# Efficient Universal Lossless Data Compression Algorithms Based on a Greedy Sequential Grammar Transform—Part One: Without Context Models

En-hui Yang, *Member, IEEE,* and John C. Kieffer, *Fellow, IEEE*

***Abstract*—A grammar transform is a transformation that converts any data sequence to be compressed into a grammar from which the original data sequence can be fully reconstructed. In a grammar-based code, a data sequence is first converted into a grammar by a grammar transform and then losslessly encoded. In this paper, a greedy grammar transform is first presented; this grammar transform constructs sequentially a sequence of irreducible grammars from which the original data sequence can be recovered incrementally. Based on this grammar transform, three universal lossless data compression algorithms, a sequential algorithm, an improved sequential algorithm, and a hierarchical algorithm, are then developed. These algorithms combine the power of arithmetic coding with that of string matching. It is shown that these algorithms are all universal in the sense that they can achieve asymptotically the entropy rate of any stationary, ergodic source. Moreover, it is proved that their worst case redundancies among all individual sequences of length $n$ are upper-bounded by $c \log \log n / \log n$, where $c$ is a constant. Simulation results show that the proposed algorithms outperform the Unix Compress and Gzip algorithms, which are based on LZ78 and LZ77, respectively.**

***Index Terms*—Arithmetic coding, entropy, grammar-based source codes, redundancy, string matching, universal sequential and hierarchical data compression.**

## I. Introduction

UNIVERSAL data compression theory aims at designing data compression algorithms, whose performance is asymptotically optimal for a class of sources. The field of universal data compression theory can be divided into two subfields: universal lossless data compression and universal lossy data compression. In this paper, we are concerned with universal lossless data compression. Our goal is to develop new practical lossless data compression algorithms which are asymptotically optimal for a broad class of sources, including stationary, ergodic sources.

Manuscript received December 30, 1998; revised July 7, 1999. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada under Grant RGPIN203035-98, by the Communications and Information Technology Ontario, and by the National Sciences Foundation under Grant NCR-9627965.

E.-h. Yang is with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, Ont., Canada N2L 3G1 (e-mail: ehyang@bbcr.uwaterloo.ca).

J. C. Kieffer is with the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN 55455 USA (e-mail: kieffer@ece.umn.edu).

Communicated by N. Merhav, Associate Editor for Source Coding.

Publisher Item Identifier S 0018-9448(00)00067-5.

To put things into perspective, let us first review briefly, from the information-theoretic point of view, the existing universal lossless data compression algorithms. So far, the most widely used universal lossless compression algorithms are arithmetic coding algorithms [1], [20], [22], [23], [29], Lempel–Ziv algorithms [16], [35], [36], and their variants. Arithmetic coding algorithms and their variants are statistical model-based algorithms. To use an arithmetic coding algorithm to encode a data sequence, a statistical model is either built dynamically during the encoding process, or assumed to exist in advance. Several approaches have been proposed in the literature to build dynamically a statistical model. These include the prediction by partial match algorithm [4], dynamic Markov modeling [5], context gathering algorithm [24], [26], and context-tree weighting method [27], [28]. Typically, in all these methods, the next symbol in the data sequence is predicted by a proper context and coded by the corresponding estimated conditional probability. Good compression can be achieved if a good tradeoff between the number of contexts and the conditional entropy of the next symbols given contexts is maintained during the encoding process. Arithmetic coding algorithms and their variants are universal only with respect to the class of Markov sources with Markov order less than some designed parameter value. Note that in arithmetic coding, the original data sequence is encoded letter by letter. In contrast, no statistical model is used in Lempel–Ziv algorithms and their variants. During the encoding process, the original data sequence is parsed into nonoverlapping, variable-length phrases according to some kind of string matching mechanism, and then encoded phrase by phrase. Each parsed phrase is either distinct or replicated with the number of repetitions less than or equal to the size of the source alphabet. Phrases are encoded in terms of their positions in a dictionary or database. Lempel–Ziv algorithms are universal with respect to a class of sources which is broader than the class of Markov sources of bounded order; the incremental parsing Lempel–Ziv algorithm [36] is universal for the class of stationary, ergodic sources.

Other universal compression algorithms include the dynamic Huffman algorithm [10], the move to front coding scheme [3], [9], [25], and some two-stage compression algorithms with codebook transmission [17], [19]. These algorithms are either inferior to arithmetic coding algorithms and Lempel–Ziv algorithms, or too complicated to implement.

Very recently, a new type of lossless source code called a grammar-based code was proposed in [12]. The class of grammar-based codes is broad enough to include block

0018–9448/00$10.00 © 2000 IEEE

codes, Lempel–Ziv types of codes, multilevel pattern matching (MPM) grammar-based codes [13], and other codes as special cases. To compress a data sequence, each grammar-based code first transforms the data sequence into a context-free grammar, from which the original data sequence can be fully reconstructed by performing parallel substitutions, and then uses an arithmetic coding algorithm to compress the context-free grammar. It has been proved in [12] that if a grammar-based code transforms each data sequence into an irreducible context-free grammar, then the grammar-based code is universal for the class of stationary, ergodic sources. (For the definition of grammar-based codes and irreducible context free grammars, please see Section II.) Each irreducible grammar also gives rise to a nonoverlapping, variable-length parsing of the data sequence it represents. Unlike the parsing in Lempel–Ziv algorithms, however, there is no upper bound on the number of repetitions of each parsed phrase. More repetitions of each parsed phrase imply that now there is room for arithmetic coding, which operates on phrases instead of letters, to kick in. (In Lempel–Ziv algorithms, there is not much gain from applying arithmetic coding to parsed phrases since each parsed phrase is either distinct or replicated with the number of repetitions less than or equal to the size of the source alphabet.) The framework of grammar-based codes suggests that one should try to optimize arithmetic coding and string matching capability by properly designing grammar transforms. We address this optimization problem in this paper.

Within the design framework of grammar-based codes, we first present in this paper an efficient greedy grammar transform that constructs sequentially a sequence of irreducible context-free grammars from which the original data sequence can be recovered incrementally. Based on this greedy grammar transform, we then develop three universal lossless data compression algorithms: a sequential algorithm, an improved sequential algorithm, and a hierarchical algorithm. These algorithms combine the power of arithmetic coding with that of string matching in a very elegant way and jointly optimize in some sense string matching and arithmetic coding capability. It is shown that these algorithms are universal in the sense that they can achieve asymptotically the entropy rate of any stationary, ergodic source. Moreover, it is proved that their worst case redundancies among all individual sequences of length $n$ are upper-bounded by $c \log\log n / \log n$, where $c$ is a constant. These algorithms have essentially linear computation and storage complexity. Simulation results show that these algorithms outperform the Unix Compress and Gzip algorithms, which are based on LZ78 and LZ77, respectively.

The paper is organized as follows. In Section II, we briefly review grammar-based codes. In Section III, we present our greedy grammar transform and discuss its properties. Section IV is devoted to the description of the sequential algorithm, improved sequential algorithm, and hierarchical algorithm. In Sections V and VI, we analyze the performance of the hierarchical algorithm and that of the sequential and improved sequential algorithms, respectively. Finally, we show some simulation results in Section VII and draw some conclusions in Section VIII.

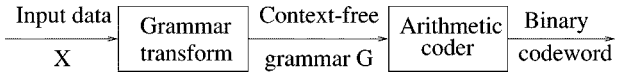


Fig. 1. Structure of a grammar-based code.

## II. Review of Grammar-Based Codes

The purpose of this section is to briefly review grammar-based codes so that this paper is self-contained and to provide some additional insights into grammar-based codes. For the detailed description of grammar-based codes, please refer to [12].

