

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/3080039>

Grammar Based Codes: A New Class of Universal Lossless Source Codes

Article in IEEE Transactions on Information Theory · June 2000

DOI: 10.1109/18.841160 · Source: IEEE Xplore

CITATIONS

432

READS

383

2 authors, including:



J.C. Kieffer

University of Minnesota

158 PUBLICATIONS 3,438 CITATIONS

[SEE PROFILE](#)

Grammar Based Codes: A New Class of Universal Lossless Source Codes

John C. Kieffer and En-hui Yang

This work was supported in part by National Science Foundation Grants NCR-9304984, NCR-9508282, NCR-9627965, and by the Natural Sciences and Engineering Research Council of Canada under Grant RGPIN203035-98.

John C. Kieffer is with the 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

En-hui Yang is with the Department of Electrical & Computer Engineering, University of Waterloo, Waterloo, Ontario, CA N2L 3G1. E-mail: ehyang@bbcr.uwaterloo.ca

Abstract

We investigate a type of lossless source code called a grammar based code, which, in response to any input data string \mathbf{x} over a fixed finite alphabet, selects a context-free grammar $G_{\mathbf{x}}$ representing \mathbf{x} in the sense that \mathbf{x} is the unique string belonging to the language generated by $G_{\mathbf{x}}$. Lossless compression of \mathbf{x} takes place indirectly via compression of the production rules of the grammar $G_{\mathbf{x}}$. It is shown that, subject to some mild restrictions, a grammar based code is a universal code with respect to the family of finite state information sources over the finite alphabet. Redundancy bounds for grammar based codes are established. Reduction rules for designing grammar based codes are presented.

Index Terms: lossless coding, universal coding, redundancy, context-free grammars, entropy, Kolmogorov complexity, Chomsky hierarchy

1 Introduction

Grammars (especially context-free grammars) have many applications in engineering and computer science. Some of these applications are speech recognition ([8], Chapter 13), image understanding ([22], p. 289), compiler design [1], and language modeling ([10], Theorems 4.5, 4.6). In this paper, we shall be interested in using context-free grammars for lossless data compression. There has been some previous work of this nature, including the papers [3] [2] [11] [14] [24] [18]. Two approaches have been used. In one of these approaches (as illustrated in [2] [11] [14]), one fixes a context-free grammar G , known to both encoder and decoder, such that the language generated by G contains all of the data strings that are to be compressed. To compress a particular data string, one then compresses the derivation tree ([2], p. 844) showing how the given string is derived from the start symbol of the grammar G . In the second of the two approaches (exemplified by the papers [3] [24] [18]), a different context-free grammar G_x is assigned to each data string x , so that the language generated by G_x is $\{x\}$. If the data string x is to be compressed, the encoder transmits codebits to the decoder that allow reconstruction of the grammar G_x , from which the decoder infers x . This second approach is the approach that we employ in this paper. We shall put forth a class of lossless source codes that employ this approach that we call *grammar based codes*. Unlike previous workers using the second approach, we place our results in an information-theoretic perspective, showing how to properly design a grammar based code so that it will be a universal code with respect to the family of finite state information sources on a fixed finite alphabet.

In this introduction, we wish to give the reader an informal notion of the idea of a grammar based code. For this purpose, we do not need a precise definition of the concept of context-free grammar (this will be done in the next section). All we need to know about a context-free grammar G here is that it furnishes us with some production rules via which we can construct certain sequences over a finite alphabet which form what is called the *language generated by G* , denoted by $L(G)$.

A grammar based code consists of encoder and decoder:

- Figure 1 depicts the encoder structure. Letting x denote the data string that is to be compressed, consisting of finitely many terms chosen from some fixed finite

alphabet, the *grammar transform* in Figure 1 constructs a context-free grammar G_x satisfying the property that $L(G_x) = \{x\}$, which tells us that x may be inferred from G_x because x is the unique string belonging to the language $L(G_x)$. The *grammar encoder* in Figure 1 assigns to the grammar G_x a binary codeword which is denoted $B(G_x)$.

- When the decoder is presented with the codeword $B(G_x)$, the data string x is recovered by first reconstructing the grammar G_x , and then inferring x from the production rules of G_x .

From the preceding, the reader can see that our philosophy is not to directly compress the data string x ; instead, we try to “explain” x by finding a grammar G_x that is simple and generates x in the sense that $L(G_x) = \{x\}$. Since x can be recovered from G_x , we can compress G_x instead of x . As the grammar G_x that we shall use to represent x will be simple, we will get good compression by compressing G_x .

The main results of this paper (Theorems 7 and 8) tell us that, under some weak restrictions, a grammar based code is a universal lossless source code for any finite state information source. We shall be able to obtain specific redundancy bounds for grammar based codes with respect to finite state information sources. Also, we shall see how to design efficient grammar based codes by means of reduction rules.

As a result of this paper, the code designer is afforded with more flexibility in universal lossless source code design. For example, for some data strings, a properly designed grammar based code yields better compression performance than that afforded by the Lempel-Ziv universal data compression algorithm [27].

Notation and Terminology. We explain the following notations and terminologies used throughout the paper:

- $|\mathcal{S}|$ denotes the cardinality of a finite set \mathcal{S} .
- $|x|$ denotes the length of a string x of finite length.
- $[x]$ denotes the smallest positive integer greater than or equal to the real number x .

- \mathcal{S}^* denotes the set of all strings of finite length whose entries come from the finite set \mathcal{S} , including the empty string. We represent each nonempty string in \mathcal{S}^* multiplicatively and uniquely as $x_1x_2 \cdots x_n$, where n is the length of the string and $x_1, x_2, \dots, x_n \in \mathcal{S}$; we write the empty string in \mathcal{S}^* as $1_{\mathcal{S}}$.
- If x and y are elements of \mathcal{S}^* , we define the product xy to be the element of \mathcal{S}^* such that
 - (i) If $x = 1_{\mathcal{S}}$, then $xy = y$; if $y = 1_{\mathcal{S}}$, then $xy = x$.
 - (ii) If $x \neq 1_{\mathcal{S}}$ and $y \neq 1_{\mathcal{S}}$, then

$$xy = x_1x_2 \cdots x_n y_1y_2 \cdots y_m,$$

where $x_1x_2 \cdots x_n$ and $y_1y_2 \cdots y_m$ are the unique multiplicative representations of x and y , respectively.

The multiplication operation $(x, y) \rightarrow xy$ on \mathcal{S}^* is associative. Therefore, given $s_1, s_2, \dots, s_j \in \mathcal{S}^*$, the product $s_1s_2 \cdots s_j$ is an unambiguously defined element of \mathcal{S}^* . (The set \mathcal{S}^* , with the multiplication we have defined, is called a *multiplicative monoid*.)

- A *parsing* of a string $u \in \mathcal{S}^*$ is any sequence (u_1, u_2, \dots, u_m) in which u_1, u_2, \dots, u_m are strings in \mathcal{S}^* such that $u_1u_2 \cdots u_m = u$.
- \mathcal{S}^+ denotes the set \mathcal{S}^* with the empty string $1_{\mathcal{S}}$ removed.
- For each positive integer n , \mathcal{S}^n denotes the set of all strings in \mathcal{S}^* of length n .
- All logarithms are to base two.

2 Admissible Grammars

In this section, we introduce a subclass of the class of all context-free grammars called the class of *admissible grammars*. For each admissible grammar G , it is guaranteed that

$$L(G) = \{\mathbf{x}\} \tag{2.1}$$

will hold for some string \mathbf{x} . A simple test is given, which will allow us to determine whether or not a grammar is admissible (Theorem 2). We will also present an algorithm for the calculation of the string \mathbf{x} in (2.1), from a given admissible grammar G (Theorem 3).

A *production rule* is an expression of the form

$$A \rightarrow \alpha \tag{2.2}$$

where $A \in \mathcal{S}$ and $\alpha \in \mathcal{S}^*$ for some finite set \mathcal{S} . The left member (resp., right member) of the production rule (2.2) is defined to be A (resp., α). Following [10], a *context-free grammar* is a quadruple $G = (V, T, P, S)$ in which

- V is a finite nonempty set whose elements shall be called *variables*.
- T is a finite nonempty set, disjoint from V , whose elements shall be called *terminal symbols*.
- P is a finite set of production rules whose left members come from V and whose right members come from $(V \cup T)^*$. We assume that for each $A \in V$, there is at least one production rule in P whose left member is A .
- S is a special variable in V called the *start symbol*.

We adopt the following notational conventions. We shall denote the set of variables, the set of terminal symbols, and the set of production rules for a context-free grammar G by $V(G), T(G), P(G)$, respectively. When a variable in $V(G)$ is denoted “ S ”, that will always mean that the variable is the start symbol. Upper case symbols A, B, C, D, \dots (with or without subscripts) are used to denote variables, and lower case symbols a, b, c, d, \dots (with or without subscripts) are used to denote terminal symbols. Given a context-free grammar G , and a variable $A \in V(G)$, there may exist a *unique* production rule in $P(G)$ whose left member is A — we shall refer to this production rule as “the A production rule.”

Let G be a context-free grammar. If α and β are strings in $(V(G) \cup T(G))^*$,

- We write $\alpha \xrightarrow{G} \beta$ if there are strings α_1, α_2 and a production rule $A \rightarrow \gamma$ of G such that (α_1, A, α_2) is a parsing of α and $(\alpha_1, \gamma, \alpha_2)$ is a parsing of β . (In other words,

we obtain β from α by replacing some variable in α with the right member of a production rule whose left member is that variable.)

- We write $\alpha \xrightarrow{G} \beta$ if there exists a sequence of strings $\alpha_1, \alpha_2, \dots, \alpha_k$ such that

$$\alpha = \alpha_1 \xrightarrow{G} \alpha_2, \alpha_2 \xrightarrow{G} \alpha_3, \dots, \alpha_{k-1} \xrightarrow{G} \alpha_k = \beta$$

The language $L(G)$ generated by G is defined by:

$$L(G) \triangleq \{u \in T^* : S \xrightarrow{G} u\}$$

We are now ready to define the notion of an admissible grammar. We define a context-free grammar G to be *admissible* if all of the following properties hold:

- G is *deterministic*. This means that for each variable $A \in V(G)$, there is exactly one production rule in $P(G)$ whose left member is A .
- The empty string is not the right member of any production rule in $P(G)$.
- $L(G)$ is nonempty.
- G has no useless symbols. By this, we mean that for each symbol $Y \in V(G) \cup T(G)$, $Y \neq S$, there exist finitely many strings $\alpha_1, \alpha_2, \dots, \alpha_n$ such that at least one of the strings contains Y and

$$S = \alpha_1 \xrightarrow{G} \alpha_2, \alpha_2 \xrightarrow{G} \alpha_3, \dots, \alpha_{n-1} \xrightarrow{G} \alpha_n \in L(G)$$

It can be seen that for any deterministic grammar G , the language $L(G)$ is either empty or consists of exactly one string. Therefore, if G is an admissible grammar, there exists a unique string $\mathbf{x} \in T(G)^+$ such that $L(G) = \{\mathbf{x}\}$. This string \mathbf{x} shall be called the *data string represented by G* . We shall also say that G *represents* \mathbf{x} .

When we want to specify an admissible grammar G , we need only list the production rules of G , because $V(G), T(G)$, and the start symbol S can be uniquely inferred from the production rules. The set of variables $V(G)$ will consist of the left members of the

production rules, the set of terminal symbols $T(G)$ will consist of the symbols which are not variables and which appear in the right members of the production rules, and the start symbol S is the unique variable which does not appear in the right members of the production rules.

Example 1. Suppose that a grammar G (which will be shown to be admissible in Example 2) has production rules:

$$\begin{aligned} A_0 &\rightarrow aA_1A_2A_3 \\ A_1 &\rightarrow ab \\ A_2 &\rightarrow A_1b \\ A_3 &\rightarrow A_2b \end{aligned}$$

Looking at the left members of the production rules, we see that $V(G) = \{A_0, A_1, A_2, A_3\}$. Of these four variables, A_0 is the only one not appearing in the right members, and so the start symbol of the grammar G is $S = A_0$. Striking out A_1, A_2, A_3 from the right members, the remaining symbols give us $T(G) = \{a, b\}$. It will be determined in Example 3 that the data string represented by G is $aababbabbb$. This means that $L(G) = \{aababbabbb\}$.

Let \mathcal{S} be a finite set. An *endomorphism* on \mathcal{S}^* is a mapping f from \mathcal{S}^* into itself such that the following two conditions hold:

- $f(1_{\mathcal{S}}) = 1_{\mathcal{S}}$
- $f(u_1u_2) = f(u_1)f(u_2), \quad u_1, u_2 \in \mathcal{S}^*$

Notice that an endomorphism f on \mathcal{S}^* is uniquely determined once $f(u) \in \mathcal{S}^*$ has been specified for every $u \in \mathcal{S}$.

Given an endomorphism f on \mathcal{S}^* , we can define a family of endomorphisms $\{f^k : k = 0, 1, 2, \dots\}$ on \mathcal{S}^* by:

$$\begin{aligned} f^0 &= \text{identity map} \\ f^1 &= f \\ f^k(u) &= f(f^{k-1}(u)), \quad k \geq 2, \quad u \in \mathcal{S}^* \end{aligned}$$

Following [17] [21], an L-system¹ is a triple (\mathcal{S}, f, u) in which

- \mathcal{S} is a finite set.
- f is an endomorphism on \mathcal{S}^* .
- $u \in \mathcal{S}^*$.

The *fixed point* u^* (if it exists) of an L-system (\mathcal{S}, f, u) is the unique string u^* such that

- $u^* \in \{f^k(u) : k = 0, 1, 2, \dots\}$
- $f(u^*) = u^*$

Suppose G is a deterministic context-free grammar in which the empty string is not the right member of any production rule. We define f_G to be the endomorphism on $(V(G) \cup T(G))^*$ such that

- $f_G(a) = a$, $a \in T(G)$
- If $A \rightarrow \alpha$ is a production rule of G , then $f_G(A) = \alpha$.

