammar-based code. Suppose that

$$\beta \overset{\Delta}{=} \sup_{x \in \mathcal{A}^+} \beta(G'_x) < \infty \tag{6.3}$$

That is, we require that the structure grammars $\{G'_x\}$ have uniformly bounded spreading factors. We now consider two cases. In Case 1, we assume that

$$|\mathcal{G}^n| = 1, \quad n = 1, 2, \cdots \tag{6.4}$$

In this case, the encoding of G'_x is free because all strings $x \in \mathcal{A}^n$ share a common structure grammar and hence there is no need to encode the common structure grammar. Theorem 3, stated below and proved in Appendix B, upper bounds the redundancy of this type of structured grammar-based code.

Theorem 3. *Let $\mathcal{C}$ be a structured grammar-based code induced by a structure transform satisfying (6.3) and (6.4). Let*

$$D \overset{\Delta}{=} 72\beta^2(|\mathcal{A}|+2)^2 \log(|\mathcal{A}|+2).$$

Then, for every positive integer k,

$$\mathrm{R}ed_n(\mathcal{C}, \Lambda_k) \le \frac{|\mathcal{A}|}{n} + \frac{(4+\log k)D}{\log n}, \quad n = 2, 3, \cdots \tag{6.5}$$

Remark. Theorem 3 tells us that any structured grammar-based code satisfying the regularity conditions (6.3) and (6.4) has maximal redundancy/sample $O(1/\log n)$. Compared to the conditions considered in [5] and [4], these are mild regularity conditions because they allow the use of grammars having highly unbalanced derivation trees. Previously considered $O(1/\log n)$ redundancy/sample grammar-based codes (such as the MPM code [5] and the codes in [4]) were more restrictive in that they employed grammars with approximately balanced derivation trees.

In Case 2, we allow strings $x \in \mathcal{A}^n$ to have different structure grammars, but require that structure grammars in $\mathcal{G}^n$, $n = 1, 2, \cdots$, satisfy the following condition:

Condition A: For any structure grammar $G'_x \in \mathcal{G}^n$, $n = 1, 2, \cdots$, different variables U_j of G'_x represent distinct $\{0\}$-strings.

(If $G'_x = G_{str}(\sigma')$, then the $\{0\}$-string represented by $U_j \in V(G'_x)$ is obtained by striking out all brackets from the subexpression σ'_j of σ' corresponding to U_j.) It is easy to see that the structure grammar G' in Example 7 satisfies Condition A with U_3, U_2, U_1, and U_0 representing $\{0\}$-strings 00, 0000, 00000000, and 000000000000, respectively. Under Condition A and (6.3), the encoding of each G'_x contributes only a negligible overhead, as shown in Theorem 4.

Theorem 4. *Let $\mathcal{C}$ be a structured grammar-based code induced by a structure transform satisfying (6.3) and Condition A. Assume that each structure grammar $G'_x \in \mathcal{G}^n$ is encoded into a codeword of length $\lceil \log |\mathcal{G}^n| \rceil$. Then, for every positive integer k,*

$$\mathrm{R}ed_n(\mathcal{C}, \Lambda_k) \le \frac{|\mathcal{A}|}{n} + \frac{(4+\log k)D}{\log n} + 12\beta^{3/2}\left(\frac{\log n}{\sqrt{n}}\right), \quad n = 2, 3, \cdots \tag{6.6}$$

In Theorem 4, the third term on the right hand side of (6.6) represents the overhead per sample contributed by the encoding of the structure grammar G'_x. Theorem 4 is proved in Appendix C.

Appendix A. This appendix is devoted to the proof of Theorem 1. The following lemma is our first step towards proving Theorem 1.

LEMMA 2. *Let n be an integer at least 2 and let $\mathcal{D}$ be a finite set with at least two elements. Let $J(n,\mathcal{D})$ be the largest number of distinct strings in $\mathcal{D}^+$ which are of total length at most n. Then,*

$$J(n,\mathcal{D}) \leq 4|\mathcal{D}|^2 \log|\mathcal{D}| \left(\frac{n}{\log n}\right).$$