Let $\mathcal{A}$ be our source alphabet with cardinality greater than or equal to 2. Let $\mathcal{A}^*$ be the set of all finite strings drawn from $\mathcal{A}$, including the empty string $\lambda$, and $\mathcal{A}^+$ the set of all finite strings of positive length from $\mathcal{A}$. The notation $|\mathcal{A}|$ stands for the cardinality of $\mathcal{A}$, and for any $x \in \mathcal{A}^*$, $|x|$ denotes the length of $x$. For any positive integer $n$, $\mathcal{A}^n$ denotes the set of all sequences of length $n$ from $\mathcal{A}$. Similar notation will be applied to other finite sets and finite strings drawn from them. To avoid possible confusion, a sequence from $\mathcal{A}$ is sometimes called an $\mathcal{A}$-sequence. Let $x \in \mathcal{A}^+$ be a sequence to be compressed. As shown in Fig. 1, in a grammar-based code, the sequence $x$ is first transformed into a context-free grammar (or simply a grammar) $G$ from which $x$ can be fully recovered, and then compressed indirectly by using a zero-order arithmetic code[1] to compress $G$. To get an appropriate $G$, string matching is often used in some manner. It is clear that to describe grammar-based codes, it suffices to specify grammar transforms. We begin with explaining how context-free grammars are used to represent sequences $x$ in $\mathcal{A}^+$.

### A. Context-Free Grammars

Fix a countable set $\mathcal{S} = \{s_0, s_1, s_2, \cdots\}$ of symbols, disjoint from $\mathcal{A}$. Symbols in $\mathcal{S}$ will be called *variables*; symbols in $\mathcal{A}$ will be called *terminal symbols*. For any $j \geq 1$, let $\mathcal{S}(j) = \{s_0, s_1, \cdots, s_{j-1}\}$. For our purpose, a context-free grammar $G$ is a mapping from $\mathcal{S}(j)$ to $(\mathcal{S}(j) \cup \mathcal{A})^+$ for some $j \geq 1$. The set $\mathcal{S}(j)$ shall be called the variable set of $G$ and, to be specific, the elements of $\mathcal{S}(j)$ shall be called sometimes $G$-variables. To describe the mapping explicitly, we shall write, for each $s_i (i < j)$, the relationship $(s_i, G(s_i))$ as $s_i \to G(s_i)$, and call it a production rule. Thus the grammar $G$ is completely described by the set of production rules[2] $\{s_i \to G(s_i): 0 \leq i < j\}$. Start with the variable $s_0$. Replacing in parallel each variable $s$ in $G(s_0)$ by $G(s)$, we get another sequence from $\mathcal{S}(j) \cup \mathcal{A}$. If we keep doing this parallel replacement procedure, one of the following will hold:

1) After finitely many parallel replacement steps, we obtain a sequence from $\mathcal{A}$.
2) The parallel replacement procedure never ends because each string so obtained contains an entry which is a $G$-variable.

For the purpose of data compression, we are interested only in grammars $G$ for which the parallel replacement procedure ter-

[1]This term is an abbreviation for "an arithmetic code with a zero-order statistical model."

[2]There are many other ways to describe $G$. For example, $G$ is described by a substitution table in [14] and by a directed graph in [15].

minates after finitely many steps and every $G$-variable $s_i (i < j)$ is replaced at least once by $G(s_i)$ in the whole parallel replacement process. Such grammars $G$ are called *admissible grammars* and the unique sequence from $\mathcal{A}$ resulting from the parallel replacement procedure is called a sequence represented by $G$ or by $s_0$. Since each variable $s_i$ is replaced at least once by $G(s_i)$, it is easy to see that each variable $s_i (i \neq 0)$ represents a substring of the $\mathcal{A}$-sequence represented by $s_0$, as shown in the following example.

*Example 1:* Let $\mathcal{A} = \{0, 1\}$. Below is an example of an admissible grammar $G$ with variable set $\{s_0, s_1, s_2, s_3\}$

$$\begin{aligned} s_0 &\rightarrow 0 s_3 s_2 s_1 s_1 s_3 10 \\ s_1 &\rightarrow 01 \\ s_2 &\rightarrow s_1 1 \\ s_3 &\rightarrow s_1 s_2. \end{aligned}$$

Perform the following parallel replacements:[3]

$$\begin{aligned} s_0 &\xrightarrow{G} 0 s_3 s_2 s_1 s_1 s_3 10 \\ 0 s_3 s_2 s_1 s_1 s_3 10 &\xrightarrow{G} 0 s_1 s_2 s_1 10101 s_1 s_2 10 \\ 0 s_1 s_2 s_1 10101 s_1 s_2 10 &\xrightarrow{G} 001 s_1 1011010101 s_1 110 \\ 001 s_1 1011010101 s_1 110 &\xrightarrow{G} 00101101101010101110. \end{aligned}$$

In the above, we start with $s_0$ and then repeatedly apply the parallel replacement procedure. We see that after four steps—each appearance of the notation $\xrightarrow{G}$ represents one step of parallel replacements—we get a sequence from $\mathcal{A}$ and the parallel replacement procedure terminates. Also, each variable $s_i (0 \leq i < 4)$ is replaced at least once by $G(s_i)$ in the whole parallel replacement process. Therefore, in this example, $s_0$ (or $G$) represents the sequence $x = 00101101101010101110$. Each of the other $G$-variables represents a substring of $x$: $s_1$ represents 01, $s_2$ represents 011, and $s_3$ represents 01011.

Let $G$ be an admissible grammar with variable set $\mathcal{S}(j)$. The size $|G|$ of $G$ is defined as the sum of the length $|G(s)|$ over $\mathcal{S}(j)$

$$|G| \triangleq \sum_{s \in \mathcal{S}(j)} |G(s)| \tag{2.1}$$

where $|G(s)|$ denotes the length of the $\mathcal{S}(j) \cup \mathcal{A}$-sequence $G(s)$. For example, the size of the admissible grammar $G$ in Example 1 is equal to 14. Given any sequence $x$ from $\mathcal{A}$, if the length $|x|$ of $x$ is large, then there are many admissible grammars $G$ that represent $x$. Some of these grammars will be more compact than others in the sense of having smaller size $|G|$. Since in a grammar-based code, the sequence $x$ is compressed indirectly by using a zero-order arithmetic code to compress an admissible grammar $G$ that represents $x$, the size of $G$ is quite influential in the performance of the grammar-based code. In principle, an admissible grammar $G$ that represents $x$ should be designed so that the following properties hold:

a.1) The size $|G|$ of $G$ should be small enough, compared to the length of $x$.

a.2) $\mathcal{A}$-strings represented by distinct variables of $G$ are distinct.

a.3) The frequency distribution of variables and terminal symbols of $G$ in the range of $G$ should be such that effective arithmetic coding can be accomplished later on.

Starting with an admissible grammar $G$ that represents $x$, one can apply repeatedly a set of reduction rules to get another admissible grammar $G'$ which represents the same $x$ and satisfies Properties a.1)–a.3) in some sense. This set of reduction rules is introduced in [12] and will be described next.

### B. Reduction Rules

*Reduction Rule 1:* Let $s$ be a variable of an admissible grammar $G$ that appears only once in the range of $G$. Let $s' \rightarrow \alpha s \beta$ be the unique production rule in which $s$ appears on the right. Let $s \rightarrow \gamma$ be the production rule corresponding to $s$. Reduce $G$ to the admissible grammar $G'$ obtained by removing the production rule $s \rightarrow \gamma$ from $G$ and replacing the production rule $s' \rightarrow \alpha s \beta$ with the production rule $s' \rightarrow \alpha \gamma \beta$. The resulting admissible grammar $G'$ represents the same sequence $x$ as does $G$.