We have recounted standard background material on context-free grammars and L-systems. We now present new material which will allow us to reconstruct a data string from an admissible grammar which represents it.

The following theorem indicates why L-systems are important to us. (The proof, which is almost self-evident, is omitted.)

Theorem 1 *Let G be an admissible grammar. Then the data string \mathbf{x} represented by G can be characterized as the fixed point of the L-system $(V(G) \cup T(G), f_G, S)$.*

Derivation Graphs. Let G be a deterministic context-free grammar for which the empty string is not the right member of any production rule. We can associate with G a finite directed graph called the *derivation graph* of G . There are $|V(G) \cup T(G)|$ vertices in the derivation graph of G . Each vertex of the derivation graph is labelled with

¹sometimes referred to as a D0L system

a symbol from $V(G) \cup T(G)$, with no two vertices carrying the same label. There are $|T(G)|$ terminal vertices in the derivation graph, whose labels come from $T(G)$, whereas the labels on the nonterminal vertices come from $V(G)$. If a nonterminal vertex is labelled by a variable $A \in V(G)$, and if $A \rightarrow Y_1 Y_2 \cdots Y_k$ is the A production rule, then there are k edges emanating from the vertex; the labels on the vertices at which these edges terminate are Y_1, Y_2, \dots, Y_k , respectively. We shall refer to a vertex of the derivation graph of G labelled by $Y \in V(G) \cup T(G)$ as the “ Y vertex.”

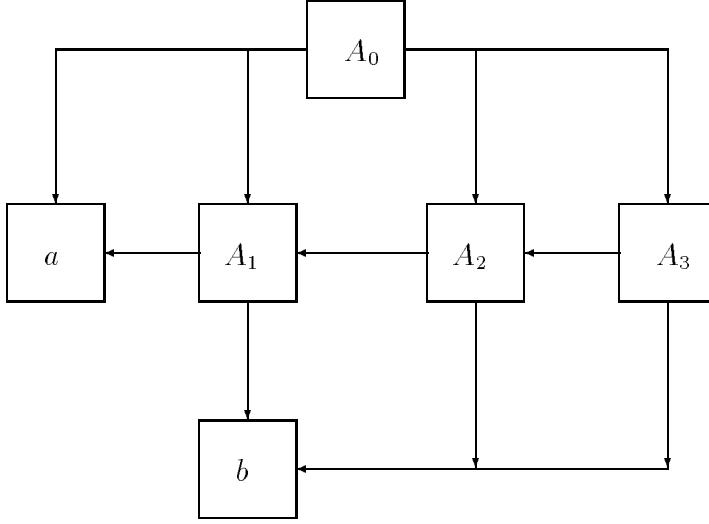
We can use the derivation graph of a grammar G to deduce certain properties of the grammar. Before we do that, we discuss some characteristics of directed graphs. A *path* in a directed graph is a finite or infinite sequence $\{e_i\}$ of edges of the graph, such that for every pair of consecutive edges (e_i, e_{i+1}) from the sequence, the edge e_i terminates at the vertex where edge e_{i+1} begins. A directed graph is said to be *rooted* at one of its vertices v if for each vertex $v' \neq v$ of the graph, there is a path whose first edge begins at v and whose last edge ends at v' . A path in a directed graph which begins and ends at the same vertex is called a *cycle*. A directed graph with no cycles is called an *acyclic* graph.

The following theorem, proved in Appendix A, gives us some simple conditions to check to see whether a grammar is admissible.

Theorem 2 *Let G be a deterministic context-free grammar such that the empty string is not the right member of any production rule of G . Then the following three statements are equivalent:*

- (i) *G is admissible.*
- (ii) *The derivation graph of G is acyclic and is rooted at the S vertex.*
- (iii) *$f_G^{|V(G)|}(S) \in T(G)^+$ and each symbol in $V(G) \cup T(G)$ is an entry of at least one of the strings $f_G^i(S)$, $i = 0, 1, \dots, |V(G)|$.*

Example 2. The grammar G in Example 1 has the following derivation graph:



Notice that the graph is acyclic, and is rooted at the vertex labelled with the start symbol $S = A_0$. Theorem 2 allows us to conclude that the grammar G is admissible.

The following theorem, which follows from Theorem 2, gives us an algorithm for computing the data string represented by an admissible grammar.

Theorem 3 *Let G be an admissible grammar. Then the data string \mathbf{x} represented by G is computed by doing the calculation*

$$\mathbf{x} = f_G^{|V(G)|}(S) \quad (2.3)$$

Example 3. Let G again be the grammar in Example 1. From Example 2, we know that G is admissible. Since $|V(G)| = 4$, Theorem 3 tells us that the data string represented by G is $f_G^4(A_0)$, which we compute as follows:

$$\begin{aligned}
 f_G(A_0) &= aA_1A_2A_3 \\
 f_G^2(A_0) &= aabA_1bA_2b \\
 f_G^3(A_0) &= aababbA_1bb \\
 f_G^4(A_0) &= aababbabbb
 \end{aligned}$$

Notice that condition (iii) of Theorem 2 holds. (Each of the symbols A_1, A_2, A_3, a, b appears at least once in the strings computed above; also, $f_G^4(S)$ is a string in $T(G)^+$.)

This gives us another verification that G is admissible.

The following theorem generalizes Theorem 1 and follows easily from Theorem 2 in combination with Lemma 4 of Appendix A. It shall be useful to us in subsequent sections of the paper.

Theorem 4 *Let G be an admissible context-free grammar. Let u be any string in $(V(G) \cup T(G))^+$. Then, the L-system $(V(G) \cup T(G), f_G, u)$ has a fixed point u^* , and $u^* \in T(G)^+$. The fixed point u^* is computable via the formula*

$$u^* = f_G^{|V(G)|}(u)$$

A useful endomorphism. Let G be an admissible grammar. In view of Theorem 4, we may define a mapping f_G^∞ from $(V(G) \cup T(G))^*$ into itself such that, if u is any string in $(V(G) \cup T(G))^*$, then $f_G^\infty(u)$ is the fixed point of the L-system $(V(G) \cup T(G), f_G, u)$. The following result gives us a number of properties of the mapping f_G^∞ that shall be needed later on.

Theorem 5 *Let G be an admissible grammar. Then:*

- (i) *f_G^∞ is an endomorphism on $(V(G) \cup T(G))^*$.*
- (ii) *$f_G^\infty(u) \in T(G)^+$ for each $u \in (V(G) \cup T(G))^+$.*
- (iii) *For each $u \in (V(G) \cup T(G))^+$,*

$$f_G^\infty(u) = f_G^{|V(G)|}(u)$$

- (iv) *If $A \rightarrow \alpha$ is a production rule of G , then*

$$f_G^\infty(A) = f_G^\infty(\alpha)$$

Proof of Theorem 5. Properties (i)-(ii) are trivially seen to be true. Property (iii) is a consequence of Theorem 4. Property (iv) follows from THE fact that if $A \rightarrow \alpha$ is a production rule, then the sequence $\{f_G^i(\alpha) : i \geq 1\}$ is obtained by throwing away the

first term of the sequence $\{f_G^i(A) : i \geq 0\}$, whence the fixed points arising from these sequences are the same.

3 Grammar Transforms

The grammar transform in Figure 1 is the most important component of a grammar based code. Formally, a grammar transform shall be defined as a mapping which assigns to each data string a grammar which represents the string. This section is devoted to the study of grammar transforms. We shall focus on two general classes of grammar transforms, the *asymptotically compact grammar transforms* (Section 3.1) and the *irreducible grammar transforms* (Section 3.2).

For the rest of the paper, we let \mathcal{A} denote an arbitrary finite set of size at least two; the set \mathcal{A} shall serve as the alphabet from which our data strings are to be drawn. We shall call a string in \mathcal{A}^+ an \mathcal{A} -string. We fix a countably infinite set of symbols

$$\{A_0, A_1, A_2, A_3, \dots\} \quad (3.4)$$

from which we will select the variables to be used in forming the grammars to be employed in a grammar transform. We assume that each of the symbols in (3.4) is not a member of the alphabet \mathcal{A} .

We define $\mathcal{G}(\mathcal{A})$ to be the set of all grammars G satisfying the following properties:

- (i) G is admissible.
- (ii) $T(G) \subset \mathcal{A}$.
- (iii) The start symbol of G is A_0 .
- (iv) $V(G) = \{A_0, A_1, A_2, \dots, A_{|V(G)|-1}\}$.
- (v) If we list the variables in $V(G)$ in order of their first left-to-right appearance in the string

$$f_G^0(A_0)f_G^1(A_0)f_G^2(A_0)\cdots f_G^{|V(G)|-1}(A_0) \quad (3.5)$$

then we obtain the list

$$A_0, A_1, A_2, \dots, A_{|V(G)|-1}$$

Discussion. For the purposes of this paper, the function of a grammar is to represent a data string. From this point of view, it makes no difference what symbols are used as the “names” for the variables in $V(G)$. Indeed, in reconstructing a data string from a grammar G which represents it, the variables in $V(G)$ are “dummy” variables which are substituted for in the reconstruction process. By means of property (v), we have required that the variables in $V(G)$ be named in a fixed way, according to a “canonical ordering” of the variables in $V(G)$. Our canonical ordering is the unique ordering induced by the depth-first search through the vertices of the derivation graph of G in which the daughter vertices of a vertex are visited in order of the left-to-right appearance of terms in the right member of a production rule of G . It is precisely this ordering that will allow the grammar decoder (implicit in the proof of Theorem 6 in Section 4) to determine the name of each new variable that must be decoded (if the decoder has previously dealt with variables A_1, A_2, \dots, A_m , then the next new variable that will appear in the decoding process will be A_{m+1}).

Given any grammar G which is not in $\mathcal{G}(\mathcal{A})$, but which satisfies properties (i) and (ii), one can re-name the variables in $V(G)$ in a unique way so that properties (iii)-(v) will also be satisfied. This gives us a new grammar, denoted by $[G]$, which is a member of $\mathcal{G}(\mathcal{A})$ and which represents the same data string as G . (If a grammar G is already a member of $\mathcal{G}(\mathcal{A})$, we define $[G] = G$.) The grammar $[G]$ shall be called the *canonical form* of the grammar G .

Example 4. Consider the admissible grammar whose production rules are

$$\begin{aligned} S &\rightarrow BaA \\ A &\rightarrow aC \\ B &\rightarrow Db \\ C &\rightarrow bB \\ D &\rightarrow ab \end{aligned}$$

One sees that

$$\begin{aligned}
 f_G^0(S) &= S \\
 f_G^1(S) &= BaA \\
 f_G^2(S) &= DbbaaC \\
 f_G^3(S) &= abbaabB \\
 f_G^4(S) &= abbaabDb
 \end{aligned}$$

Multiplying these strings together as in (3.5), one obtains the string

$$SBaADbaaCabbaabBabbaabDb$$

Listing the variables in $V(G)$ in order of their first left-to-right appearance in this string, the following list results:

$$S, B, A, D, C$$

Employing this list, we re-name the variables according to the prescription

$$\begin{aligned}
 S &\rightarrow A_0 \\
 B &\rightarrow A_1 \\
 A &\rightarrow A_2 \\
 D &\rightarrow A_3 \\
 C &\rightarrow A_4
 \end{aligned}$$

thereby obtaining the grammar $[G]$ in $\mathcal{G}(\{a, b\})$ whose production rules are

$$\begin{aligned}
 A_0 &\rightarrow A_1aA_2 \\
 A_1 &\rightarrow A_3b \\
 A_2 &\rightarrow aA_4 \\
 A_3 &\rightarrow ab
 \end{aligned}$$

$$A_4 \rightarrow bA_1 \quad (3.6)$$

The grammars G and $[G]$ both represent the data string $abbaababb$.

A *grammar transform* is a mapping from \mathcal{A}^+ into $\mathcal{G}(\mathcal{A})$ such that the grammar $G_x \in \mathcal{G}(\mathcal{A})$ assigned to each \mathcal{A} -string x represents x . We adopt the notational convention of writing $x \rightarrow G_x$ to denote a grammar transform. In this notation, x is a generic variable denoting an arbitrary \mathcal{A} -string, and G_x is the grammar in $\mathcal{G}(\mathcal{A})$ assigned to x by the grammar transform.

Definition. In subsequent sections, we shall occasionally make use of a set of grammars $\mathcal{G}^*(\mathcal{A})$ which is a proper subset of $\mathcal{G}(\mathcal{A})$. The set $\mathcal{G}^*(\mathcal{A})$ consists of all grammars $G \in \mathcal{G}(\mathcal{A})$ satisfying the property that $f_G^\infty(A) \neq f_G^\infty(B)$ whenever A, B are distinct variables from $V(G)$. At this point, it is not clear to the reader why the smaller set of grammars $\mathcal{G}^*(\mathcal{A})$ is needed. This will become clear in Lemma 8 of Appendix B, where use of a grammar $G \in \mathcal{G}^*(\mathcal{A})$ to represent an \mathcal{A} -string x will allow us to set up a one-to-one correspondence between certain entries of the right members of the production rules of G and substrings of x forming the entries of a parsing of x ; this correspondence will allow us to relate the encoding of the right members of G (as described in Section 4) to the left-to-right encoding of x in the usual manner of sequential encoders.