Proof. Let $d = |\mathcal{D}|$. Let us assume until the end of the proof that $n > d^8$. If we list all strings in $\mathcal{D}^+$ in order of length, then the first $J(n,\mathcal{D})$ strings in this list will have total length $\leq n$ and the first $J(n,\mathcal{D})+1$ strings in this list will have total length $> n$. It follows that there is an integer $j \geq 2$ such that

$$d + d^2 + \cdots + d^j \leq J(n,\mathcal{D}) < d + d^2 + \cdots + d^{j+1} \tag{A1}$$

and

$$d + 2d^2 + \cdots + jd^j \leq n < d + 2d^2 + \cdots + (j+1)d^{j+1}. \tag{A2}$$

Summing the right side of (A1), we obtain

$$J(n,\mathcal{D}) < d\left(\frac{d^{j+1}-1}{d-1}\right) \leq d^2\left(\frac{d^j}{d-1}\right). \tag{A3}$$

Summing the left and right sides of (A2), we obtain

$$d\left[\frac{jd^j}{d-1} + \frac{1-d^j}{(d-1)^2}\right] \leq n < d\left[\frac{(j+1)d^{j+1}}{d-1} + \frac{1-d^{j+1}}{(d-1)^2}\right]$$

and then

$$\frac{(j-1)d^j}{d-1} \leq n < (d^2)^{j+1} \tag{A4}$$

follows because

$$d\left[\frac{jd^j}{d-1} + \frac{1-d^j}{(d-1)^2}\right] \geq \frac{(j-1)d^j}{d-1}$$

and

$$d\left[\frac{(j+1)d^{j+1}}{d-1} + \frac{1-d^{j+1}}{(d-1)^2}\right] \leq \frac{(j+1)d^{j+2}}{d-1} \leq (d^2)^{j+1}.$$

The right half of inequality (A4) gives us

$$j > (1/2)\left(\frac{\log n}{\log d}\right) - 1$$

which, applied to the left half of (A4), yields

$$\begin{aligned}\frac{d^j}{d-1} &\le \frac{n}{j-1}\\ &\le \frac{n}{(1/2)(\log n/\log d) - 2}\\ &= \frac{2n\log d}{\log(n/d^4)}\end{aligned}$$

Applying the preceding to (A3), we obtain

$$J(n,\mathcal{D}) < d^2\left(\frac{d^j}{d-1}\right) \le 2d^2\log d\left(\frac{n}{\log(n/d^4)}\right).$$

Since we are assuming that $n > d^8$, it follows that

$$\frac{n}{\log(n/d^4)} \le 2\left(\frac{n}{\log n}\right)$$

and the conclusion of Lemma 2 is true. For $2 \le n \le d^8$, the conclusion of Lemma 2 is also true, because

$$J(n,\mathcal{D}) \le n \le 8\log d\left(\frac{n}{\log n}\right) < 4d^2\log d\left(\frac{n}{\log n}\right).$$

Proof of Theorem 1. Fix $n \ge 2$ and an arbitrary representational grammar G which represents an $\mathcal{A}$-string of length n. Let σ be the $\mathcal{A}$-expression such that $G = G_{rep}(\sigma)$. Let $T(G|G^*)$ be a tree with $|G|$ edges and $|V(G)|$ internal vertices, obtained by pruning $T(\sigma)$ from the bottom up, such that

Prop(1): Each vertex v of $T(G|G^*)$ carries a label $\sigma(v)$ which is either a subexpression of σ or an element of $\mathcal{A}$.

Prop(2): Each label $\sigma(v)$ on an internal vertex v of $T(G|G^*)$ is a subexpression of σ, and each distinct subexpression of σ is a label $\sigma(v)$ for exactly one internal vertex v of $T(G|G^*)$.

Prop(3): The expression σ can be obtained via the concatenation of the labels $\sigma(v)$ on the leaf vertices v of $T(G|G^*)$ and some left and right brackets.

Let r be the number of leaf vertices of $T(G|G^*)$. Since $|G| \le 2r$, we may upper bound $|G|$ by doubling any upper bound we obtain on r. Since each internal vertex of $T(G|G^*)$ has at most $K \triangleq \lfloor\beta(G^*)\rfloor$ children, it follows that there must exist $s \ge r/K$ distinct internal vertices $v_1, v_2, \cdots, v_s$ of $T(G|G^*)$, such that each leaf vertex of $T(G|G^*)$ has one of these vertices v_i as its parent and each v_i has a leaf vertex of $T(G|G^*)$ as one of its children. Pick a leaf vertex u_i which is the child of v_i for each i.