*Example 2:* Consider the grammar $G$ with variable set $\{s_0, s_1, s_2\}$ given by

$$\{s_0 \rightarrow s_1 s_1,\ s_1 \rightarrow s_2 1,\ s_2 \rightarrow 010\}.$$

Applying Reduction Rule 1, one gets the grammar $G'$ with variable set $\{s_0, s_1\}$ given by

$$\{s_0 \rightarrow s_1 s_1,\ s_1 \rightarrow 0101\}.$$

*Reduction Rule 2:* Let $G$ be an admissible grammar possessing a production rule of form $s \rightarrow \alpha_1 \beta \alpha_2 \beta \alpha_3$, where the length of $\beta$ is at least 2. Let $s' \in \mathcal{S}$ be a variable which is not a $G$-variable. Reduce $G$ to the grammar $G'$ obtained by replacing the production rule $s \rightarrow \alpha_1 \beta \alpha_2 \beta \alpha_3$ of $G$ with $s \rightarrow \alpha_1 s' \alpha_2 s' \alpha_3$, and by appending the production rule $s' \rightarrow \beta$. The resulting grammar $G'$ includes a new variable $s'$ and represents the same sequence $x$ as does $G$.

*Example 3:* Consider the grammar $G$ with variable set $\{s_0, s_1\}$ given by

$$\{s_0 \rightarrow s_1 01 s_1 01,\ s_1 \rightarrow 11\}.$$

Applying Reduction Rule 2, one gets the grammar $G'$ with variable set $\{s_0, s_1, s_2\}$ given by

$$\{s_0 \rightarrow s_1 s_2 s_1 s_2,\ s_1 \rightarrow 11,\ s_2 \rightarrow 01\}.$$

*Reduction Rule 3:* Let $G$ be an admissible grammar possessing two distinct production rules of form $s \rightarrow \alpha_1 \beta \alpha_2$ and $s' \rightarrow \alpha_3 \beta \alpha_4$, where $\beta$ is of length at least two, either $\alpha_1$ or $\alpha_2$ is not empty, and either $\alpha_3$ or $\alpha_4$ is not empty. Let $s'' \in \mathcal{S}$ be a variable which is not a $G$-variable. Reduce $G$ to the grammar $G'$ obtained by doing the following: replace rule $s \rightarrow \alpha_1 \beta \alpha_2$ by $s \rightarrow \alpha_1 s'' \alpha_2$, replace rule $s' \rightarrow \alpha_3 \beta \alpha_4$ by $s' \rightarrow \alpha_3 s'' \alpha_4$, and append the new rule $s'' \rightarrow \beta$.

[3]One can also perform serial replacements. However, the parallel replacement procedure makes things look simple.

*Example 4:* Consider the grammar $G$ with variable set $\{s_0, s_1, s_2\}$ given by

$$\{s_0 \to s_1 0 s_2,\ s_1 \to 10,\ s_2 \to 0 s_1 0\}.$$

Applying Reduction Rule 3, one gets the grammar $G'$ with variable set $\{s_0, s_1, s_2, s_3\}$ given by

$$\{s_0 \to s_3 s_2,\ s_1 \to 10,\ s_2 \to 0 s_3,\ s_3 \to s_1 0\}.$$

*Reduction Rule 4:* Let $G$ be an admissible grammar possessing two distinct production rules of the form $s \to \alpha_1 \beta \alpha_2$ and $s' \to \beta$, where $\beta$ is of length at least two, and either $\alpha_1$ or $\alpha_2$ is not empty. Reduce $G$ to the grammar $G'$ obtained by replacing the production rule $s \to \alpha_1 \beta \alpha_2$ with the production rule $s \to \alpha_1 s' \alpha_2$.

*Example 5:* Consider the grammar $G$ with variable set $\{s_0, s_1, s_2\}$ given by

$$\{s_0 \to s_2 01 s_1,\ s_1 \to s_2 0,\ s_2 \to 11\}.$$

Applying Reduction Rule 4, one gets the grammar $G'$ with variable set $\{s_0, s_1, s_2\}$ given by

$$\{s_0 \to s_1 1 s_1,\ s_1 \to s_2 0,\ s_2 \to 11\}.$$

*Reduction Rule 5:* Let $G$ be an admissible grammar in which two variables $s$ and $s'$ represent the same substring of the $\mathcal{A}$-sequence represented by $G$. Reduce $G$ to the grammar $G'$ obtained by replacing each appearance of $s'$ in the range of $G$ by $s$ and deleting the production rule corresponding to $s'$. The grammar $G'$ may not be admissible since some $G'$-variables may not be involved in the whole parallel replacement process of $G'$. If so, further reduce $G'$ to the admissible grammar $G''$ obtained by deleting all production rules corresponding to variables of $G'$ that are not involved in the whole parallel replacement process of $G'$. Both $G$ and $G''$ represent the same sequence from $\mathcal{A}$.

An admissible grammar $G$ is said to be *irreducible* if none of Reduction Rules 1–5 can be applied to $G$ to get a new admissible grammar. The admissible grammar shown in Example 1 is irreducible. Clearly, an irreducible grammar $G$ satisfies the following properties:

- b.1) Each $G$-variable $s$ other than $s_0$ appears at least twice in the range of $G$.
- b.2) There is no nonoverlapping repeated pattern of length greater than or equal to 2 in the range of $G$.
- b.3) Each distinct $G$-variable represents a distinct $\mathcal{A}$-sequence.

Property b.3) is due to Reduction Rule 5 and very important to the compression performance of a grammar-based code. A grammar-based code for which the transformed grammar does not satisfy Property b.3), may give poor compression performance and cannot be guaranteed to be universal. The reason for this is that once different variables of a grammar represent the same $\mathcal{A}$-sequence, the empirical entropy of the grammar gets expanded. Since the compression performance of the corresponding grammar-based code is related to the empirical entropy of the grammar, the entropy expansion translates into poor compression performance.

An irreducible grammar satisfies Properties a.1)–a.3) in some sense, as shown in the next subsection.

### C. Grammar Transforms

Let $x$ be a sequence from $\mathcal{A}$ which is to be compressed. A grammar transform converts $x$ into an admissible grammar that represents $x$. In this paper, we are interested particularly in a grammar transform that starts from the grammar $G$ consisting of only one production rule $s_0 \to x$, and applies repeatedly Reduction Rules 1–5 in some order to reduce $G$ into an irreducible grammar $G'$. Such a grammar transform is called an *irreducible* grammar transform. To compress $x$, the corresponding grammar-based code then uses a zero-order arithmetic code to compress the irreducible grammar $G'$. After receiving the codeword of $G'$, one can fully recover $G'$ from which $x$ can be obtained via parallel replacement. Different orders via which the reduction rules are applied give rise to different irreducible grammar transforms, resulting in different grammar-based codes. No matter how the reduction rules are applied, all these grammar-based codes are universal, as guaranteed by the following results, which were proved in [12].

*Result 1:* Let $G$ be an irreducible grammar representing a sequence $x$ from $\mathcal{A}$. The size $|G|$ of $G$ divided by the length $|x|$ of $x$ goes to 0 uniformly as $|x|$ increases. Specifically

$$\max\{|G|\colon G \text{ is an irreducible grammar representing } x, x \in \mathcal{A}^n\} = O(n/\log n). \quad (2.2)$$

*Result 2:* Any grammar-based code with an irreducible grammar transform is universal in the sense that for any stationary, ergodic source $\{X_i\}_{i=1}^{\infty}$, the compression rate resulting from using the grammar-based code to compress the initial segment $X_1 X_2 \cdots X_n$ of length $n$ converges, with probability one, to the entropy rate of the source as $n$ goes to infinity.