3.1 Asymptotically Compact Grammar Transforms

If G is any context-free grammar, let $|G|$ denote the total length of the right members of the production rules of G . We say that a grammar transform $x \rightarrow G_x$ is *asymptotically compact* if both of the following properties hold:

- (i) For each \mathcal{A} -string x , the grammar G_x belongs to $\mathcal{G}^*(\mathcal{A})$.
- (ii) $\lim_{n \rightarrow \infty} \max_{x \in \mathcal{A}^n} \frac{|G_x|}{|x|} = 0$

Asymptotically compact grammar transforms are important for the following reason: Employing an asymptotically compact grammar transform as the grammar transform in Figure 1 yields a grammar based code which is universal (Theorem 7). We present here two examples of asymptotically compact grammar transforms, the *Lempel-Ziv grammar transform* and the *bisection grammar transform*.

3.1.1 Lempel-Ziv Grammar Transform

Let $\mathbf{x} = x_1x_2 \cdots x_n$ be an \mathcal{A} -string. Let (u_1, u_2, \dots, u_t) be the Lempel-Ziv parsing of \mathbf{x} , by which we mean the parsing of \mathbf{x} established in the paper [15] and used in the 1978 version of the Lempel-Ziv data compression algorithm [27]. Let $\mathcal{S}_{lz}(\mathbf{x})$ be the set of substrings of \mathbf{x} defined by

$$\mathcal{S}_{lz}(\mathbf{x}) \triangleq \{\mathbf{x}\} \cup \{u_1, u_2, \dots, u_t\}$$

For each $u \in \mathcal{S}_{lz}(\mathbf{x})$, let (s_u, a_u) be the parsing of u in which $a_u \in \mathcal{A}$, and let A^u be a variable uniquely assigned to u . Let $G_{\mathbf{x}}^{lz}$ be the admissible grammar such that

- The set of variables and the set of terminal symbols are given by

$$\begin{aligned} V(G_{\mathbf{x}}^{lz}) &= \{A^u : u \in \mathcal{S}_{lz}(\mathbf{x})\} \\ T(G_{\mathbf{x}}^{lz}) &= \{a_u : u \in \mathcal{S}_{lz}(\mathbf{x})\} \end{aligned}$$

- The start symbol is $A^{\mathbf{x}}$ and the $A^{\mathbf{x}}$ production rule is

$$A^{\mathbf{x}} \rightarrow A^{u_1} A^{u_2} \cdots A^{u_t}$$

- For each $u \in \mathcal{S}_{lz}(\mathbf{x})$ other than \mathbf{x} , the A^u production rule is

$$A^u \rightarrow A^{s_u} a_u$$

The *Lempel-Ziv grammar transform* is the mapping $\mathbf{x} \rightarrow [G_{\mathbf{x}}^{lz}]$ from \mathcal{A}^+ into $\mathcal{G}(\mathcal{A})$. For the Lempel-Ziv parsing (u_1, \dots, u_t) of an \mathcal{A} -string \mathbf{x} , let us write $t = t(\mathbf{x})$ to emphasize the dependence of the number of phrases on \mathbf{x} . It is well-known that

$$\max_{\mathbf{x} \in \mathcal{A}^n} t(\mathbf{x}) = O\left(\frac{n}{\log n}\right) \quad (3.7)$$

from which it follows that the Lempel-Ziv grammar transform is asymptotically compact.

Example 5. The Lempel-Ziv parsing of the data string $\mathbf{x} = 010010000001$ is $(0, 1, 00, 10, 000, 001)$. The grammar $G_{\mathbf{x}}^{lz}$ has the production rules

$$\begin{aligned} A^{\mathbf{x}} &\rightarrow A^0 A^1 A^{00} A^{10} A^{000} A^{001} \\ A^{00} &\rightarrow A^0 0 \\ A^{10} &\rightarrow A^1 0 \\ A^{000} &\rightarrow A^{00} 0 \\ A^{001} &\rightarrow A^{00} 1 \\ A^0 &\rightarrow 0 \\ A^1 &\rightarrow 1 \end{aligned}$$

The grammar $[G_{\mathbf{x}}^{lz}]$ can be verified by the reader to be the grammar in $\mathcal{G}^*(\{0,1\})$ with the production rules

$$\begin{aligned} A_0 &\rightarrow A_1 A_2 A_3 A_4 A_5 A_6 \\ A_1 &\rightarrow 0 \\ A_2 &\rightarrow 1 \\ A_3 &\rightarrow A_1 0 \\ A_4 &\rightarrow A_2 0 \\ A_5 &\rightarrow A_3 0 \\ A_6 &\rightarrow A_3 1 \end{aligned}$$

Discussion. The reader of [15] will find notions called producibility and reproducibility introduced there that allow one to describe a recursive copying process for certain parsings of a data string (not just the parsing considered above). For each such parsing, it is easy to construct a grammar which embodies this copying process and represents the given data string; the grammar we built in Example 5 was just one instance of this paradigm.

3.1.2 Bisection Grammar Transform

Let $\mathbf{x} = x_1x_2 \cdots x_n$ be an arbitrary \mathcal{A} -string. Let $\mathcal{S}_{bis}(\mathbf{x})$ be the following set of substrings of \mathbf{x} :

$$\mathcal{S}_{bis}(\mathbf{x}) \triangleq \{\mathbf{x}\} \cup \{(x_i, x_{i+1}, \dots, x_j) : (i-1)/(j-i+1) \text{ & } \log(j-i+1) \text{ are integers}\}$$

For each $u \in \mathcal{S}_{bis}(\mathbf{x})$, let A^u be a variable uniquely assigned to u . For each $u \in \mathcal{S}_{bis}(\mathbf{x})$ of even length, let $(s(u, L), s(u, R))$ be the parsing of u in which the strings $s(u, L)$ and $s(u, R)$ are of equal length. Let $G_{\mathbf{x}}^{bis}$ be the admissible grammar such that

- The set of variables and the set of terminal symbols are given by

$$\begin{aligned} V(G_{\mathbf{x}}^{bis}) &= \{A^u : u \in \mathcal{S}_{bis}(\mathbf{x})\} \\ T(G_{\mathbf{x}}^{bis}) &= \{u \in \mathcal{S}_{bis}(\mathbf{x}) : |u| = 1\} \end{aligned}$$

- The start symbol is $A^{\mathbf{x}}$.
- If $u \in \mathcal{S}_{bis}(\mathbf{x})$ and $|u| = 1$, the A^u production rule is

$$A^u \rightarrow u$$

- If $u \in \mathcal{S}_{bis}(\mathbf{x})$ and $\log |u|$ is a positive integer, the A^u production rule is

$$A^u \rightarrow A^{s(u,L)} A^{s(u,R)}$$

- If $u \in \mathcal{S}_{bis}(\mathbf{x})$ and $\log |u|$ is not an integer (which means that $u = \mathbf{x}$), the A^u production rule is

$$A^u \rightarrow A^{u_1} A^{u_2} \cdots A^{u_t},$$

where (u_1, u_2, \dots, u_t) is the unique parsing of \mathbf{x} into strings in $\mathcal{S}_{bis}(\mathbf{x})$ for which $|\mathbf{x}| > |u_1| > |u_2| > \dots > |u_t|$.

The *bisection grammar transform* is the mapping $\mathbf{x} \rightarrow [G_{\mathbf{x}}^{bis}]$ from \mathcal{A}^+ into $\mathcal{G}(\mathcal{A})$. In the paper [13], it is shown that the bisection grammar transform is asymptotically compact,

and a lossless compression algorithm with good redundancy properties is developed based upon the bisection grammar transform.

Example 6. For the data string $\mathbf{x} = 0001010$, we have

$$\mathcal{S}_{bis}(\mathbf{x}) = \{\mathbf{x}, 0001, 01, 0, 00, 1\},$$

and the production rules of the grammar $G_{\mathbf{x}}^{bis}$ are:

$$\begin{aligned} A^{\mathbf{x}} &\rightarrow A^{0001}A^{01}A^0 \\ A^{0001} &\rightarrow A^{00}A^{01} \\ A^{00} &\rightarrow A^0A^0 \\ A^{01} &\rightarrow A^0A^1 \\ A^0 &\rightarrow 0 \\ A^1 &\rightarrow 1 \end{aligned}$$

We then see that the production rules of $[G_{\mathbf{x}}^{bis}]$ are given by

$$\begin{aligned} A_0 &\rightarrow A_1A_2A_3 \\ A_1 &\rightarrow A_4A_2 \\ A_2 &\rightarrow A_3A_5 \\ A_3 &\rightarrow 0 \\ A_4 &\rightarrow A_3A_3 \\ A_5 &\rightarrow 1 \end{aligned}$$

3.2 Irreducible Grammar Transforms

We define a context-free grammar G to be *irreducible* if the following four properties are satisfied:

(a.1) G is admissible.

(a.2) If v_1, v_2 are distinct variables in $V(G)$, then $f_G^\infty(v_1) \neq f_G^\infty(v_2)$.

- (a.3) Every variable in $V(G)$ other than the start symbol appears at least twice as an entry in the right members of the production rules of the grammar G .
- (a.4) There does not exist any pair Y_1, Y_2 of symbols in $V(G) \cup T(G)$ such that the string $Y_1 Y_2$ appears more than once in nonoverlapping positions as a substring of the right members of the production rules for G .

Example 7. The admissible grammar with production rules

$$\begin{aligned}
S &\rightarrow ACBBEA \\
A &\rightarrow DD1 \\
B &\rightarrow C10 \\
C &\rightarrow 0D \\
D &\rightarrow 0E \\
E &\rightarrow 11
\end{aligned} \tag{3.8}$$

can be verified to be an irreducible grammar.

A grammar transform $\mathbf{x} \rightarrow G_{\mathbf{x}}$ is defined to be an *irreducible grammar transform* if each grammar $G_{\mathbf{x}}$ is irreducible. In principle, it is easy to obtain irreducible grammar transforms. One can start with any grammar transform $\mathbf{x} \rightarrow G_{\mathbf{x}}$ and exploit the presence of matching substrings in the right members of the production rules of each $G_{\mathbf{x}}$ to reduce $G_{\mathbf{x}}$ to an irreducible grammar representing \mathbf{x} in finitely many reduction steps. A wealth of different irreducible grammar transforms are obtained by doing the reductions in different ways. In Section 6, we develop a systematic approach for reducing grammars to irreducible grammars, and present examples of irreducible grammar transforms which have yielded good performance in compression experiments on real data.

4 Entropy and Coding of Grammars

In this section, we define the entropy $H(G)$ of a grammar $G \in \mathcal{G}(\mathcal{A})$, and present a result (Theorem 6) stating that we can losslessly encode each G using approximately $H(G)$ codebits.

First, we need to define the concept of *unnormalized entropy*, which will be needed in this section and in subsequent parts of the paper. Suppose u is either a string $u_1 u_2 \cdots u_n$ in a multiplicative monoid or a parsing (u_1, u_2, \dots, u_n) of a string in a multiplicative monoid. For each $s \in \{u_1, u_2, \dots, u_n\}$, let $m(s|u)$ be the number of entries of u which are equal to s :

$$m(s|u) = |\{1 \leq i \leq n : u_i = s\}|$$

We define the unnormalized entropy of u to be the following nonnegative real number $H^*(u)$:

$$H^*(u) \triangleq \sum_{i=1}^n \log \left(\frac{n}{m(u_i|u)} \right)$$

Let G be an arbitrary grammar in $\mathcal{G}(\mathcal{A})$; recalling the notation we introduced in Section 3, we have $V(G) = \{A_0, A_1, A_2, \dots, A_{|V(G)|-1}\}$. We define ρ_G to be the following string of length $|G|$:

$$\rho_G \triangleq f_G(A_0)f_G(A_1)f_G(A_2)\cdots f_G(A_{|V(G)|-1}) \quad (4.1)$$

Notice that the string ρ_G is simply the product of the right members of the production rules in $P(G)$. We define ω_G to be the string obtained from ρ_G by removing from ρ_G the first left-to-right appearance of each variable in $\{A_1, \dots, A_{|V(G)|-1}\}$. We define the entropy $H(G)$ of the grammar G to be the number

$$H(G) \triangleq H^*(\omega_G)$$

Theorem 6 *There is a one-to-one mapping $B : \mathcal{G}(\mathcal{A}) \rightarrow \{0, 1\}^+$ such that*

- *If G_1 and G_2 are distinct grammars in $\mathcal{G}(\mathcal{A})$, then the binary codeword $B(G_1)$ is not a prefix of the binary codeword $B(G_2)$.*
- *For each $G \in \mathcal{G}(\mathcal{A})$, the length of the codeword $B(G)$ satisfies*

$$|B(G)| \leq |\mathcal{A}| + 4|G| + \lceil H(G) \rceil \quad (4.2)$$

Proof. Given the grammar $G \in \mathcal{G}(\mathcal{A})$, the binary codeword $B(G)$ has a parsing $(B_1, B_2, B_3, B_4, B_5, B_6)$ in which

- (i) B_1 has length $|V(G)|$ and indicates what $V(G)$ is. (Specifically, B_1 consists of $|V(G)| - 1$ zeroes followed by a one.)
- (ii) B_2 has length $|\mathcal{A}|$ and tells what $T(G)$ is. (For each element of \mathcal{A} , transmit a codebit to indicate whether or not that element is in $T(G)$.)
- (iii) B_3 has length $|G|$ and indicates the frequency with which each symbol in $(V(G) \cup T(G)) - \{S\}$ appears in the right members of the production rules of G . (Each frequency is given a unary representation as in (i).)
- (iv) B_4 has length $|G|$ and indicates the lengths of the right members of the production rules of G .
- (v) B_5 has length $|G|$ and indicates which entries of ρ_G are variables in $V(G)$ appearing for the first time as ρ_G is scanned from left to right.
- (vi) B_6 has length at most $[H(G)]$ and indicates what ω_G is. The well-known enumerative encoding technique [4] is used to obtain B_6 from ω_G . This technique exploits the frequencies of symbols in ω_G learned from B_3 to encode ω_G into a codeword of length equal to the smallest integer greater than or equal to the logarithm of the size of the type class of ω_G (see [6] or the beginning of Appendix B). From the definition of $H(G)$ and a standard bound on the size of a type class ([6], Lemma 2.3), it is clear that the codeword length can be no more than $[H(G)]$.