Since $L(v_i|T(\sigma)) \leq KL(u_i|T(\sigma))$, and since the length of $\sigma(v_i)$ is at most three times $L(v_i|T(\sigma))$, we have

$$|\sigma(v_i)| \leq 3L(v_i|T(\sigma)) \leq 3KL(u_i|T(\sigma)) \leq 3K|\sigma(u_i)|.$$

By *Prop(3)*, the summation of the $|\sigma(u_i)|$ is less than $|\sigma|$; also, σ is of length at most $3n-2$. Therefore, the distinct strings $\sigma(v_i)$ have total length $< 9Kn$. Applying Lemma 2 with $\mathcal{D} = \mathcal{A} \cup \{[,]\}$ and $|\mathcal{D}| = |\mathcal{A}| + 2$, we obtain

$$\begin{aligned} s &\leq 4(|\mathcal{A}|+2)^2 \log(|\mathcal{A}|+2)(9Kn/\log(9Kn)) \\ &\leq 36K(|\mathcal{A}|+2)^2 \log(|\mathcal{A}|+2)(n/\log n) \end{aligned}$$

The bound on r is then K times this, and the bound on $|G|$ is obtained by doubling the bound on r. This gives us the conclusion of Theorem 1.

Appendix B. This appendix is devoted to the proof of Theorem 3. We present the key lemma needed to prove Theorem 3.

LEMMA 3. *Let k be any positive integer. Then*

$$H(G|G^*) - H(x|\Lambda_k) \leq |G| \log k \tag{B1}$$

for any $\mathcal{A}$-string x and any representational grammar G which represents x.

Proof of Lemma 3. Let x be a fixed $\mathcal{A}$-string and let G be a fixed representational grammar which represents x. Let k be a fixed positive integer. Our task is to show that (B1) is true. Given a set $\mathcal{S}$ with k elements, a symbol $s_0 \in \mathcal{S}$, and a set of nonnegative real numbers $p = \{p(s,u|s') : s, s' \in \mathcal{S}, u \in \mathcal{A}\}$ satisfying (6.1), let μ_{p,s_0} denote the source in Λ_k defined by equation (6.2). As we let s_0 and p vary through all possibilities, μ_{p,s_0} varies over all sources in Λ_k. Fix p and define λ_p to be the function

$$\lambda_p(u) = \max_{s_0 \in \mathcal{S}} \mu_{p,s_0}(u), \quad u \in \mathcal{A}^+.$$

The function λ_p has the following two properties which are exploited in this proof:

(p.1): If $u_1, u_2, \cdots, u_j$ are $\mathcal{A}$-strings which yield the $\mathcal{A}$-string u when concatenated together, then

$$\lambda_p(u) \leq \lambda_p(u_1)\lambda_p(u_2)\cdots\lambda_p(u_j).$$

(p.2): For every positive integer m,

$$1 \leq \sum_{u \in \mathcal{A}^m} \lambda_p(u) \leq k.$$

Let σ be the $\mathcal{A}$-expression such that $G_{rep}(\sigma) = G$, and let σ' be the $\{0\}$-expression such that $G_{str}(\sigma') = G^*$. Let $T(G|G^*)$ be the tree used in our earlier proofs. For our purposes here, $T(G|G^*)$ has the following properties:

(p.3): The tree $T(G|G^*)$ has $|G| - |V(G)| + 1$ leaf vertices.

(p.4): Each leaf vertex v of $T(G|G^*)$ has a label $\sigma(v)$ which is either a subexpression of σ or an element of $\mathcal{A}$, a label $\sigma'(v)$ which is either 0 or a subexpression of σ', and a label $x(v)$ which is a substring of x. The label $\sigma'(v)$ is obtained from the label $\sigma(v)$ by replacing each $\mathcal{A}$-entry of $\sigma(v)$ with 0, and the label $x(v)$ is obtained from the label $\sigma(v)$ by removing all brackets from $\sigma(v)$.

(p.5): For each α' which is either 0 or a proper subexpression of σ', if we let $S(\alpha')$ be the sequence formed by the labels $\sigma(v)$ for those leaf vertices v of $T(G|G^*)$ for which $\sigma'(v) = \alpha'$, then

$$H(G|G^*) = \sum_{\alpha'} H(S(\alpha')). \tag{B2}$$

(p.6): The labels $x(v)$ on the leaf vertices v of $T(G|G^*)$ yield x when concatenated together (according to the left-to-right ordering of the leaf vertices).