Clearly, Reduction Rules 2–4 are string matching reduction rules. The reason that grammar-based codes with irreducible grammar transforms are universal lies in the fact that such codes combine the power of string matching with that of arithmetic coding. The above results, however, do not say how to construct explicitly such codes or irreducible grammar transforms although there are many of them to choose from. Also, within the framework of grammar-based codes, it needs to be determined how one can design irreducible grammar transforms that can in some sense jointly optimize arithmetic coding and string matching capability.

In this paper, we address the concerns raised in the preceding paragraph. In the next section, we shall present a greedy grammar transform that can construct sequentially a sequence of irreducible grammars from which the original data sequence can be recovered incrementally. This greedy grammar transform then enables us to develop three universal lossless data compression algorithms.

## III. A Greedy Grammar Transform

As mentioned at the end of the last section, the purpose of this section is to describe our greedy irreducible grammar transform.

*The Proposed Irreducible Grammar Transform:* Let $x = x_1 x_2 \cdots x_n$ be a sequence from $\mathcal{A}$ which is to be com-

pressed. The proposed irreducible grammar transform is a greedy one. It parses the sequence $x$ sequentially into nonoverlapping substrings $\{x_1, x_2\cdots x_{n_2}, \cdots, x_{n_{t-1}+1}\cdots x_{n_t}\}$ and builds sequentially an irreducible grammar for each $x_1\cdots x_{n_i}$, where $1 \le i \le t$, $n_1 = 1$, and $n_t = n$. The first substring is $x_1$ and the corresponding irreducible grammar $G_1$ consists of only one production rule $s_0 \to x_1$. Suppose that $x_1, x_2\cdots x_{n_2}, \cdots, x_{n_{i-1}+1}\cdots x_{n_i}$ have been parsed off and the corresponding irreducible grammar $G_i$ for $x_1\cdots x_{n_i}$ has been built. Suppose that the variable set of $G_i$ is equal to

$$\mathcal{S}(j_i) = \{s_0, s_1, \cdots, s_{j_i-1}\}$$

where $j_1 = 1$. The next substring $x_{n_i+1}\cdots x_{n_{i+1}}$ is the longest prefix of $x_{n_i+1}\cdots x_n$ that can be represented by $s_j$ for some $0 < j < j_i$ if such a prefix exists. Otherwise, $x_{n_i+1}\cdots x_{n_{i+1}} = x_{n_i+1}$ with $n_{i+1} = n_i + 1$. If $n_{i+1} - n_i > 1$ and $x_{n_i+1}\cdots x_{n_{i+1}}$ is represented by $s_j$, then append $s_j$ to the right end of $G_i(s_0)$; otherwise, append the symbol $x_{n_i+1}$ to the right end of $G_i(s_0)$. The resulting grammar is admissible, but not necessarily irreducible. Apply Reduction Rules 1–5 to reduce the grammar to an irreducible grammar $G_{i+1}$. Then $G_{i+1}$ represents $x_1\cdots x_{n_{i+1}}$. Repeat this procedure until the whole sequence $x$ is processed. Then the final irreducible grammar $G_t$ represents $x$.

Since only one symbol from $\mathcal{S}(j_i)\cup\mathcal{A}$ is appended to the end of $G_i(s_0)$, not all reduction rules can be applied to get $G_{i+1}$. Furthermore, the order via which reduction rules are applied is unique. Before we see why this is the case, let us look at an example first.

*Example 6:* Let $\mathcal{A} = \{0, 1\}$ and

$$x = 10011100010001110001111111000.$$

Apply the above irreducible grammar transform to $x$. It is easy to see that the first three parsed substrings (or phrases) are 1, 0, and 0. The corresponding irreducible grammars $G_1$, $G_2$, and $G_3$ are given by $\{s_0 \to 1\}$, $\{s_0 \to 10\}$, and $\{s_0 \to 100\}$, respectively. Since $j_3 = 1$, the fourth parsed phrase is $x_4 = 1$. Appending the symbol 1 to the end of $G_3(s_0)$, we get an admissible grammar $G_3'$ given by $\{s_0 \to 1001\}$. $G_3'$ itself is irreducible; so none of Reduction Rules 1–5 can be applied and $G_4$ is equal to $G_3'$. Similarly, the fifth and sixth parsed phrases are $x_5 = 1$ and $x_6 = 1$, respectively; $G_5$ and $G_6$ are given, respectively, by $\{s_0 \to 10011\}$ and $\{s_0 \to 100111\}$. The seventh parsed phrase is $x_7 = 0$. Appending the symbol 0 to the end of $G_6(s_0)$, we get an admissible grammar $G_6'$ given by

$$s_0 \to 1001110.$$

$G_6'$ is not irreducible any more since there is a nonoverlapping repeated pattern 10 in the range of $G_6'$. At this point, only Reduction Rule 2 is applicable. Applying Reduction Rule 2 once, we get the irreducible grammar $G_7$ given by

$$s_0 \to s_1011s_1$$
$$s_1 \to 10.$$

Since the sequence from $\mathcal{A}$ represented by $s_1$ is not a prefix of the remaining part of $x$, the next parsed phrase is $x_8 = 0$. Appending the symbol 0 to the end of $G_7(s_0)$, we get an admissible grammar $G_7'$ given by

$$s_0 \to s_1011s_10$$
$$s_1 \to 10.$$

$G_7'$ is not irreducible. Applying Reduction Rule 2 once, which is the only applicable reduction rule at this point, we get a grammar $G_7''$

$$s_0 \to s_211s_2$$
$$s_1 \to 10$$
$$s_2 \to s_10.$$

In the above, the variable $s_1$ appears only once in the range of $G_7''$. Applying Reduction Rule 1 once, we get our irreducible grammar $G_8$:

$$s_0 \to s_111s_1$$
$$s_1 \to 100.$$

From $G_7$ to $G_8$, we have applied Reduction Rule 2 followed by Reduction Rule 1. Based on $G_8$, the next two parsed phrases are $x_9 = 0$ and $x_{10}x_{11}x_{12} = 100$, respectively. The irreducible grammar $G_9$ is given by

$$s_0 \to s_111s_10$$
$$s_1 \to 100$$

and the grammar $G_{10}$ is given by

$$s_0 \to s_111s_10s_1$$
$$s_1 \to 100.$$

Note that from $G_9$ to $G_{10}$, we simply append the symbol $s_1$ to the end of $G_9(s_0)$ since the phrase $x_{10}x_{11}x_{12}$ is represented by $s_1$. The eleventh parsed phrase is $x_{13} = 0$. Appending 0 to the end of $G_{10}(s_0)$ and then applying Reduction Rule 2 once, we get $G_{11}$

$$s_0 \to s_111s_2s_2$$
$$s_1 \to 100$$
$$s_2 \to s_10.$$

The twelfth parsed phrase is $x_{14} = 1$ and $G_{12}$ is obtained by simply appending 1 to the end of $G_{11}(s_0)$. The thirteenth parsed phrase is $x_{15} = 1$. Appending 1 to the end of $G_{12}(s_0)$ and then applying Reduction Rule 2 once, we get $G_{13}$

$$s_0 \to s_1s_3s_2s_2s_3$$
$$s_1 \to 100$$
$$s_2 \to s_10$$
$$s_3 \to 11.$$

The fourteenth parsed phrase is $x_{16}x_{17}x_{18}x_{19} = 1000$, which is represented by $s_2$. Appending $s_2$ to the end of $G_{13}(s_0)$ and then applying Reduction Rule 2 followed by Reduction Rule 1, we get $G_{14}$

$$s_0 \to s_1s_3s_2s_3$$
$$s_1 \to 100$$
$$s_2 \to s_10$$
$$s_3 \to 11s_2.$$

The fifteenth parsed phrase is $x_{20} = 1$, and $G_{15}$ is obtained by appending 1 to the end of $G_{14}(s_0)$. The sixteenth parsed phrase is $x_{21} = 1$. Appending 1 to the end of $G_{15}(s_0)$ and then applying Reduction Rule 3 once, we get $G_{16}$

$$\begin{aligned} s_0 &\rightarrow s_1 s_3 s_2 s_3 s_4 \\ s_1 &\rightarrow 100 \\ s_2 &\rightarrow s_1 0 \\ s_3 &\rightarrow s_4 s_2 \\ s_4 &\rightarrow 11. \end{aligned}$$