From ω_G and the information conveyed by B_5 , the string ρ_G can be reconstructed, since new variables in ρ_G are numbered consecutively as they first appear. From ρ_G and the information conveyed by B_4 , the right members of the production rules in G can be obtained, completing the reconstruction of G from $B(G)$. The total length of the strings B_1, B_2, \dots, B_6 is at most the right side of (4.2).

Example 8. Consider the grammar $G \in \mathcal{G}(\mathcal{A})$ with the production rules given in (3.6). We have

$$\rho_G = A_1aA_2A_3baA_4abbA_1$$

$$\begin{aligned}\omega_G &= abaabbA_1 \\ H(G) &= H^*(\omega_G) = 3 \log\left(\frac{7}{3}\right) + 3 \log\left(\frac{7}{3}\right) + \log 7 = 10.14\end{aligned}$$

Substituting $H(G)$, $|G| = 11$, and $|\mathcal{A}| = 2$ into (4.2), we see that the codeword $B(G)$ is of length no more than 57.

5 Coding Theorems

We embark upon the main section of the paper. A formal definition of the concept of grammar based code is given. Specific redundancy bounds for a grammar based code with respect to families of finite state information sources (Theorems 7 and 8) are obtained.

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

$$\begin{aligned}\mu(1_{\mathcal{A}}) &= 1 \\ \mu(\mathbf{x}) &= \sum_{a \in \mathcal{A}} \mu(\mathbf{x}a), \quad \mathbf{x} \in \mathcal{A}^*\end{aligned}$$

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:

$$\sum_{s, x} p(s, x|s') = 1, \quad s' \in \mathcal{S} \tag{5.3}$$

$$\mu(x_1 x_2 \cdots x_n) = \sum_{s_1, s_2, \dots, s_n \in \mathcal{S}} \prod_{i=1}^n p(s_i, x_i | s_{i-1}), \quad x_1 x_2 \cdots x_n \in \mathcal{A}^+ \tag{5.4}$$

We let $\Lambda_{fs}^k(\mathcal{A})$ denote the family of all alphabet \mathcal{A} k -th order finite state sources. We call members of the set $\cup_k \Lambda_{fs}^k(\mathcal{A})$ *finite state sources*.

Remark. If in addition to (5.3)-(5.4), we require that for each (x, s') , the quantity $p(s, x|s')$ is nonvanishing for just one s , then the finite state source μ is said to be *unifilar*. We point out that our definition of finite state source includes sources which are not unifilar as well as those which are unifilar.

Stationary Sources. We define $\Lambda_{sta}(\mathcal{A})$ to be the set of all alphabet \mathcal{A} information sources μ for which

$$\mu(\mathbf{x}) = \sum_{a \in \mathcal{A}} \mu(a\mathbf{x}), \quad \mathbf{x} \in \mathcal{A}^+$$

The members of $\Lambda_{sta}(\mathcal{A})$ are called *stationary sources*.

Lossless source codes. We define an *alphabet \mathcal{A} lossless source code* to be a pair $\phi = (\epsilon_\phi, \delta_\phi)$ in which

- (i) ϵ_ϕ is a mapping (called the encoder of ϕ) which maps each \mathcal{A} -string \mathbf{x} into a codeword $\epsilon_\phi(\mathbf{x}) \in \{0,1\}^+$, and δ_ϕ is the mapping (called the decoder of ϕ) which maps $\epsilon_\phi(\mathbf{x})$ back into \mathbf{x} ; and
- (ii) for each $n = 1, 2, \dots$, and each distinct pair of strings $\mathbf{x}_1, \mathbf{x}_2$ in \mathcal{A}^n , the codeword $\epsilon_\phi(\mathbf{x}_1)$ is not a prefix of the codeword $\epsilon_\phi(\mathbf{x}_2)$.

An alphabet \mathcal{A} lossless source code ϕ is defined to be an alphabet \mathcal{A} *grammar based code* if there is a grammar transform $\mathbf{x} \rightarrow G_{\mathbf{x}}$ such that

$$\epsilon_\phi(\mathbf{x}) = B(G_{\mathbf{x}}), \quad \mathbf{x} \in \mathcal{A}^+$$

The grammar transform in this definition shall be called the *grammar transform underlying the grammar based code ϕ* . We isolate two classes of grammar based codes. We let $\mathcal{C}_{ac}(\mathcal{A})$ be the class of all alphabet \mathcal{A} grammar based codes for which the underlying grammar transform is asymptotically compact. We let $\mathcal{C}_{irr}(\mathcal{A})$ denote the class of all alphabet \mathcal{A} grammar based codes for which the underlying grammar transform is irreducible.

Redundancy Results. The type of redundancy we employ in this paper is *maximal pointwise redundancy*, a notion of redundancy that has been studied previously [20] [23]. Let Λ be a family of alphabet \mathcal{A} information sources. Let ϕ be an alphabet \mathcal{A} lossless source code. The n -th order maximal pointwise redundancy of ϕ with respect to the family of sources Λ is the number defined by

$$\text{Red}_n(\phi, \Lambda) \triangleq n^{-1} \max_{\mathbf{x} \in \mathcal{A}^n} \sup_{\mu \in \Lambda} [|\epsilon_\phi(\mathbf{x})| + \log \mu(\mathbf{x})] \quad (5.5)$$

We present two results concerning the asymptotic behavior of the maximal pointwise redundancy for alphabet \mathcal{A} grammar based codes with respect to each family of sources $\Lambda_{fs}^k(\mathcal{A})$ ($k \geq 1$). These are the main results of this paper.

Theorem 7 *Let ϕ be a grammar based code from the class $\mathcal{C}_{ac}(\mathcal{A})$, and let $\mathbf{x} \rightarrow G_{\mathbf{x}}$ be the grammar transform underlying ϕ . Let $\{\nu_n\}$ be a sequence of positive numbers converging to zero such that*

$$\max_{\mathbf{x} \in \mathcal{A}^n} \frac{|G_{\mathbf{x}}|}{|\mathbf{x}|} = O(\nu_n)$$

Then, for every positive integer k ,

$$\text{Red}_n(\phi, \Lambda_{fs}^k(\mathcal{A})) = O(\gamma(\nu_n)) \quad (5.6)$$

where γ is the function defined by

$$\gamma(x) \triangleq x \log(1/x), \quad x > 0$$

Theorem 8 *The class of codes $\mathcal{C}_{irr}(\mathcal{A})$ is a subset of the class of codes $\mathcal{C}_{ac}(\mathcal{A})$. Furthermore, for every positive integer k ,*

$$\max_{\phi \in \mathcal{C}_{irr}(\mathcal{A})} \text{Red}_n(\phi, \Lambda_{fs}^k(\mathcal{A})) = O\left(\frac{\log \log n}{\log n}\right)$$

Remarks.

- (i) Theorem 7 tells us that the maximal pointwise redundancies asymptotically decay to zero for each code in $\mathcal{C}_{ac}(\mathcal{A})$; the speed of decay is dependent upon the code. Theorem 8 tells us that the maximal pointwise redundancies decay to zero uniformly over the class of codes $\mathcal{C}_{irr}(\mathcal{A})$, with the uniform speed of decay at least as fast as a constant times $\log \log n / \log n$.
- (ii) An alphabet \mathcal{A} lossless source code ϕ is said to be a *weakly minimax universal code* [7] with respect to a family of alphabet \mathcal{A} information sources Λ if

$$\lim_{n \rightarrow \infty} n^{-1} \sum_{\mathbf{x} \in \mathcal{A}^n} (|\epsilon_\phi(\mathbf{x})| + \log \mu(\mathbf{x})) \mu(\mathbf{x}) = 0, \quad \mu \in \Lambda$$

Theorem 7 tells us that every code in $\mathcal{C}_{ac}(\mathcal{A})$ is a weakly minimax universal code with respect to the family of sources $\cup_k \Lambda_{fs}^k(\mathcal{A})$. It is then automatic that the codes in $\mathcal{C}_{ac}(\mathcal{A})$ are each weakly minimax with respect to the family of sources $\Lambda_{sta}(\mathcal{A})$ (easily established using Markov approximations of stationary sources [9]).

- (iii) An alphabet \mathcal{A} lossless source code ϕ is said to be a *minimax universal code* [7] with respect to a family of alphabet \mathcal{A} information sources Λ if

$$\lim_{n \rightarrow \infty} n^{-1} \sup_{\mu \in \Lambda} \sum_{\mathbf{x} \in \mathcal{A}^n} (|\epsilon_\phi(\mathbf{x})| + \log \mu(\mathbf{x})) \mu(\mathbf{x}) = 0$$

Theorem 7 tells us that every code in $\mathcal{C}_{ac}(\mathcal{A})$ is a minimax universal code with respect to each family of sources $\Lambda_{fs}^k(\mathcal{A})$, $k \geq 1$.

- (iv) J. Ziv and A. Lempel define an *individual sequence* to be an infinite sequence (x_1, x_2, x_3, \dots) each of whose entries x_i belongs to the alphabet \mathcal{A} . These authors [27] [26] have put forth a notion of what it means for an alphabet \mathcal{A} lossless source code to be a universal code with respect to the family of individual sequences. (Leaving aside the technical details, we point out that Ziv and Lempel define a class of lossless codes called finite state codes, and define a code to be universal if it encodes each individual sequence asymptotically as well as each finite state code.) It can be shown that if an alphabet \mathcal{A} lossless source code ϕ satisfies

$$\limsup_{n \rightarrow \infty} \left[n^{-1} \sup_{\mu \in \Lambda_{fs}^k(\mathcal{A})} (|\epsilon_\phi(x_1 x_2 \cdots x_n)| + \log \mu(x_1 x_2 \cdots x_n)) \right] \leq 0$$

for every $k \geq 1$ and every individual sequence (x_1, x_2, \dots) , then ϕ is a universal code with respect to the family of individual sequences. This fact, together with Theorem 7, allows us to conclude that every code in $\mathcal{C}_{ac}(\mathcal{A})$ is a universal code with respect to the family of individual sequences.

The following two lemmas, together with Theorem 6, immediately imply Theorems 7 and 8. They are proved in Appendix B.

Lemma 1 *Let \mathbf{x} be any \mathcal{A} -string, and let G be any grammar in $\mathcal{G}^*(\mathcal{A})$ which represents*

x. Then, for every positive integer k , and every $\mu \in \Lambda_{fs}^k(\mathcal{A})$,

$$H(G) \leq -\log \mu(\mathbf{x}) + |G|(2 + \log k) + 2|\mathbf{x}| \gamma \left(\frac{|G| - |V(G)| + 1}{|\mathbf{x}|} \right) \quad (5.7)$$

Lemma 2 Let \mathbf{x} be any \mathcal{A} -string of length at least $|\mathcal{A}|^{34}$. Then

$$\frac{|G|}{|\mathbf{x}|} \leq \frac{|\mathcal{A}|}{|\mathbf{x}|} + \frac{12 \log |\mathcal{A}|}{\log |\mathbf{x}| - 8 \log |\mathcal{A}| - 8} \quad (5.8)$$

for any irreducible grammar G which represents \mathbf{x} .

In concluding this section, we remark that our grammar based encoding technique and Theorems 7 and 8 based on it are predicated on the implicit assumption that a data string \mathbf{x} is first batch processed before formation of a grammar representing \mathbf{x} ; only after the batch processing and grammar formation can the grammar then be encoded. An approach involving less delay is to form and encode production rules of a grammar on the fly as we sequentially process the data from left to right, with the grammar encoding terminating simultaneously with the sequential processing of the last data sample. The Improved Sequential Algorithm of [25] adopts this approach, necessitating a different method for encoding grammars than used in Section 4, as well as new proofs of the universal coding theorems.

6 Reduction Rules

We present five reduction rules, such that if an admissible grammar G is not irreducible, there will be at least one of the five reduction rules which can be applied to the grammar G ; any of these rules applicable to G will produce a new admissible grammar G' satisfying the properties

- (i) G' represents the same data string that is represented by G .
- (ii) G' is closer to being irreducible than G (in a sense made clear in the discussion just prior to Section 6.1).

Reduction Rule 1. Let G be an admissible grammar. Let A be a variable of G that appears only once in the right members of the production rules of G . Let $B \rightarrow \alpha A \beta$ be the unique production rule in which A appears in the right member. Let $A \rightarrow \gamma$ be the A production rule of G . Simultaneously do the following to the set of production rules of G :

- Delete the production rule $A \rightarrow \gamma$ from the production rules of G .
- Replace the production rule $B \rightarrow \alpha A \beta$ with the production rule $B \rightarrow \alpha \gamma \beta$.

Let P' be the resulting smaller set of production rules. Define G' to be the unique admissible grammar whose set of production rules is P' .

Reduction Rule 2. Let G be an admissible grammar. Suppose there is a production rule

$$A \rightarrow \alpha_1 \beta \alpha_2 \beta \alpha_3 \quad (6.9)$$

where $|\beta| \geq 2$. Let B be a symbol which does not belong to $V(G) \cup T(G)$. Perform the following operations simultaneously to $P(G)$:

- Replace the rule (6.9) with the rule

$$A \rightarrow \alpha_1 B \alpha_2 B \alpha_3$$

- Append the rule $B \rightarrow \beta$.

Let P' be the resulting set of production rules. Define G' to be the unique admissible grammar whose set of production rules is P' .