For each α' which is either 0 or a proper subexpression of σ', let the positive integer $m(\alpha')$ be the number of entries of α' which are equal to 0, and let $\mathcal{V}(\alpha')$ be the set of leaf vertices v of $T(G|G^*)$ for which $\sigma'(v) = \alpha'$. Then, for each $v \in \mathcal{V}(\alpha')$, the string $x(v)$ has length $m(\alpha')$. For each positive integer m, let τ_m be the probability distribution on $\mathcal{A}^m$ such that

$$\tau_m(u) = \lambda_p(u)/\Sigma_m, \;\; u \in \mathcal{A}^m,$$

where, using property *(p.2)*,

$$\Sigma_m \triangleq \sum_{u \in \mathcal{A}^m} \lambda_p(u) \le k. \tag{B3}$$

It follows that

$$\begin{aligned} H(S(\alpha')) &\le \sum_{v \in \mathcal{V}(\alpha')} - \log \tau_{m(\alpha')}(x(v)) \\ &\le |S(\alpha')| \log k + \sum_{v \in \mathcal{V}(\alpha')} - \log \lambda_p(x(v)). \end{aligned}$$

Summing each side of the preceding inequality over α', we obtain

$$H(G|G^*) + \log \lambda_p(x) \le |G| \log k. \tag{B4}$$

Here, we used (B2) and the fact that

$$\sum_{\alpha'} \sum_{v \in \mathcal{V}(\alpha')} - \log \lambda_p(x(v)) \le - \log \lambda_p(x),$$

which follows from properties *(p.1)* and *(p.6)*. We also used the fact that

$$\sum_{\alpha'} |S(\alpha')| \le |G|,$$

which is true because $T(G|G^*)$ has no more than $|G|$ leaf vertices (property *(p.3)*). Inequality (B1) now results by taking the supremum of both sides of (B4) over p.

Proof of Theorem 3. Let $\mathcal{C} = \{(\epsilon_n, \delta_n)\}$ be a structured grammar-based code induced by a structure transform satisfying (6.3) and (6.4). Let n be a fixed positive integer. Since all $\mathcal{A}$-strings of length n share a common structure grammar, denote this common structure grammar by G^n. Fix an arbitrary string $x \in \mathcal{A}^n$ and let G_x be the representational grammar representing x whose structure grammar is G^n. Applying Theorem 2,

$$|\epsilon_n(x)| = |B(G_x|G^n)| \leq H(G_x|G^n) + 4|G_x| + |\mathcal{A}|.$$

Subtracting $H(x|\Lambda_k)$ from both sides and applying Lemma 3, we see that

$$\mathrm{R}ed_n(\mathcal{C}, \Lambda_k) \leq n^{-1}|\mathcal{A}| + n^{-1}(4 + \log k) \max_{x \in \mathcal{A}^n} |G_x|.$$

Applying Theorem 1 to the last term on the right in the preceding equation, we see that the conclusion of Theorem 3 is true.

Appendix C. In this appendix, we prove Theorem 4. In view of the proof of Theorem 3, it suffices to upper bound the overhead contributed by the encoding of each G'_x.

LEMMA 4. *Under Condition A and (6.3),*

$$\lceil \log |\mathcal{G}^n| \rceil \leq 12\beta^{3/2}\sqrt{n}\log n$$

for any integer $n \geq 2$.

Proof of Lemma 4: Let G' be an arbitrary structure grammar from $\mathcal{G}^n$. Let σ be the $\{0\}$-expression such that $G' = G_{str}(\sigma)$. We first use a technique similar to the proof of Theorem 1 to upper bound the size of G'. Let G be the representational grammar $G_{rep}(\sigma)$; then $G^* = G'$ and the grammars G, G' are identical except for the fact that variables of G are denoted A_j and variables of G' are denoted U_j. Let r be the number of leaf vertices of the tree $T(G|G^*)$ defined in the proof of Theorem 1. (Throughout the rest of this proof, notation will be the same as in the proof of Theorem 1, unless otherwise specified.) Then

$$|G'| \leq 2r \leq 2\beta(G')s \tag{C1}$$

For each vertex v of $T(G|G^*)$, let $x(v)$ be the $\{0\}$-string obtained by striking out all possible brackets from $\sigma(v)$. From the proof of Theorem 1, it follows that

$$|x(v_i)| \leq \beta(G')|x(u_i)|$$

and hence

$$\sum_{i=1}^{s} |x(v_i)| \leq \beta(G') \sum_{i=1}^{s} |x(u_i)| \leq \beta(G')n \tag{C2}$$