The seventeenth parsed phrase is $x_{22}x_{23} = 11$ and $G_{17}$ is obtained by appending $s_4$ to the end of $G_{16}(s_0)$. The final parsed phrase is $x_{24} \cdots x_{29} = 111000$ and $G_{18}$ is obtained by appending $s_3$ to the end of $G_{17}(s_0)$. In summary, our proposed irreducible grammar transform parses $x$ into

$$\{1, 0, 0, 1, 1, 1, 0, 0, 0, 100, 0, 1, 1, 1000, 1, 1, 11, 111000\}$$

and transforms $x$ into the irreducible grammar $G_{18}$

$$\begin{aligned} s_0 &\rightarrow s_1 s_3 s_2 s_3 s_4 s_4 s_3 \\ s_1 &\rightarrow 100 \\ s_2 &\rightarrow s_1 0 \\ s_3 &\rightarrow s_4 s_2 \\ s_4 &\rightarrow 11. \end{aligned}$$

In Example 6, we see that to get $G_{i+1}$ from the appended $G_i$, only Reduction Rules 1–3 are possibly involved. Furthermore, the order via which these rules are applied is unique, and the number of times these rules need to be applied is at most 2. This phenomenon is true not only for Example 6, but also for all other sequences, as shown in Theorem 1 below.

Before we state Theorem 1, we define a function

$$I\colon \{1, 2, \cdots, t\} \rightarrow \{0, 1\}$$

as follows: $I(1) = 0$, and for any $i > 1$, $I(i)$ is equal to 0 if $G_i$ is equal to the appended $G_{i-1}$, and 1 otherwise. According to this definition, the sequence $\{I(i)\}_{i=1}^{18}$ in Example 6 is 000000110010110100. Note that we assume that the variable set of $G_i$ is $\mathcal{S}(j_i) = \{s_0, s_1, \cdots, s_{j_i-1}\}$.

*Theorem 1:* Let $\alpha$ be the last symbol of $G_i(s_0)$. Let $\beta$ be the symbol $s_j$ that represents $x_{n_i+1} \cdots x_{n_{i+1}}$ if $n_{i+1} - n_i > 1$, and $x_{n_i+1}$ itself otherwise. Let $G_i'$ be the admissible grammar obtained by appending $\beta$ to the end of $G_i(s_0)$. Then the following steps specify how to get $G_{i+1}$ from $G_i'$:

Case 1: The pattern $\alpha\beta$ does not appear in two nonoverlapping positions in the range of $G_i'$. In this case, $G_i'$ is irreducible and hence $G_{i+1}$ is equal to $G_i'$.

Case 2: The pattern $\alpha\beta$ appears in two nonoverlapping positions in the range of $G_i'$, and $I(i) = 0$. In this case, apply Reduction Rule 2 once if the pattern $\alpha\beta$ repeats itself in $G_i'(s_0)$, and Reduction Rule 3 once otherwise. The resulting grammar is irreducible and hence equal to $G_{i+1}$. The variable set of $G_{i+1}$ is $\mathcal{S}(j_{i+1})$ with $j_{i+1} = j_i + 1$, and the newly created production rule is $s_{j_i} \rightarrow \alpha\beta$.

Case 3: The pattern $\alpha\beta$ appears in two nonoverlapping positions in the range of $G_i'$, and $I(i) = 1$. In this case, apply Reduction Rule 2 followed by Reduction Rule 1 if the pattern $\alpha\beta$ repeats itself in $G_i'(s_0)$, and Reduction Rule 3 followed by Reduction Rule 1 otherwise. The resulting grammar is irreducible and hence equal to $G_{i+1}$. The variable set of $G_{i+1}$ is the same as that of $G_i$ with $j_{i+1} = j_i$, and $G_{i+1}(s_{j_{i+1}-1})$ is obtained by appending $\beta$ to the end of $G_i(s_{j_i-1})$.

*Proof:* Since $G_i$ is irreducible, it is easy to see that in the range of $G_i'$, the pattern $\alpha\beta$ is the only possible nonoverlapping repeated pattern of length $\geq 2$. If $\alpha\beta$ does not appear in two nonoverlapping positions in the range of $G_i'$, as in Case 1, then $G_i'$ itself is irreducible. Therefore, in Case 1, $G_{i+1}$ is equal to $G_i'$ and no action needs to be taken.

Let us now look at Cases 2 and 3. If $\alpha\beta$ is a nonoverlapping repeated pattern in the range of $G_i'$, then $\alpha\beta$ repeats itself only once in the range of $G_i'$ since $G_i$ is irreducible. (When $\alpha = \beta$, however, there might be an exception. This exception occurs if the pattern $\alpha\alpha\alpha$ appears somewhere in the range of $G_i'$. Nonetheless, the following argument applies to this case as well. To avoid ambiguity, we shall replace $\alpha\alpha$ on the right by a new variable when Reduction Rule 2 or 3 is applied and this exception occurs. Also, in this case, we still consider that $\alpha\beta$ repeats itself only once.) In Case 2, $I(i) = 0$ implies that the symbol $\alpha$ represents the $i$th phrase $x_{n_{i-1}+1} \cdots x_{n_i}$. Since $\beta$ represents the $(i+1)$th phrase, it is not hard to see that Reduction Rule 4 is not applicable in this case. To see this is true, suppose that there is a production rule $s_j \rightarrow \alpha\beta$ for some $0 < j < j_i$ in $G_i'$. Since $I(i) = 0$, $G_{i-1}$, $G_i$, and $G_i'$ all have the same variable set, and for any $s_j$ other than $s_0$, $G_{i-1}(s_j)$, $G_i(s_j)$, and $G_i'(s_j)$ are all the same. In view of the greedy nature of the proposed irreducible grammar transform, the production rule $s_j \rightarrow \alpha\beta$ in $G_{i-1}$ then implies that the $i$th phrase is $x_{n_{i-1}+1} \cdots x_{n_i} x_{n_i+1} \cdots x_{n_{i+1}}$ instead of $x_{n_{i-1}+1} \cdots x_{n_i}$. This is a contradiction. Thus at this point, only Reduction Rule 2 or 3 is applicable. Apply Reduction Rule 2 once if the pattern $\alpha\beta$ repeats itself in $G_i'(s_0)$; otherwise, apply Reduction Rule 3 once. The resulting grammar has a variable set $\mathcal{S}(j_i + 1)$ and a new production rule $s_{j_i} \rightarrow \alpha\beta$. We claim that the resulting grammar is irreducible and hence equal to $G_{i+1}$. To see this is true, first note that there is no nonoverlapping repeated pattern of length $\geq 2$ any more in the resulting grammar, since $\alpha\beta$ is the only nonoverlapping repeated pattern of length $\geq 2$ in the range of $G_i'$ and repeats itself only once in the range of $G_i'$. Second, if $\alpha$ is a variable, then $I(i) = 0$ implies that $\alpha$ appears in the range of $G_i$ at least three times. If $\alpha = \beta$, then $\alpha$ appears in the range of $G_i'$ at least four times; as a result, when a new production rule $s_{j_i} \rightarrow \alpha\beta$ (which is $s_{j_i} \rightarrow \alpha\alpha$ in this special case) is introduced, each variable $s \in \mathcal{S}(j_i + 1)$ other than $s_0$ still appears at least twice in the range of the resulting grammar. On the other hand, if $\alpha \neq \beta$ and $\beta$ is a variable, then $\beta$ appears in the range of $G_i'$ at least three times; as a result, when a new rule $s_{j_i} \rightarrow \alpha\beta$ is introduced, each variable $s \in \mathcal{S}(j_i + 1)$ other than $s_0$ still appears at least twice in the range of the resulting grammar. The result also holds in all other cases: neither $\alpha$ nor $\beta$ is a variable or only one of them is a variable. Finally, the new variable $s_{j_i}$ represents the sequence $x_{n_{i-1}+1} \cdots x_{n_i} x_{n_i+1} \cdots x_{n_{i+1}}$ which is