Reduction Rule 3. Let G be an admissible grammar. Suppose there are two distinct production rules of form

$$A \rightarrow \alpha_1 \beta \alpha_2 \quad (6.10)$$

$$B \rightarrow \alpha_3 \beta \alpha_4 \quad (6.11)$$

where β is of length at least two, either α_1 or α_2 is not the empty string, and either α_3 or α_4 is not the empty string. Let C be a symbol not appearing in $V(G) \cup T(G)$. Perform the following operations simultaneously to $P(G)$:

- Replace the rule (6.10) with the rule

$$A \rightarrow \alpha_1 C \alpha_2$$

- Replace the rule (6.11) with the rule

$$B \rightarrow \alpha_3 C \alpha_4$$

- Append the rule

$$C \rightarrow \beta$$

Let P' be the resulting set of production rules. Define G' to be the unique admissible grammar whose set of production rules is P' .

Reduction Rule 4. Let G be an admissible grammar. Suppose there are two distinct production rules of the form

$$\begin{aligned} A &\rightarrow \alpha_1 \beta \alpha_2 \\ B &\rightarrow \beta \end{aligned} \tag{6.12}$$

where β is of length at least two, and either α_1 or α_2 is not the empty string. In $P(G)$, replace the production rule (6.12) with the production rule

$$A \rightarrow \alpha_1 B \alpha_2$$

Let P' be the resulting set of production rules. Define G' to be the unique admissible grammar whose set of production rules is P' .

Reduction Rule 5. Let $G = (V, T, P, S)$ be an admissible grammar. Suppose there exist distinct variables $A, B \in V$ such that $f_G^\infty(A) = f_G^\infty(B)$. Let P^* be the set of production rules that results by substituting A for each appearance of B in the right members of the production rules in P . Let U be the set of those variables in V which are useless symbols with respect to the grammar (V, T, P^*, S) . (Note that U is nonempty, because $B \in U$.) Let P' be the set of production rules obtained by removing from P^*

all production rules whose left member is in U . Define G' to be the unique admissible grammar whose set of production rules is P' .

Example 9. Consider the admissible grammar G whose production rules are

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow CD \\ B &\rightarrow aE \\ C &\rightarrow ab \\ D &\rightarrow cd \\ E &\rightarrow bD \end{aligned}$$

Notice that $f_G^\infty(A) = f_G^\infty(B) = abcd$. Replace every B on the right with A :

$$\begin{aligned} S &\rightarrow AA \\ A &\rightarrow CD \\ B &\rightarrow aE \\ C &\rightarrow ab \\ D &\rightarrow cd \\ E &\rightarrow bD \end{aligned} \tag{6.13}$$

Consider the grammar G^* in which $V(G^*) = \{S, A, B, C, D, E\}$, $T(G^*) = \{a, b, c, d\}$, and $P(G^*)$ is the set of production rules listed in (6.13). Let us compute the members of $V(G^*)$ which are useless symbols with respect to the grammar G^* :

We have:

$$\begin{aligned} f_{G^*}^0(S) &= S \\ f_{G^*}^1(S) &= AA \\ f_{G^*}^2(S) &= CDCD \\ f_{G^*}^n(S) &= abcdabcd, n \geq 3 \end{aligned} \tag{6.14}$$

The useless members of $V(G^*)$ are the members of $V(G^*)$ not appearing in the right hand sides of the equations in (6.14). These are the variables B and E . Removing the two production rules from the list (6.13) which have B and E as left members, we obtain the set of production rules

$$\begin{aligned} S &\rightarrow AA \\ A &\rightarrow CD \\ C &\rightarrow ab \\ D &\rightarrow cd \end{aligned}$$

which uniquely defines an admissible grammar G' . The reader can verify that the grammars G and G' both represent the data string $abcdabcd$.

Example 10. Consider the data string

$$\mathbf{x} = 01101110011001110001110110110111$$

We will obtain an irreducible grammar representing \mathbf{x} in finitely many reduction steps, where on each reduction step, one of the Reduction Rules 2-4 is used. We start with the list of production rules consisting of just one rule:

$$S \rightarrow \mathbf{x}$$

Applying Reduction Rule 2, we get the list of production rules:

$$\begin{aligned} S &\rightarrow A001100111000111011A \\ A &\rightarrow 0110111 \end{aligned}$$

We apply Reduction Rule 2 again, getting the list of production rules

$$\begin{aligned} S &\rightarrow A0011BB11A \\ A &\rightarrow 0110111 \end{aligned}$$

$$B \rightarrow 001110$$

Applying Reduction Rule 3, we obtain:

$$\begin{aligned} S &\rightarrow ACBB11A \\ A &\rightarrow 0110111 \\ B &\rightarrow C10 \\ C &\rightarrow 0011 \end{aligned}$$

The following is then obtained via application of Reduction Rule 2 followed by Reduction Rule 4:

$$\begin{aligned} S &\rightarrow ACBB11A \\ A &\rightarrow DD1 \\ B &\rightarrow C10 \\ C &\rightarrow 0D \\ D &\rightarrow 011 \end{aligned}$$

Applying Reduction Rule 3 at this point yields the list of production rules (3.8), which is seen to define an irreducible grammar. This grammar will automatically represent the string \mathbf{x} .

Discussion. Notice that in the preceding example, we started with a grammar representing our data string and obtained an irreducible grammar representing the same string via finitely many reduction steps, in which each reduction step involved exactly one of the Reduction Rules 1-5. How can we be sure that it is always possible to do this? To answer this question, define

$$C(G) \triangleq 2|G| - |V(G)|$$

for any admissible grammar G . The number $C(G)$ is a positive integer for any admissible grammar G . Also, the reader can check that if the grammar G' is obtained from the grammar G by applying one of the Reduction Rules 1-5, then $C(G') < C(G)$. From these facts, it follows that if we start with a grammar G which is not irreducible, then in at most

$C(G) - 1$ reduction steps (in which each reduction step involves the application of exactly one of the Reduction Rules 1-5), we will arrive at an irreducible grammar representing the same data string as G . It does not matter how the reductions are done—they will always lead to an irreducible grammar in finitely many steps.

Remark. It is possible to define more reduction rules than Reduction Rules 1-5. For example, if the right members of the production rules of a grammar G contain nonoverlapping substrings $\alpha \neq \alpha'$ for which $f_G^\infty(\alpha) = f_G^\infty(\alpha')$, one can reduce G by replacing α, α' with a new variable A , while introducing a new production rule (either $A \rightarrow \alpha$ or $A \rightarrow \alpha'$). This new reduction rule is somewhat difficult to implement in practice, however. We limited ourselves to Reduction Rules 1-5 because

- Reduction Rules 1-5 are simple to implement.
- Reduction Rules 1-5 yield grammars which are sufficiently reduced so as to yield excellent data compression capability (Theorem 8).

Remark. Cook et al. [3] developed a hill climbing search process to infer a simple grammar G whose language $L(G)$ contains a given set of strings S . The grammar inferred by their algorithm locally minimizes an objective function $M(G|S)$ which measures the “goodness of fit” of the grammar G to the set of strings S . It is interesting to note that Reduction Rules 1-4 were proposed in [3] as part of the search process, along with some other rules. However, unlike our approach in the present paper, Cook et al. do a reduction step only if the objective function is made smaller by doing so.

Using Reduction Rules 1-5, it is possible to design a variety of irreducible grammar transforms. We discuss two of these, the *longest matching substring algorithm* and the *modified SEQUITUR algorithm*.

6.1 Longest Matching Substring Algorithm

For a given data string \mathbf{x} , start with the trivial grammar consisting of the single production rule $S \rightarrow \mathbf{x}$, and then look for a substring of \mathbf{x} that is as long as possible and appears in at least two nonoverlapping positions in \mathbf{x} . (We call such a substring a *longest matching substring*.) A first round of reductions using Reduction Rules 2-4 is then performed, in which each nonoverlapping appearance of the longest matching substring is replaced by

a variable, resulting in a new grammar. In subsequent rounds of reductions, one first detects a longest matching substring (the longest \mathcal{A} -string appearing in nonoverlapping positions in the right members of the previously constructed grammar), and then applies Reduction Rules 2-4 to obtain a new grammar. The rounds of reductions terminate as soon as a grammar is found for which no longest matching substring can be found. This grammar is irreducible and represents \mathbf{x} . Calling this grammar $G_{\mathbf{x}}$, we have defined a grammar transform $\mathbf{x} \rightarrow G_{\mathbf{x}}$. This grammar transform is the longest matching substring algorithm. Example 10 illustrates the use of the longest matching substring algorithm. In each list of production rules that was generated in Example 10, the right member of the last rule listed is the longest matching substring that was used in the round of reductions that led to that list.

6.2 Modified SEQUITUR Algorithm

Process the data string $\mathbf{x} = x_1x_2 \cdots x_n$ one data sample at a time, from left to right. Irreducible grammars are generated recursively, with the i -th grammar G_i representing the first i data samples. Each new data sample x_i is appended to the right end of the right member of the S production rule of the previous grammar G_{i-1} , and then reductions take place to generate the next grammar G_i before the next sample x_{i+1} is processed. Since only one sample is appended at each stage of recursive grammar formation, the reductions that need to be performed to recursively generate the irreducible grammars $\{G_i\}$ are simple. The final grammar G_n is an irreducible grammar which represents the entire data string \mathbf{x} . Calling this final grammar $G_{\mathbf{x}}$, we have defined a grammar transform $\mathbf{x} \rightarrow G_{\mathbf{x}}$. We call this transform the modified SEQUITUR algorithm because of its resemblance to the SEQUITUR algorithm studied in the papers [18] [19].

Remark. The SEQUITUR algorithm [18] [19] can generate a grammar $G_{\mathbf{x}}$ representing a data string \mathbf{x} which is not a member of the set of grammars $\mathcal{G}^*(A)$, and therefore we cannot apply Theorem 8 to the SEQUITUR algorithm. It is an open problem whether the SEQUITUR algorithm leads to a universal source code. On the other hand, the modified SEQUITUR algorithm does lead to a universal source code.

7 Conclusions

We conclude the paper by embedding our grammar based coding approach into a general framework which lends perspective to the approach and allows one to more easily relate the approach to other source coding approaches.

Our general framework employs Turing machines. We adopt the usual definition of Turing machine (see [16], pp. 26-27), considering all Turing machines whose output alphabet is the set \mathcal{A} . Each Turing machine possesses a doubly-infinite tape consisting of cells

$$\dots, C_{-2}, C_{-1}, C_0, C_1, C_2, \dots$$

which are linked together from left to right in the indicated order; each cell C_i can store a symbol from \mathcal{A} or else its content is blank. There is also a read/write head which can be positioned over any of the cells on the machine tape. A Turing machine executes a computation by going through finitely or infinitely many computational cycles, possibly changing its machine state during each cycle. A computational cycle of a Turing machine consists of an operation of one of the following two types:

- (i) Read/write head is moved one cell to the left or right of its current position, and the machine moves to a new state or stays in the same state.
- (ii) Read/write head stays in its current position, and either a symbol from \mathcal{A} or a blank is written into the cell below, replacing the previous cell content, or the machine state is changed (or both).

In our use of Turing machines, we differ from the standard approach in that we do not program our Turing machines. (A Turing machine is programmed by placing finitely many inputs in the cells of the machine tape before the machine goes through its first computational cycle—the inputs can be varied to induce the machine to produce different computations.) We always assume that in performing a computation using a given Turing machine, the initial configuration of the machine’s input tape is “all cells blank” (in other words, the input to the machine is the empty string). By initializing the machine’s tape cells to be blank, the machine is set up to do one and only one computation. (Nothing is lost by doing this—if a string is computed using a Turing machine whose

initial tape configuration has finitely many nonblank cells, it is not hard to construct another Turing machine which starts with blank cells and emulates the computations done on the first machine after finitely many computational cycles.) When a Turing machine does a computation, either the computation goes on forever or the machine halts after finitely many computational cycles. We say that a Turing machine computes an \mathcal{A} -string $x_1x_2\dots x_n$ if the machine halts with consecutive tape cells C_1, C_2, \dots, C_n having contents x_1, x_2, \dots, x_n , respectively, and with every other tape cell having blank content. The reader now sees that in our formulation, given a Turing machine T , either (i) there exists exactly one \mathcal{A} -string \mathbf{x} such that T computes \mathbf{x} , or else the machine T computes no string in \mathcal{A}^+ (meaning that the machine did not halt, or else halted with cell contents not of the proper form described previously).

General Framework. Let $\tau = (T_1, T_2, \dots)$ be any sequence of Turing machines such that the following property holds:

$$\forall \mathbf{x} \in \mathcal{A}^+, \quad T_i \text{ computes } \mathbf{x} \text{ for at least one } i \quad (7.15)$$

Let B_1, B_2, B_3, \dots be the lexicographical ordering of all binary strings in $\{0, 1\}^+$. (This is the sequence 0, 1, 00, 01, 10, 11, 000, 001, etc.) Define

$$C(\mathbf{x}|\tau) \triangleq \min\{|B_i| : T_i \text{ computes } \mathbf{x}\}, \quad \mathbf{x} \in \mathcal{A}^+$$

Also, define a lossless alphabet \mathcal{A} source code to be a τ based code if for each \mathcal{A} -string \mathbf{x} , the codeword into which \mathbf{x} is encoded is a $B_i \in \{B_1, B_2, \dots\}$ such that T_i computes \mathbf{x} .

The following coding theorem is an easy consequence of these definitions.

Theorem 9 *Let $\tau = (T_1, T_2, \dots)$ satisfy (7.15). Then*

(a) *for any τ based code ϕ ,*

$$C(\mathbf{x}|\tau) \leq |\epsilon_\phi(\mathbf{x})|, \quad \mathbf{x} \in \mathcal{A}^+$$

(b) *there exists a τ based code ϕ such that*

$$C(\mathbf{x}|\tau) = |\epsilon_\phi(\mathbf{x})|, \quad \mathbf{x} \in \mathcal{A}^+ \quad (7.16)$$

Discussion. Let us call a τ based code satisfying (7.16) an optimal τ based code. Let us call the function $\mathbf{x} \rightarrow C(\mathbf{x}|\tau)$ from \mathcal{A}^+ to $\{1, 2, 3, \dots\}$ the τ complexity function. Theorem 9 tells us that there is an optimal τ based code, and that its codeword length performance is governed by the τ complexity function.