Note that under Condition A, all $x(v_i)$, $i = 1, 2, \cdots, s$, are distinct. Let N be the positive integer equal to the left side of (C2). Let $J(N)$ be the maximum number of distinct $\{0\}$-strings which are of total length at most N. Then

$$\frac{1}{2}J(N)[J(N)+1] \leq N < \frac{1}{2}[J(N)+1][J(N)+2] < [J(N)+1]^2.$$

From this, we have

$$J(N) \leq 2\sqrt{N}$$

which, together with (C2) and (C1), implies

$$s \leq 2\sqrt{\beta(G')n} \text{ and } |G'| \leq 4[\beta(G')]^{3/2}\sqrt{n} \tag{C3}$$

Let us now describe how to encode G'. The first part B_1 of the codeword of G' tells the decoder the length of the right member of each production rule of G'. This can be accomplished by the unary representation of each length. Thus, the length of B_1 is $|G'|$. The second part B_2 tells the decoder the actual symbols in the right member of each production rule. This can be accomplished by using $\lceil \log n \rceil$ bits to represent each symbol U_j or 0. The length of B_2 is $|G'|\lceil \log n \rceil$. The complete codeword is the concatenation of B_1 with B_2. The total codeword length is

$$\begin{aligned}|B_1B_2| &= |G'|(1 + \lceil \log n \rceil) \\ &\leq 4\lfloor [\beta(G')]^{3/2}\sqrt{n} \rfloor (1 + \lceil \log n \rceil) \\ &\leq 4\lfloor \beta^{3/2}\sqrt{n} \rfloor (1 + \lceil \log n \rceil)\end{aligned}$$

In the above, the first inequality is due to (C3), and the second inequality is attributed to (6.3). Since such a coding scheme is a prefix code for $\mathcal{G}^n$, Lemma 4 follows.

Proof of Theorem 4: It now follows immediately from Lemma 4 and the proof of Theorem 3.

REFERENCES

[1] J. G. Cleary and I. H. Witten, *Data compression using adaptive coding and partial string matching*, IEEE Trans. Commun., 32(1984), pp. 396–402.

[2] L. Davisson, *Universal Noiseless Coding*, IEEE Trans. Inform. Theory, 19(1973), pp. 783–795.

[3] J. Kieffer and E.-H. Yang, *Grammar-Based Codes: A New Class of Universal Lossless Source Codes*, IEEE Trans. Inform. Theory, 46(2000), pp. 737–754.

[4] J. Kieffer and E.-H. Yang, *Lossless Data Compression via Guided Approximate Bisections*, Proc. 2000 Conf. Inform. Sci. Systems (Princeton Univ.), Volume II, pp. TP6-1–TP6-6.

[5] J. Kieffer, E.-H. Yang, G. Nelson, and P. Cosman, *Universal Lossless Compression via Multilevel Pattern Matching*, IEEE Trans. Inform. Theory, 46(2000), pp. 1227–1245.

[6] E. Plotnik, M. Weinberger, and J. Ziv, *Upper Bounds on the Probability of Sequences Emitted by Finite-State Sources and on the Redundancy of the Lempel-Ziv Algorithm*, IEEE Trans. Inform. Theory, 38(1992), pp. 66–72.

[7] N. Sloane, *On-Line Encyclopedia of Integer Sequences*, `http://www.research.att.com/~njas /sequences/`.

[8] R. Stanley, *Enumerative Combinatorics*, Volume 2. Cambridge University Press, Cambridge, UK, 1999.

[9] M. J. Weinberger, J. Rissanen, and M. Feder, *A universal finite memory source*, IEEE Trans. Inform. Theory, 41:3(1995), pp. 643–652.

[10] F.M.J. Willems, Y.M. Shtarkov, and Tj.J. Tjalkens, *The context tree weighting method: Basic properties*, IEEE Trans. Inform. Theory, 41(1995), pp. 653–664.

[11] F. M. J. Willems, Y. M. Shtarkov, and Tj.J. Tjalkens, *Context weighting for general finite-context sources*, IEEE Trans. Inform. Theory, 42:5(1996), pp. 1514–1520.

[12] E.-H. Yang and J. C. Kieffer, *Efficient universal lossless data compression algorithms based on a greedy sequential grammar transform—Part one: Without context models*, IEEE Trans. Inform. Theory, 46(2000), pp. 755–777.

[13] J. Ziv and A. Lempel, *A Universal Algorithm for Data Compression*, IEEE Trans. Inform. Theory, 23(1977), pp. 337–343.

[14] J. Ziv and A. Lempel, *Compression of Individual Sequences via Variable-Rate Coding*, IEEE Trans. Inform. Theory, 24(1978), pp. 530–536.