distinct from all other sequences represented by $s_j, 0 \le j < j_i$. To see this is true, note that otherwise, one gets the contradiction that the $i$th parsed phrase is $x_{n_i-1+1} \cdots x_{n_i} x_{n_i+1} \cdots x_{n_{i+1}}$ instead of $x_{n_i-1+1} \cdots x_{n_i}$. Therefore, the resulting grammar is indeed irreducible and hence equal to $G_{i+1}$.

In Case 3, $I(i) = 1$ implies that $\alpha$ is equal to the newly introduced variable $s_{j_i-1}$ in $G_i$ and appears only twice in the range of $G_i$. Using mathematical induction, one can show that in this case, $\alpha$ represents the substring obtained by concatenating the $i$th parsed phrase, the $(i-1)$th parsed phrase, ..., and up to the $(i-l)$th parsed phrase for some $l \ge 1$. Note that in Case 3, $\alpha \ne \beta$, and $\alpha\beta$ repeats itself only once in the range of $G_i'$. A similar argument to that in the above paragraph can be used to show that at this point, Reduction Rule 4 is not applicable. Apply Reduction Rule 2 once if the pattern $\alpha\beta$ repeats itself in $G_i'(s_0)$; otherwise, apply Reduction Rule 3 once. The resulting grammar, which is denoted by $G_i''$, has a variable set $\mathcal{S}(j_i+1)$ and a new production rule $s_{j_i} \to \alpha\beta$. However, the resulting grammar $G_i''$ is not irreducible since $\alpha$ appears only twice in the range of $G_i''$ and as a result, $\alpha$ appears only once in the range of $G_i''$. In fact, $\alpha$ appears only in the newly introduced rule $s_{j_i} \to \alpha\beta$. Apply Reduction Rule 1 to $G_i''$ and change $s_{j_i}$ back to $s_{j_i-1}$. The resulting grammar has the same variable set $\mathcal{S}(j_i)$ as does $G_i$, and the production rule corresponding to $s_{j_i-1}$ is obtained by appending $\beta$ to the end of $G_i(s_{j_i-1})$. We now claim that the resulting grammar is irreducible and hence equal to $G_{i+1}$. To see that this is true, first note that since both $G_{i-1}$ and $G_i$ are irreducible and since $\alpha\beta$ is the only repeated pattern of length $\ge 2$ and repeats itself only once in the range of $G_i'$, there is no nonoverlapping repeated pattern of length $\ge 2$ in the range of the resulting grammar. (Note that the irreducibility of $G_{i-1}$ guarantees that the pattern consisting of the last symbol of $G_i(s_{j_i-1})$ and $\beta$ in the range of the resulting grammar is not a nonoverlapping repeated pattern.) Second, if $\beta$ is a variable, then $\beta$ appears at least three times in the range of $G_i'$; as a result, every variable other than $s_0$ in the resulting grammar appears at least twice in the range of the resulting grammar. Finally, due to the greedy nature of the proposed irreducible grammar transform, the variable $s_{j_i-1}$ in the resulting grammar represents the sequence obtained by concatenating the $(i+1)$th parsed phrase, the $i$th parsed phrase, ..., and up to the $(i-l)$th parsed phrase, which is distinct from all other sequences represented by $s_j$, $0 \le j < j_i - 1$. Therefore, the resulting grammar is irreducible and equal to $G_{i+1}$.

Finally, note that there is no other case other than Cases 1–3. This completes the proof of Theorem 1.

From Theorem 1, we see that once the $(i+1)$th phrase is parsed off, it is pretty fast to get $G_{i+1}$ from the appended $G_i$.

*Remark 1:* There is a variant of the proposed irreducible grammar transform in which the next substring $x_{n_i+1} \cdots x_{n_{i+1}}$ is the longest prefix of $x_{n_i+1} \cdots x_n$ that can be represented by $s_j$ for some $0 \le j < j_i$ if such a prefix exists, and otherwise, $x_{n_i+1} \cdots x_{n_{i+1}} = x_{n_i+1}$ with $n_{i+1} = n_i + 1$. In other words, the symbol $s_0$ now is also involved in the parsing. To get $G_{i+1}$ from the appended $G_i$, special attention must be paid to the case where $s_0$ is appended to the end of $G_i(s_0)$; in this case, one changes $s_0$ in $G_i$ to a new variable $s_{j_i}$ and introduces a new rule $s_0 \to s_{j_i} s_{j_i}$. This variant was first described by the authors in [14]. All the results in this paper hold as well for this variant. We shall not use this variant as our grammar transform since, in practice, it is highly unlikely that the entire previously processed string will occur again right away (except near the beginning of the data).

*Remark 2:* In their recent paper [18], Nevill-Manning and Witten presented independently a grammar transform that constructs sequentially a sequence of grammars. However, the grammars constructed by their transform are not necessarily irreducible because they do not satisfy Property b.3). As a result, the corresponding grammar code may not be universal.

## IV. Universal Algorithms

Having described our irreducible grammar transform, we now describe our compression algorithms: a hierarchical algorithm, a sequential algorithm, and an improved sequential algorithm. They share the common greedy grammar transform, but have different encoding strategies. We first describe the hierarchical algorithm which consists of the greedy irreducible grammar transform followed by the arithmetic coding of the final irreducible grammar $G_t$.

*The Hierarchical Algorithm:* Let $x$ be an $\mathcal{A}$ sequence which is to be compressed. Let $G_t$ be the final irreducible grammar for $x$ furnished by our irreducible grammar transform. In the hierarchical algorithm, we use a zero-order arithmetic code with a dynamic alphabet to encode $G_t$ (or its equivalent form). After receiving the binary codeword, the decoder recovers $G_t$ (or its equivalent form) and then performs the parallel replacement procedure mentioned in Section II to get $x$.

To illustrate how to encode the final irreducible grammar $G_t$, let us look at Example 6 again. The final irreducible grammar for the sequence $x$ in Example 6 is $G_{18}$ given by

$$\begin{aligned} s_0 &\to s_1 s_3 s_2 s_3 s_4 s_4 s_3 \\ s_1 &\to 100 \\ s_2 &\to s_1 0 \\ s_3 &\to s_4 s_2 \\ s_4 &\to 11. \end{aligned}$$

The above form of $G_{18}$, however, is not convenient for transmission. To encode $G_{18}$ efficiently, we first convert $G_{18}$ into its canonical form $G_{18}^g$ given by

$$\begin{aligned} s_0 &\to s_1 s_2 s_3 s_2 s_4 s_4 s_2 \\ s_1 &\to 100 \\ s_2 &\to s_4 s_3 \\ s_3 &\to s_1 0 \\ s_4 &\to 11. \end{aligned}$$

To get $G_{18}^g$ from $G_{18}$, we simply rename the variable $s_3$ in $G_{18}$ as $s_2$ in $G_{18}^g$ and the variable $s_2$ in $G_{18}$ as $s_3$ in $G_{18}^g$. The difference between $G_{18}^g$ and $G_{18}$ is that the following property holds for $G_{18}^g$, but not for $G_{18}$:

c.1) If we read $G_{18}^g(s_i)$ from left to right and from top ($i = 0$) to bottom ($i = 4$), then for any $j \ge 1$, the first appearance of $s_j$ always precedes that of $s_{j+1}$.