By changing τ , we get different families of τ based codes, as the following two examples indicate.

Example 11. Let $\tau = (T_1, T_2, \dots)$ be an effective enumeration of all Turing machines, as described in ([16], Section 1.4.1). The τ complexity function is then the Kolmogorov complexity function ([16], pp. 90–91). The family of τ based codes is very wide. To see this, let $\mathcal{C}_{rec}(\mathcal{A})$ be the family of all alphabet \mathcal{A} lossless source codes whose encoder is a one-to-one total recursive function on \mathcal{A}^+ and whose decoder is a partial recursive function on $\{0, 1\}^+$. Let ϕ be a code in $\mathcal{C}_{rec}(\mathcal{A})$. Using the Invariance Theorem of Kolmogorov complexity theory ([16], Section 2.1), one can show that there is a positive constant C and a τ based code ϕ' in $\mathcal{C}_{rec}(\mathcal{A})$ such that

$$\epsilon_{\phi'}(\mathbf{x}) \leq \epsilon_\phi(\mathbf{x}) + C, \quad \mathbf{x} \in \mathcal{A}^+$$

On the other hand, any optimal τ based code is not a member of $\mathcal{C}_{rec}(\mathcal{A})$, because, if it were, there would be a computable version of the Kolmogorov complexity function, and this is known not to be true (the paper [12] gives a rather strong refutation).

Example 12. Let

$$G^1, G^2, G^3, G^4, \dots$$

be the ordering of the grammars in $\mathcal{G}(\mathcal{A})$ such that the corresponding codewords

$$B(G^1), B(G^2), B(G^3), B(G^4), \dots$$

are in lexicographical order. For each $G \in \mathcal{G}(\mathcal{A})$, define the new codeword $B(G)^*$ in which $B(G)^* = B_i$ for that i for which $G^i = G$. Since $|B(G)^*| \leq |B(G)|$ for every G , we lose nothing by redefining the concept of grammar based code to use the codewords $\{B(G)^* : G \in \mathcal{G}(\mathcal{A})\}$ instead of the codewords $\{B(G) : G \in \mathcal{G}(\mathcal{A})\}$. Accordingly, let us now define a code ϕ to be a grammar based code if there exists a grammar transform

$\mathbf{x} \rightarrow G_{\mathbf{x}}$ for which

$$\epsilon_\phi(\mathbf{x}) = B(G_{\mathbf{x}})^*, \quad \mathbf{x} \in \mathcal{A}^+$$

For each grammar G in $\mathcal{G}(\mathcal{A})$, one can construct a Turing machine $T(G)$ with control function based on the production rules of G , which computes the data string represented by G . Let $\tau = (T(G^1), T(G^2), \dots)$. The family of τ based codes is the family of grammar based codes. Therefore, an optimal τ based code is an optimal grammar based code. It can be seen that there is an optimal grammar based code belonging to the family of codes $\mathcal{C}_{rec}(\mathcal{A})$ introduced in Example 11. The complexity function $\mathbf{x} \rightarrow C(\mathbf{x}|\tau)$, which describes the codeword length performance of optimal grammar based codes, is computable, unlike the Kolmogorov complexity function (although we conjecture that there is no polynomial time algorithm which computes an optimal grammar based code or this complexity function). Future research could focus on obtaining bounds on the complexity function $\mathbf{x} \rightarrow C(\mathbf{x}|\tau)$ so that we could have a better idea how optimal grammar based codes perform.

We conclude the paper by remarking that the Chomsky hierarchy of grammars ([10], Chapter 9) can be mined to provide other instances in which it might be useful to look at a family of τ based codes for a sequence of machines τ associated with a set of grammars. To illustrate, the set of context-sensitive grammars belongs to the Chomsky hierarchy. Each data string could be represented using a context-sensitive grammar, and then a machine could be constructed which computes the data string, using the production rules of the grammar as part of the machine's logic. Letting τ be an enumeration of these machines, the corresponding family of τ based codes (which is strictly larger than the family of grammar based codes of this paper), might contain codes of practical significance that are waiting to be discovered.

Acknowledgements. The authors are grateful to W. Evans, R. Maier, A. Mantilla, G. Nelson, M. Weinberger, and S. Yakowitz for helpful comments concerning this work. Special thanks are due to Professors M. Marcellin of the University of Arizona Department of Electrical and Computer Engineering, H. Flaschka of the University of Arizona Department of Mathematics, and J. Massey of the Swiss Federal Institute of Technology (Zürich, Switzerland) for arranging for financial support during the first au-

thor's 1996–97 sabbatical that helped to make the completion of this work possible.

8 Appendix A

In this Appendix, we prove Theorem 2 by means of a sequence of lemmas.

Lemma 3 *Let G be a deterministic context-free grammar such that the empty string is not the right member of any production rule of G . Suppose that the derivation graph of G is acyclic. Let u be a string in $(V(G) \cup T(G))^+$ which is not a string in $T(G)^+$. Then there exists a variable $A \in V(G)$ such that both of the following hold:*

- *A is an entry of u .*
- *A is not an entry of any of the strings $f_G^i(u)$, $i = 1, 2, \dots$*

Proof. We suppose that the conclusion of the lemma is false, and prove that there must exist a cycle in the derivation graph. Let \mathcal{S} be the set of all variables in $V(G)$ which are entries of u . By assumption, \mathcal{S} is not empty. For each $A \in \mathcal{S}$, let $\mathcal{S}(A)$ be the set

$$\mathcal{S}(A) = \{B \in V(G) : B \text{ is an entry of some } f_G^i(A), i \geq 1\}$$

Notice that each variable in $V(G)$ appearing in at least one of the strings $f_G^i(u)$, $i \geq 1$, must lie in the union of the sets $\mathcal{S}(A)$. Since the conclusion of the lemma was assumed to be false, for each $A \in \mathcal{S}$, there exists $B \in \mathcal{S}$ such that $A \in \mathcal{S}(B)$. Pick an infinite sequence A^1, A^2, A^3, \dots from \mathcal{S} such that

$$A^i \in \mathcal{S}(A^{i+1}), \quad i \geq 1 \tag{8.17}$$

Since the set \mathcal{S} is finite, there must exist $A \in \mathcal{S}$ and positive integers $i_1 < i_2$ such that

$$A^{i_1} = A^{i_2} = A \tag{8.18}$$

Observe that if $B \in \mathcal{S}(A)$, then there is a path in the derivation graph which starts at the A vertex and ends at the B vertex. Applying this observation to the statements in

(8.17) for which $i_1 \leq i \leq i_2 - 1$, we see that there is a path in the derivation graph such that the vertices visited by the path, in order, are

$$A^{i_2}, A^{i_2-1}, \dots, A^{i_1+1}, A^{i_1}$$

Relation (8.18) tells us that this path begins and ends at A and is therefore a cycle.

Lemma 4 *Let G be a deterministic context-free grammar for which the empty string is not the right member of any production rule of G , and for which the derivation graph is acyclic. Then*

$$f_G^{|V(G)|}(u) \in T(G)^+, \quad \forall u \in (V(G) \cup T(G))^+$$

Proof. Fix $u \in (V(G) \cup T(G))^+$. We assume that

$$f_G^{|V(G)|}(u) \notin T(G)^+ \tag{8.19}$$

and show that this leads to a contradiction. The assumption (8.19) leads us to conclude that each string $f_G^i(u)$, $i = 0, 1, \dots, |V(G)|$, must have at least one entry which is a member of $V(G)$. Applying the previous lemma, there exists a sequence $A^0, A^1, \dots, A^{|V(G)|}$ of variables from $V(G)$ such that the following hold:

- (i) A^i is an entry of $f_G^i(u)$, $i = 0, 1, \dots, |V(G)|$.
- (ii) For each $i = 0, 1, \dots, |V(G)|$, the variable A^i is not an entry of any of the strings $f_G^j(u)$, $j > i$.

There are more terms in the sequence $A^0, A^1, \dots, A^{|V(G)|}$ than there are members of $V(G)$. Therefore, we may find a variable A and integers $i_1 < i_2$ from the set $\{0, 1, \dots, |V(G)|\}$ such that $A^{i_1} = A^{i_2} = A$. Because of statements (i) and (ii) above, we see that A^{i_1} , and therefore A , is an entry of $f_G^{i_1}(u)$ but not an entry of $f_G^{i_2}(u)$. From statement (i) above, we see that A^{i_2} , and therefore A , is an entry of $f_G^{i_2}(u)$. We have arrived at the desired contradiction.

Lemma 5 *Let G be a deterministic context-free grammar for which the empty string is not the right member of any production rule of G , and for which the derivation graph is*

acyclic and rooted at the S vertex. Then each symbol in $V(G) \cup T(G)$ is an entry of at least one of the strings $f_G^i(S)$, $i = 0, 1, \dots, |V(G)|$.

Proof. If $Y \in V(G) \cup T(G)$ and $A \in V(G)$, and there is a path in the derivation graph consisting of i edges which starts at the A vertex and ends at the Y vertex, then it is easy to see that Y is an entry of $f_G^i(A)$. Fix $Y \in V(G) \cup T(G)$, $Y \neq S$. The proof is complete once we show that Y is an entry of $f_G^i(S)$ for some positive integer $i \leq |V(G)|$. Since the derivation graph is rooted at S , there is a path $\{e_1, e_2, \dots, e_i\}$ which starts at the S vertex and ends at the Y vertex. For each $j = 1, \dots, i$, let $A^j \in V(G)$ be the variable such that the edge e_j starts at the A^j vertex of the derivation graph. Since the derivation graph is acyclic, the terms in the sequence A^1, A^2, \dots, A^i are distinct members of $V(G)$. Therefore, it must be that $i \leq |V(G)|$. By our observation at the beginning of the proof, we also have that Y is an entry of $f_G^i(S)$. The proof is complete.

Lemma 6 *Let G be an admissible context-free grammar. Then the derivation graph of G is rooted at the S vertex.*

Proof. The proof is by induction. Let $Y \neq S$ be a symbol in $V(G) \cup T(G)$. We must show that there is a path in the derivation graph of G which starts at the S vertex of the graph and terminates at the Y vertex of the graph. Since Y is not a useless symbol, we can find a sequence of strings $\alpha_1, \alpha_2, \dots, \alpha_k$ such that

- α_1 is the right member of the production rule whose left member is S .
- If $k > 1$, then $\alpha_i \xrightarrow{G} \alpha_{i+1}$, $i = 1, \dots, k - 1$.
- Y is an entry of α_k .

Suppose $k = 1$. In the derivation tree, there is a path consisting of one edge which starts at the S vertex and terminates at the Y vertex. Suppose $k > 1$. We may take as our induction hypothesis the property that for every symbol in α_{k-1} , there is a path in the derivation graph leading from the S vertex to the vertex labelled by that symbol. Pick an entry $A \in V(G)$ from α_{k-1} such that α_k arises when the right member of the A production rule is substituted for A in α_{k-1} . To a path leading from the S vertex to the A vertex, we may then append an edge leading from the A vertex to the Y vertex, thereby obtaining a path leading from the S vertex to the Y vertex.

Lemma 7 *Let G be an admissible context-free grammar. Then the derivation graph of G is acyclic.*

Proof. Since $L(G)$ is not empty, for some $i \geq 1$, the string $f_G^i(S)$ is a member of $L(G)$ and therefore a member of $T(G)^+$, which implies that the following property holds:

Property: All but finitely many terms of the sequence $\{f_G^i(S) : i \geq 1\}$ coincide with a string in $T(G)^+$.

Suppose A, B are variables in $V(G)$ and there is a path in the derivation graph leading from the A vertex to the B vertex. Since A is not a useless symbol, A is an entry of $f_G^i(S)$ for some i . Using the path from the A vertex to the B vertex, one then sees that B is an entry of $f_G^j(S)$ for some $j > i$. This implies that if there were a cycle in the derivation graph, some $A \in V(G)$ (the variable labelling the vertex at the beginning and the end of the cycle) would be an entry of $f_G^i(S)$ for infinitely many i . This being a contradiction of the Property, the derivation graph must be acyclic.

Proof of Theorem 2. Statement (i) of Theorem 2 implies Statement (ii) of Theorem 2 by Lemmas 6 and 7. Statement (ii) of Theorem 2 implies Statement (iii) of Theorem 2 by Lemmas 4 and 5. It is evident that Statement (iii) implies Statement (i).

9 Appendix B

In this Appendix, we prove Lemmas 1 and 2. We adopt a notation that will be helpful in these proofs: If u and v are strings in the same multiplicative monoid \mathcal{S}^* , we shall write $u \sim v$ to denote that v can be obtained by permuting the entries of the string u . (In the language of [6], $u \sim v$ means that u and v belong to the same *type class*.) We first need to establish the following lemma that is an aid in proving Lemmas 1 and 2.

Lemma 8 *Given any grammar $G \in \mathcal{G}^*(\mathcal{A})$, there exists a parsing π of the \mathcal{A} -string represented by G satisfying*

$$H(G) \leq H^*(\pi) + |G| \tag{9.20}$$

Furthermore, π is related to ω_G in the following way: There is a string $\sigma = \sigma_1\sigma_2 \cdots \sigma_t$ in $V(G) \cup T(G)$ such that $\sigma \sim \omega_G$ and

$$\pi = (f_G^\infty(\sigma_1), f_G^\infty(\sigma_2), \dots, f_G^\infty(\sigma_t)) \quad (9.21)$$

Proof. Fix $G \in \mathcal{G}^*(\mathcal{A})$. Let \mathbf{x} be the \mathcal{A} -string represented by G . Find any string σ for which there are strings α_i , $0 \leq i \leq |V(G)| - 1$, satisfying

- (i) $\alpha_0 = f_G(A_0)$ and $\alpha_{|V(G)|-1} = \sigma$.
- (ii) For each $1 \leq i \leq |V(G)| - 1$, the string α_i is obtained from the string α_{i-1} by replacing exactly one appearance of A_i in α_{i-1} with $f_G(A_i)$. (By this, we mean that there exist strings γ_1, γ_2 such that $(\gamma_1, A_i, \gamma_2)$ is a parsing of α_{i-1} and $(\gamma_1, f_G(A_i), \gamma_2)$ is a parsing of α_i .)

Letting t be the length of the string σ , write $\sigma = \sigma_1\sigma_2 \cdots \sigma_t$, where $\sigma_1, \dots, \sigma_t \in V(G) \cup T(G)$. Let π be the sequence of substrings of \mathbf{x} defined by (9.21). Studying the construction in (i)-(ii), it can be seen that $\sigma \sim \omega_G$. We complete the proof by showing that π is a parsing of \mathbf{x} satisfying (9.20). From the equation (9.21) and the fact that f_G^∞ is an endomorphism, π is a parsing of $f_G^\infty(\sigma)$. Therefore, π will be a parsing of \mathbf{x} provided we can show that

$$f_G^\infty(A_0) = f_G^\infty(\sigma) \quad (9.22)$$

From statement (ii) above, for each $1 \leq i \leq |V(G)| - 1$,

$$\begin{aligned} f_G^\infty(\alpha_{i-1}) &= f_G^\infty(\gamma_1)f_G^\infty(A_i)f_G^\infty(\gamma_2) \\ f_G^\infty(\alpha_i) &= f_G^\infty(\gamma_1)f_G^\infty(f_G(A_i))f_G^\infty(\gamma_2) \end{aligned}$$

From conclusion (iv) of Theorem 5, the two middle factors in the right members of the preceding equations are equal, from which we conclude that the left members are equal, and then (9.22) must hold. Using again the fact that $\sigma \sim \omega_G$, the unnormalized entropies of these two strings must coincide, whence

$$H(G) = H^*(\omega_G) = H^*(\sigma),$$

and (9.20) will be true provided we can show that

$$H^*(\sigma) \leq H^*(\pi) + |G| \quad (9.23)$$

Let $\sigma^{(1)}$ be the string obtained from σ by striking out all entries of σ which belong to $T(G)$. Let $\sigma^{(2)}$ be the string obtained from σ by striking out all entries of σ which belong to $V(G)$. For $i = 1, 2$, let $\pi^{(i)}$ be the subsequence of π obtained by applying f_G^∞ to the entries of $\sigma^{(i)}$. If $\sigma^{(1)}$ is the empty string or if $\sigma^{(2)}$ is the empty string, then the mapping f_G^∞ provides a one-to-one correspondence between the entries of σ and π , forcing $H^*(\sigma) = H^*(\pi)$ and the conclusion that (9.23) is true. So, we may assume that neither of the sequences $\sigma^{(1)}, \sigma^{(2)}$ is the empty string. Properties of unnormalized entropy give us

$$\begin{aligned} H^*(\sigma) &\leq H^*(\sigma^{(1)}) + H^*(\sigma^{(2)}) + |\sigma| \\ H^*(\pi) &\geq H^*(\pi^{(1)}) + H^*(\pi^{(2)}) \end{aligned} \quad (9.24)$$

We also have

$$\begin{aligned} H^*(\sigma^{(1)}) &= H^*(\pi^{(1)}) \\ H^*(\sigma^{(2)}) &= H^*(\pi^{(2)}) \end{aligned} \quad (9.25)$$

Combining (9.24) with (9.25), and using the fact that $|\sigma| \leq |G|$, we obtain (9.23), completing the proof of Lemma 8.

9.1 Proof of Lemma 1

Fix a positive integer k . Choose an arbitrary \mathcal{A} -string \mathbf{x} , an arbitrary grammar $G \in \mathcal{G}^*(\mathcal{A})$ which represents \mathbf{x} , and an arbitrary alphabet \mathcal{A} k -th order finite state source μ . We wish to establish the inequality (5.7). Let n be the length of \mathbf{x} , and we write out \mathbf{x} as $\mathbf{x} = x_1 x_2 \cdots x_n$, where each $x_i \in \mathcal{A}$. Appealing to the definition of the family of information sources $\Lambda_{fs}^k(\mathcal{A})$, we select a set \mathcal{S} of cardinality k , $s_0 \in \mathcal{S}_k$, and nonnegative real numbers $\{p(s, x|s') : s, s' \in \mathcal{S}_k, x \in \mathcal{A}\}$ such that (5.3)-(5.4) hold. We introduce the

function $\tau : \mathcal{A}^+ \rightarrow [0, 1]$ in which

$$\tau(\mathbf{y}) \triangleq \max_{s_0 \in \mathcal{S}_k} \sum_{s_1, s_2, \dots, s_m \in \mathcal{S}_k} \prod_{i=1}^m p(s_i, y_i | s_{i-1})$$

for every \mathcal{A} -string $\mathbf{y} = y_1 y_2 \cdots y_m$. We note for later use that for every \mathcal{A} -string \mathbf{y} and every parsing (v_1, v_2, \dots, v_j) of \mathbf{y} , the following relation holds:

$$\tau(\mathbf{y}) \leq \tau(v_1) \tau(v_2) \dots \tau(v_j) \quad (9.26)$$

There exists a probability distribution p^* on \mathcal{A}^+ such that for every positive integer r and every $\mathbf{y} \in \mathcal{A}^r$,

$$p^*(\mathbf{y}) = Q_k k^{-1} r^{-2} \tau(\mathbf{y}) \quad (9.27)$$

where it can be determined that Q_k is a positive constant that must satisfy $Q_k \geq 1/2$. Applying Lemma 8, let $\pi = (u_1, u_2, \dots, u_t)$ be a parsing of \mathbf{x} with $t = |G| - |V(G)| + 1$ such that (9.20) holds. We have

$$H^*(\pi) = \min_q \sum_{i=1}^t -\log q(u_i) \leq \sum_{i=1}^t -\log p^*(u_i) \quad (9.28)$$

where the minimum is over all probability distributions q on \mathcal{A}^+ . From (9.26)-(9.27),

$$\mu(\mathbf{x}) \leq \left[\prod_{i=1}^t p^*(u_i) \right] \left[\prod_{i=1}^t \{2k|u_i|^2\} \right] \quad (9.29)$$

Combining (9.20), (9.28), and (9.29), we have

$$H(G) \leq -\log \mu(\mathbf{x}) + t(1 + \log k) + |G| + 2 \sum_{i=1}^t \log |u_i| \quad (9.30)$$

We can appeal to the concavity of the logarithm function to obtain

$$\sum_{i=1}^t \log |u_i| \leq t^{-1} \log \left(\frac{\sum_{i=1}^t |u_i|}{t} \right) = t \log(n/t) \quad (9.31)$$

Combining (9.30)-(9.31), along with the fact that $t = |G| - |V(G)| + 1$, we see that (5.7) holds.

9.2 Proof of Lemma 2

The following lemma, used in the proof of Lemma 2, is easily proved by mathematical induction.

Lemma 9 *Let α be a real number satisfying $\alpha \geq 4/3$. The following statement holds for every integer $r \geq 2$:*

$$\sum_{n=2}^r n(n-1)\alpha^n \geq (r-3) \sum_{n=2}^r (n-1)\alpha^n \quad (9.32)$$

We begin our proof of Lemma 2 by fixing an \mathcal{A} -string \mathbf{x} of length at least $|\mathcal{A}|^{34}$. Let G be any irreducible grammar which represents \mathbf{x} . We have $V(G) = \{A_0, A_1, \dots, A_{|V(G)|-1}\}$, where A_0 is the start symbol of G . For each $0 \leq i \leq |V(G)| - 1$, we can express

$$f_G(A_i) = \alpha_1^i \alpha_2^i \cdots \alpha_{n_i}^i$$

where each $\alpha_j^i \in \mathcal{A} \cup V(G)$. For each $i = 0, 1, \dots, |V(G)| - 1$, let

$$\tilde{f}_G(A_i) \triangleq \begin{cases} f_G(A_i), & |f_G(A_i)| \text{ even} \\ \alpha_2^i \alpha_3^i \cdots \alpha_{n_i}^i, & |f_G(A_i)| > 1 \text{ and odd}, |f_G^\infty(\alpha_1^i)| \leq |f_G^\infty(\alpha_{n_i}^i)| \\ \alpha_1^i \alpha_2^i \cdots \alpha_{n_i-1}^i, & |f_G(A_i)| > 1 \text{ and odd}, |f_G^\infty(\alpha_{n_i}^i)| < |f_G^\infty(\alpha_1^i)| \\ \text{empty string}, & |f_G(A_i)| = 1 \end{cases}$$

Define the three strings

$$\begin{aligned} \hat{\mathbf{x}} &\triangleq f_G^\infty(\rho_G) \\ s(\mathbf{x}) &\triangleq \tilde{f}_G(A_0) \tilde{f}_G(A_1) \cdots \tilde{f}_G(A_{|V(G)|-1}) \\ \tilde{\mathbf{x}} &\triangleq f_G^\infty(s(\mathbf{x})) \end{aligned}$$

The strings $s(\mathbf{x})$ and $\tilde{\mathbf{x}}$ are not the empty string because $|f_G(A_i)| > 1$ for at least one i . Define $m(\mathbf{x})$ to be the positive integer

$$m(\mathbf{x}) \triangleq \frac{|s(\mathbf{x})|}{2}$$

We derive the following relationships which shall be useful to us in the proof of Lemma 2:

$$|\mathbf{x}| \leq |\hat{\mathbf{x}}| \leq 2|\mathbf{x}| \quad (9.33)$$

$$|G| \leq 3m(\mathbf{x}) + |\mathcal{A}| \quad (9.34)$$

$$|\tilde{\mathbf{x}}| \leq |\hat{\mathbf{x}}| \leq 2|\tilde{\mathbf{x}}| + |\mathcal{A}| \quad (9.35)$$

From the fact that $\mathbf{x} \sim f_G^\infty(\omega_G)$ (deducible from Lemma 8), and the fact that

$$\rho_G \sim A_1 A_2 \cdots A_{|V(G)|-1} \omega_G$$

we deduce that

$$\hat{\mathbf{x}} \sim f_G^\infty(A_1 A_2 \cdots A_{|V(G)|-1}) \mathbf{x}$$

and therefore

$$|\hat{\mathbf{x}}| = |f_G^\infty(A_1 A_2 \cdots A_{|V(G)|-1})| + |\mathbf{x}| \quad (9.36)$$

Since G is irreducible, each of the variables $A_1, A_2, \dots, A_{|V(G)|-1}$ appears at least once in ω_G , and therefore we must have

$$|f_G^\infty(A_1 A_2 \cdots A_{|V(G)|-1})| \leq |f_G^\infty(\omega_G)| = |\mathbf{x}| \quad (9.37)$$

From (9.36) and (9.37), we conclude that (9.33) is true.

Now we prove the relation (9.34). Since G is irreducible, if $|f_G(A)| = 1$ for a variable $A \in V(G)$, then $f_G(A) \in \mathcal{A}$, whence

$$|\{A \in V(G) : |f_G(A)| = 1\}| \leq |\mathcal{A}| \quad (9.38)$$

Using (9.38), we obtain

$$\begin{aligned}
|G| &= \sum_{A \in V(G)} |f_G(A)| \\
&= \sum_{|f_G(A)| \text{ even}} |f_G(A)| + \sum_{|f_G(A)| \text{ odd}} |f_G(A)| \\
&\leq \sum_{|f_G(A)| \text{ even}} 3 \frac{|f_G(A)|}{2} + \sum_{|f_G(A)| \text{ odd}} 3 \left(\frac{|f_G(A)| - 1}{2} \right) + |\mathcal{A}|
\end{aligned} \tag{9.39}$$

Noting that

$$m(\mathbf{x}) = \frac{\left[\sum_{|f_G(A)| \text{ even}} |f_G(A)| \right] + \left[\sum_{|f_G(A)| \text{ odd}} \{ |f_G(A)| - 1 \} \right]}{2}$$

we see that (9.34) follows from (9.39).

We now turn our attention to the proof of (9.35). By construction of the string $s(\mathbf{x})$ and (9.38), there are strings q_1 and q_2 such that

(i) $\rho_G \sim s(\mathbf{x})q_1q_2$

(ii) $q_1 \in \mathcal{A}^*$ and $|q_1| \leq |\mathcal{A}|$.

(iii) If q_2 is not the empty string, there is a one-to-one correspondence between the entries of q_2 and certain entries of $s(\mathbf{x})$ such that if Y is an entry of q_2 and Z_Y is the corresponding entry of $s(\mathbf{x})$, then $|f_G^\infty(Y)| \leq |f_G^\infty(Z_Y)|$.

If we apply the endomorphism f_G^∞ to (i), we see that

$$\hat{\mathbf{x}} \sim \tilde{\mathbf{x}}q_1f_G^\infty(q_2) \tag{9.40}$$

Because of (iii),

$$|f_G^\infty(q_2)| \leq |f_G^\infty(s(\mathbf{x}))| = |\tilde{\mathbf{x}}| \tag{9.41}$$

Applying the relations (9.40)-(9.41) together with the fact from (ii) that $|q_1| \leq |\mathcal{A}|$, we conclude

$$|\hat{\mathbf{x}}| = |\tilde{\mathbf{x}}| + |q_1| + |f_G^\infty(q_2)|$$

$$\leq 2|\tilde{\mathbf{x}}| + |\mathcal{A}| \quad (9.42)$$

from which (9.35) follows.