In $G_{18}$, the first appearance of $s_3$ precedes that of $s_2$; this is why we need to rename these two variables to get $G_{18}^g$. Note that both $G_{18}$ and $G_{18}^g$ represent the same sequence $x$. We now encode and transmit $G_{18}^g$ instead of $G_{18}$. To do so, we concatenate $G_{18}^g(s_0)$, $G_{18}^g(s_1)$, $\cdots$, $G_{18}^g(s_4)$ in the indicated order, insert symbol $e$ at the end of $G_{18}^g(s_0)$, and for any $i \geq 1$ satisfying $|G_{18}^g(s_i)| > 2$, insert symbol $b$ at the beginning of $G_{18}^g(s_i)$ and symbol $e$ at the end of $G_{18}^g(s_i)$, where symbols $b$ and $e$ are assumed not to belong to $\mathcal{S} \cup \mathcal{A}$. This gives rise to the following sequence from the alphabet $\mathcal{S} \cup \mathcal{A} \cup \{b, e\}$:

$$s_1 s_2 s_3 s_2 s_4 s_4 s_2 e b 100 e s_4 s_3 s_1 011 \tag{4.1}$$

where $\mathcal{A} = \{0, 1\}$ in this example. From the sequence given by (4.1), one can easily get $G_{18}^g$ back. First, $G_{18}^g(s_0)$ can be identified by looking at the first appearance of symbol $e$. Second, all $G_{18}^g(s_i)$ with $|G_{18}^g(s_i)| > 2$ can be identified by looking at pairs $(b, e)$. Finally, all the other $G_{18}^g(s_i)$ have length 2. (One may wonder why we need to introduce both symbols $e$ and $b$; after all, we can insert $e$ at the end of each $G_{18}^g(s_i)$ to identify $G_{18}^g(s_i)$. The reason is that most $G_t(s_i)$ of any $G_t$ furnished by our irreducible grammar transform have length 2. As a result, by using the pair $(b, e)$ to isolate $G_t(s_i)$ with $|G_t(s_i)| > 2$, we actually get a shorter concatenated sequence and hence better compression performance.) Since $G_{18}^g$ is canonical, i.e., $G_{18}^g$ satisfies Property c.1), the first appearance of $s_i$, for any $i \geq 1$, precedes that of $s_{i+1}$ in the sequence given by (4.1). To take advantage of this in order to get better compression performance, we go one step further. Let $s$ be a symbol which is not in $\mathcal{S} \cup \mathcal{A} \cup \{b, e\}$. For each $i \geq 1$, replace the first appearance of $s_i$ in the sequence given by (4.1) by $s$. Then we get the following sequence from $\mathcal{S} \cup \mathcal{A} \cup \{b, e, s\}$:

$$ssss_2 s s_4 s_2 e b 100 e s_4 s_3 s_1 011 \tag{4.2}$$

which will be called the sequence generated from $G_{18}$ or its canonical form $G_{18}^g$. Clearly, from the sequence given by (4.2), we can get the sequence given by (4.1) back by simply replacing the $i$th $s$ in (4.2) by $s_i$. Therefore, from the sequence generated from $G_{18}$, we can get $G_{18}^g$ and hence $x$. To compress $G_{18}^g$ or $x$, we now use a zero-order arithmetic code with a dynamic alphabet to encode the sequence generated from $G_{18}$. Specifically, we associate each symbol $\beta \in \mathcal{S} \cup \mathcal{A} \cup \{b, e, s\}$ with a counter $c(\beta)$. Initially, $c(\beta)$ is set to 1 if $\beta \in \mathcal{A} \cup \{b, e, s\}$ and 0 otherwise. The initial alphabet used by the arithmetic code is $\mathcal{A} \cup \{b, e, s\}$. Encode each symbol $\beta$ in the sequence generated from $G_{18}$ and update the related counters according to the following steps:

Step 1: Encode $\beta$ by using the probability

$$c(\beta) \Big/ \sum_{\alpha} c(\alpha)$$

where the summation $\sum_{\alpha}$ is taken over $\mathcal{A} \cup \{b, e, s\} \cup \{s_1, \cdots, s_i\}$, and $i$ is the number of times that $s$ occurs before the position of this $\beta$. Note that the alphabet used at this point by the arithmetic code is $\mathcal{A} \cup \{b, e, s\} \cup \{s_1, \cdots, s_i\}$.

Step 2: Increase the counter $c(\beta)$ by 1.

Step 3: If $\beta = s$, increase the counter $c(s_{i+1})$ from 0 to 1, where $i$ is defined in Step 1.

Repeat the above procedure until the whole generated sequence is encoded. For the generated sequence given by (4.2), the product of the probabilities used in the arithmetic coding process is

$$p = \tfrac{1}{5}\tfrac{2}{7}\tfrac{3}{9}\tfrac{1}{11}\tfrac{4}{12}\tfrac{1}{14}\tfrac{2}{15}\tfrac{1}{16}\tfrac{1}{17}\tfrac{1}{18}\tfrac{1}{19}\tfrac{2}{20}\tfrac{2}{21}\tfrac{2}{22}\tfrac{1}{23}\tfrac{1}{24}\tfrac{3}{25}\tfrac{2}{26}\tfrac{3}{27}.$$

In general, to encode the final irreducible grammar $G_t$, we first convert it into its canonical form $G_t^g$, then construct the sequence generated from $G_t$, and finally use a zero-order arithmetic code with a dynamic alphabet to encode the generated sequence.

*Remark 3:* It should be pointed out that in practice, there is no need to write down explicitly the canonical form $G_t^g$ and the generated sequence before embarking on arithmetic coding. The converting of $G_t$ into $G_t^g$, constructing of the generated sequence, and encoding of the generated sequence can all be done simultaneously in one pass, assuming that $G_t$ has been furnished by our irreducible grammar transform.

*Remark 4:* A different method for encoding canonical grammars has been presented in [12]; it is based on the concept of enumerative coding [6]. The method presented here is intuitive and more efficient.

The sequential nature of our greedy irreducible grammar transform makes it possible to parse and encode the $\mathcal{A}$-sequence $x$ simultaneously.