Having demonstrated the relations (9.33)-(9.35), we can now finish the proof of Lemma 2. Factor $s(\mathbf{x})$ as

$$s(\mathbf{x}) = w_1 w_2 \cdots w_{m(\mathbf{x})}$$

where each $w_i \in (\mathcal{A} \cup V(G))^2$. Because G is irreducible, the strings $w_1, w_2, \dots, w_{m(\mathbf{x})}$ are distinct. We express each w_i as

$$w_i = w_i^1 w_i^2$$

where $w_i^1, w_i^2 \in \mathcal{A} \cup V(G)$. Let $\alpha \in \mathcal{A}^+$ be arbitrary. We need to upper bound the cardinality of the set

$$W_\alpha = \{1 \leq i \leq m(\mathbf{x}) : f_G^\infty(w_i) = \alpha\}$$

To this end, let ϕ be the mapping from the set W_α into the set $\{1, 2, \dots, |\alpha|-1\} \times \{0, 1\} \times \{0, 1\}$ in which $i \in W_\alpha$ is mapped into

$$\phi(i) = (|f_G^\infty(w_i^1)|, b_1, b_2)$$

where for each $q = 1, 2$,

$$b_q \triangleq \begin{cases} 1, & w_i^q \in \mathcal{A} \\ 0, & \text{otherwise} \end{cases}$$

Since the mapping ϕ is one-to-one, we must have

$$|W_\alpha| \leq 4(|\alpha|-1)$$

We conclude that

$$|\{1 \leq i \leq m(\mathbf{x}) : |f_G^\infty(w_i)| = n\}| \leq 4(n-1)|\mathcal{A}|^n, \quad n \geq 2 \quad (9.43)$$

Define $\{j_n : n \geq 2\}$ and $\{k_n : n \geq 2\}$ to be the sequences of integers

$$j_n \triangleq |\{1 \leq i \leq m(\mathbf{x}) : |f_G^\infty(w_i)| = n\}|$$

$$k_n \triangleq 4(n-1)|\mathcal{A}|^n$$

Define $\{M_r : r \geq 2\}$ and $\{N_r : r \geq 2\}$ to be the sequences of positive integers

$$\begin{aligned} M_r &\triangleq \sum_{n=2}^r k_n \\ N_r &\triangleq \sum_{n=2}^r nk_n \end{aligned}$$

Notice that

$$|\mathbf{x}| \geq |\mathcal{A}|^{32} > |\mathcal{A}|^5|\mathcal{A}|^2 > 17|\mathcal{A}|^2 > 16|\mathcal{A}|^2 + |\mathcal{A}|$$

This fact implies, via (9.35), that

$$|\tilde{\mathbf{x}}| \geq 8|\mathcal{A}|^2 = N_2$$

and so we may define an integer $r(\mathbf{x}) \geq 2$ as follows:

$$r(\mathbf{x}) \triangleq \max\{r : N_r \leq |\tilde{\mathbf{x}}|\}$$

We establish a lower bound on $r(\mathbf{x})$. Notice that

$$|\tilde{\mathbf{x}}| \geq N_{r(\mathbf{x})} \geq |\mathcal{A}|^{r(\mathbf{x})} \geq 2^{r(\mathbf{x})}$$

from which it follows that

$$r(\mathbf{x}) \leq \log |\tilde{\mathbf{x}}|$$

On the other hand,

$$|\tilde{\mathbf{x}}| < N_{r(\mathbf{x})+1} \leq 4(r(\mathbf{x})+1)r(\mathbf{x})^2|\mathcal{A}|^{r(\mathbf{x})+1} \leq 8r(\mathbf{x})^3|\mathcal{A}|^{r(\mathbf{x})+1}$$

and so

$$\log |\tilde{\mathbf{x}}| \leq 3 + 3 \log r(\mathbf{x}) + (r(\mathbf{x})+1) \log |\mathcal{A}| \leq 3 + 3 \log \log |\tilde{\mathbf{x}}| + (r(\mathbf{x})+1) \log |\mathcal{A}|$$

from which we conclude that

$$r(\mathbf{x}) - 3 \geq \frac{\log |\tilde{\mathbf{x}}| - 3 \log \log |\tilde{\mathbf{x}}| - 4 \log |\mathcal{A}| - 3}{\log |\mathcal{A}|} \quad (9.44)$$

We examine the right side of (9.44) in more detail. It is a simple exercise in calculus to show that

$$\log u - 3 \log \log u \geq \frac{\log u}{2}, \quad u \geq 2^{32} \quad (9.45)$$

Notice that

$$|\mathbf{x}| \geq |\mathcal{A}|^{34} > |\mathcal{A}|^{33} + |\mathcal{A}| \geq 2^{33} + |\mathcal{A}|$$

and therefore

$$|\tilde{\mathbf{x}}| \geq 2^{32}$$

Combining this fact with (9.45), we see that

$$\log |\tilde{\mathbf{x}}| - 3 \log \log |\tilde{\mathbf{x}}| - 4 \log |\mathcal{A}| - 3 \geq \frac{\log |\tilde{\mathbf{x}}|}{2} - 4 \log |\mathcal{A}| - 3 \quad (9.46)$$

From the fact that

$$|\mathbf{x}| \geq |\mathcal{A}|^{34} > 2|\mathcal{A}|$$

it follows that

$$|\tilde{\mathbf{x}}| \geq \frac{|\mathbf{x}| - |\mathcal{A}|}{2} \geq \frac{|\mathbf{x}|}{4}$$

and therefore

$$\begin{aligned} \frac{\log |\tilde{\mathbf{x}}|}{2} - 4 \log |\mathcal{A}| - 3 &\geq \frac{\log |\mathbf{x}|}{2} - 4 \log |\mathcal{A}| - 4 \\ &\geq 13 \log |\mathcal{A}| - 4 > 0 \end{aligned} \quad (9.47)$$

Applying (9.47) to (9.44), we conclude that

$$r(\mathbf{x}) - 3 \geq \frac{\log |\mathbf{x}| - 8 \log |\mathcal{A}| - 8}{2 \log |\mathcal{A}|} > 0 \quad (9.48)$$

From the definition of $r(\mathbf{x})$, we have

$$|\tilde{\mathbf{x}}| = N_{r(\mathbf{x})} + \Delta = \sum_{n=2}^{\infty} n j_n = \left(\sum_{n=2}^{r(\mathbf{x})} n k_n \right) + \Delta$$

where $\Delta \geq 0$. From this we argue:

$$\begin{aligned} \sum_{n=r(\mathbf{x})+1}^{\infty} n j_n &= \left[\sum_{n=2}^{r(\mathbf{x})} n (k_n - j_n) \right] + \Delta \\ \sum_{r(\mathbf{x})+1}^{\infty} j_n &\leq \left[\sum_{n=2}^{r(\mathbf{x})} \left(\frac{n}{r(\mathbf{x})+1} \right) (k_n - j_n) \right] + \frac{\Delta}{r(\mathbf{x})+1} \end{aligned}$$

The preceding allows us to conclude that

$$m(\mathbf{x}) = \sum_{n=2}^{\infty} j_n \leq \left[\sum_{n=2}^{r(\mathbf{x})} \left(1 - \frac{n}{r(\mathbf{x})+1} \right) j_n \right] + \left[\sum_{n=2}^{r(\mathbf{x})} \left(\frac{n}{r(\mathbf{x})+1} \right) k_n \right] + \frac{\Delta}{r(\mathbf{x})+1} \quad (9.49)$$

Since $j_n \leq k_n$ (see (9.43)), we may replace j_n with k_n in the first term on the right in (9.49) to obtain the bounds

$$m(\mathbf{x}) \leq M_{r(\mathbf{x})} + \frac{\Delta}{r(\mathbf{x})} \leq \frac{N_{r(\mathbf{x})} + \Delta}{r(\mathbf{x}) - 3} = \frac{|\tilde{\mathbf{x}}|}{r(\mathbf{x}) - 3} \quad (9.50)$$

where in the preceding we also used Lemma 9. Applying to (9.50) the lower bound on $r(\mathbf{x}) - 3$ that was established in (9.48), we obtain

$$\frac{m(\mathbf{x})}{|\tilde{\mathbf{x}}|} \leq \frac{2 \log |\mathcal{A}|}{\log |\mathbf{x}| - 8 \log |\mathcal{A}| - 8} \quad (9.51)$$

From the relationships (9.33)-(9.35), one can see that

$$\frac{|G| - |\mathcal{A}|}{6|\mathbf{x}|} \leq \frac{m(\mathbf{x})}{2|\mathbf{x}|} \leq \frac{m(\mathbf{x})}{|\hat{\mathbf{x}}|} \leq \frac{m(\mathbf{x})}{|\tilde{\mathbf{x}}|} \quad (9.52)$$

Combining (9.51) and (9.52), we obtain (5.8), the desired conclusion of Lemma 2.

References

- [1] A. Aho, R. Sethi, and J. Ullman, *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, Reading, MA, 1986.
- [2] R. Cameron, “Source Encoding Using Syntactic Information Source Models,” *IEEE Trans. Inform. Theory*, vol. 34, pp. 843–850, 1988.
- [3] C. Cook, A. Rosenfeld, and A. Aronson, “Grammatical Inference by Hill Climbing,” *Informational Sciences*, vol. 10, pp. 59–80, 1976.
- [4] T. Cover, “Enumerative Source Encoding,” *IEEE Trans. Inform. Theory*, vol. 19, pp. 73–77, 1973.
- [5] T. Cover and J. Thomas, *Elements of Information Theory*. John Wiley & Sons, Inc., New York, 1991.
- [6] I. Csiszár and J. Körner, *Information Theory: Coding Theorems for Discrete Memoryless Systems*. Academic Press, New York, 1981.
- [7] L. Davisson, “Universal Noiseless Coding,” *IEEE Trans. Inform. Theory*, vol. 19, pp. 783–795, 1973.
- [8] J. Deller, J. Proakis, and J. Hansen, *Discrete-Time Processing of Speech Signals*. Macmillan Publishing Co., Englewood Cliffs, NJ, 1993.
- [9] C. Hobby and N. Ylvisaker, “Some Structure Theorems for Stationary Probability Measures on Finite State Sequences,” *Ann. Math. Stat.*, vol. 35, pp. 550–556, 1964.
- [10] J. Hopcroft and J. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA, 1979.
- [11] E. Kawaguchi and T. Endo, “On a Method of Binary-Picture Representation and Its Application to Data Compression,” *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 2, pp. 27–35, 1980.

- [12] J. C. Kieffer and E.-H. Yang, “Sequential Codes, Lossless compression of Individual Sequences, and Kolmogorov Complexity,” *IEEE Trans. Inform. Theory*, Vol. 42, pp. 29–39, 1996.
- [13] J. Kieffer, E-h. Yang, G. Nelson, and P. Cosman, “Universal Lossless Compression via Multilevel Pattern Matching,” *IEEE Trans. Inform. Theory*, under review.
- [14] E. Kourapova and B. Ryabko, “Application of Formal Grammars for Encoding Information Sources,” *Problems of Information Transmission*, vol. 31, pp. 23–26, 1995.
- [15] A. Lempel and J. Ziv, “On the Complexity of Finite Sequences,” *IEEE Trans. Inform. Theory*, vol. 22, pp. 75–81, 1976.
- [16] M. Li and P. Vitányi, *An Introduction to Kolmogorov Complexity and its Applications*. Springer-Verlag, New York, 1993.
- [17] A. Lindenmayer, “Mathematical Models for Cellular Interaction in Development,” *Jour. of Theoretical Biology*, vol. 18, pp. 280–315, 1968.
- [18] C. Nevill-Manning and I. Witten, “Identifying Hierarchical Structure in Sequences: A Linear-Time Algorithm,” *Jour. Artificial Intell. Res.*, vol. 7, pp. 67–82, 1997.
- [19] C. Nevill-Manning and I. Witten, “Compression and Explanation Using Hierarchical Grammars,” *Computer Journal*, vol. 40, pp. 103–116, 1997.
- [20] 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*, vol. 38, pp. 66–72, 1992.
- [21] G. Rozenberg and A. Salomaa. *The Mathematical Theory of L Systems*. Academic Press, New York, 1980.
- [22] R. Schalkoff, *Digital Image Processing and Computer Vision*. John Wiley & Sons, New York, 1989.
- [23] Y. Shtarkov, “Fuzzy Estimation of Unknown Source Model for Universal Coding,” *Proc. 1998 IEEE Information Theory Workshop (Killarney, Ireland)*, pp. 17–18.

- [24] J. Storer and T. Szymanski, “Data Compression via Textual Substitution,” *Jour. Assoc. Comput. Mach.*, vol. 29, pp. 928–951, 1982.
- [25] E-h. Yang and J. Kieffer, “Efficient Universal Lossless Data Compression Algorithms Based on a Greedy Sequential Grammar Transform—Part One: Without Context Models,” *IEEE Trans. Inform. Theory*, Jan. 2000, to appear.
- [26] J. Ziv, “Coding Theorems for Individual Sequences,” *IEEE Trans. Inform. Theory*, vol. 24, pp. 405–412, 1978.
- [27] J. Ziv and A. Lempel, “Compression of Individual Sequences via Variable-rate Coding,” *IEEE Trans. Inform. Theory*, vol. 24, pp. 530–536, 1978.

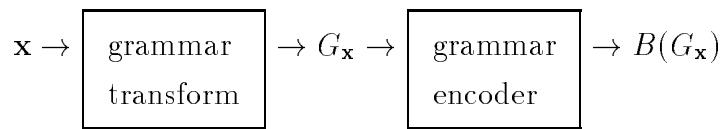


Figure 1: Encoder Structure of Grammar Based Code