*The Sequential Algorithm:* In the sequential algorithm, we encode the data sequence $x$ sequentially by using a zero-order arithmetic code with a dynamic alphabet to encode the sequence of parsed phrases $x_1, x_2 \cdots x_{n_2}, \cdots, x_{n_{t-1}+1} \cdots x_{n_t}$. Specifically, we associate each symbol $\beta \in \mathcal{S} \cup \mathcal{A}$ with a counter $c(\beta)$. Initially, $c(\beta)$ is set to 1 if $\beta \in \mathcal{A}$ and 0 otherwise. At the beginning, the alphabet used by the arithmetic code is $\mathcal{A}$. The first parsed phrase $x_1$ is encoded by using the probability $c(x_1) / \sum_{\beta \in \mathcal{A}} c(\beta)$. Then the counter $c(x_1)$ increases by 1. Suppose that $x_1, x_2 \cdots x_{n_2}, \cdots, x_{n_{i-1}+1} \cdots x_{n_i}$ have been parsed off and encoded and that all corresponding counters have been updated. Let $G_i$ be the corresponding irreducible grammar for $x_1 \cdots x_{n_i}$. Assume that the variable set of $G_i$ is equal to $\mathcal{S}(j_i) = \{s_0, s_1, \cdots, s_{j_i - 1}\}$. Let $x_{n_i+1} \cdots x_{n_{i+1}}$ be parsed off as in our irreducible grammar transform and represented by $\beta \in \{s_1, \cdots, s_{j_i - 1}\} \cup \mathcal{A}$. Encode $x_{n_i+1} \cdots x_{n_{i+1}}$ (or $\beta$) and update the relevant counters according to the following steps:

Step 1: The alphabet used at this point by the arithmetic code is $\{s_1, \cdots, s_{j_i - 1}\} \cup \mathcal{A}$. Encode $x_{n_i+1} \cdots x_{n_{i+1}}$ by using the probability

$$c(\beta) \Big/ \sum_{\alpha \in \mathcal{S}(j_i) \cup \mathcal{A}} c(\alpha). \tag{4.3}$$

Step 2: Increase $c(\beta)$ by 1.

Step 3: Get $G_{i+1}$ from the appended $G_i$ as in our irreducible grammar transform.

Step 4: If $j_{i+1} > j_i$, i.e., $G_{i+1}$ includes the new variable $s_{j_i}$, increase the counter $c(s_{j_i})$ by 1.

Repeat this procedure until the whole sequence $x$ is processed and encoded.

Note that $c(s_0)$ is always 0. Thus the summation over $\mathcal{S}(j_i) \cup \mathcal{A}$ in (4.3) is equivalent to the summation over

$\{s_1, \cdots, s_{j_i-1}\} \cup \mathcal{A}$. From Step 4, it follows that each time when a new variable $s_{j_i}$ is introduced, its counter increases from 0 to 1. Therefore, in the entire encoding process, there is no zero-frequency problem. Also, in the sequential algorithm, the parsing of phrases, encoding of phrases, and updating of irreducible grammars are all done in one pass. Clearly, after receiving enough codebits to recover the symbol $\beta$, the decoder can perform the update operation in the exact same way as does the encoder.

*Remark 5:* It is interesting to compare the sequential algorithm with LZ78. In LZ78, the parsed phrases are all distinct. As a result, there is no room for arithmetic coding, which operates on phrases rather than on symbols from $\mathcal{A}$, to kick in. On the other hand, in our sequential compression algorithm, parsed phrases are of variable length and allowed to repeat themselves. Moreover, there is no upper bound on the number of repetitions of each parsed phrase. As a result, there is room for arithmetic coding, which operates on phrases, to kick in. Our irreducible grammar update mechanism acts like a string-matching mechanism and provides candidates for new parsed phrases. One of the important roles of our irreducible grammar update mechanism is to maintain a good tradeoff among the length $|x|$, the number $t$ of parsed phrases, and the number $j_t$ of variables so that good compression performance can be obtained. In Section VI, we will show that the sequential algorithm is universal for the class of stationary, ergodic sources and has the worst case redundancy upper bound $O(\log\log|x|/\log|x|)$. Although both our sequential algorithm and LZ78 are universal for the class of stationary, ergodic sources, the simulation results presented in Section VII show that our sequential algorithm is better than Unix Compress, which is based on LZ78.

*Example 7:* We apply our sequential algorithm to compress the sequence

$$x = 1001110001000111000011111111000$$

shown in Example 6. It follows from Example 6 that $x$ is parsed into

$$\{1, 0, 0, 1, 1, 1, 0, 0, 0, 100, 0, 1, 1, 1000, 1, 1, 11, 111000\}.$$

The product of the probabilities used to encode these parsed phrases is

$$p = \tfrac{1}{2}\tfrac{1}{3}\tfrac{2}{4}\tfrac{2}{5}\tfrac{3}{6}\tfrac{4}{7}\tfrac{3}{8}\tfrac{4}{10}\tfrac{5}{11}\tfrac{1}{12}\tfrac{6}{13}\tfrac{5}{15}\tfrac{6}{16}\tfrac{1}{18}\tfrac{7}{19}\tfrac{8}{20}\tfrac{1}{22}\tfrac{1}{23}.$$

Careful examination of the above sequential algorithm reveals that the encoding of the sequence of parsed phrases does not utilize the structure of the irreducible grammars $G_i$, $1 \le i \le t$. Since $G_i$ is known to the decoder before encoding the $(i+1)$th parsed phrase, we can use the structure of $G_i$ as context information to reduce the codebits for the $(i+1)$th parsed phrase. To this end, we associate each symbol $\gamma \in \mathcal{S} \cup \mathcal{A}$ with two lists $L_1(\gamma)$ and $L_2(\gamma)$. The list $L_1(\gamma)$ consists of all symbols $\eta \in \mathcal{S} \cup \mathcal{A}$ such that the following properties are satisfied:

d.1) The pattern $\gamma\eta$ appears in the range of $G_i$.

d.2) The pattern $\gamma\eta$ is not the last two symbols of $G_i(s_0)$.

d.3) There is no variable $s_j$ of $G_i$ such that $G_i(s_j)$ is equal to $\gamma\eta$.

The list $L_2(\gamma)$ consists of all symbols $\eta \in \mathcal{S} \cup \mathcal{A}$ such that Properties d.1) and d.2) hold. The elements in $L_1(\gamma)$ (or $L_2(\gamma)$) can be arranged in some order. We can use the lists $L_1(\gamma)$ and $L_2(\gamma)$ to facilitate the process of updating $G_i$ to $G_{i+1}$ and to improve the encoding process in the above sequential algorithm. Let $\alpha$ be the last symbol of $G_i(s_0)$. Let the $(i+1)$th parsed phrase $x_{n_i+1}\cdots x_{n_{i+1}}$ be represented by $\beta \in \{s_1, \cdots, s_{j_i-1}\} \cup \mathcal{A}$. Then it follows from Theorem 1 and its proof that the pattern $\alpha\beta$ appears in two nonoverlapping positions in the range of the appended $G_i$ if and only if $\beta$ appears in the list $L_1(\alpha)$. To see how to use the lists $L_1(\gamma)$ and $L_2(\gamma)$ to improve the encoding process in the sequential algorithm, we recall from Section III that $I(i)$ is equal to 0 if $G_i$ is equal to the appended $G_{i-1}$, and 1 otherwise. From Theorem 1 and its proof, it follows that when $I(i) = 0$ and $I(i+1) = 1$, the symbol $\beta$ appears in the list $L_1(\alpha)$, and hence one can simply send the index of $\beta$ in $L_1(\alpha)$ to the decoder. When $I(i) = 1$ and $I(i+1) = 1$, $\beta$ is the only element in the list $L_1(\alpha)$ and thus no information needs to be sent. Therefore, if we transmit the bit $I(i+1)$ to the decoder, then we can use the bit $I(i+1)$ and the structure of $G_i$ to improve the encoding of $\beta$. This suggests the following improved sequential compression algorithm.

*The Improved Sequential Algorithm:* In the improved sequential algorithm, we use an order 1 arithmetic code to encode the sequence $\{I(i)\}_{i=1}^t$, and then use the sequence $\{I(i)\}_{i=1}^t$ and the structures of $\{G_i\}_{i=1}^t$ to improve the encoding of the sequence of parsed phrases $x_1, x_2\cdots x_{n_2}, \cdots, x_{n_{t-1}+1}\cdots x_{n_t}$. In addition to the counters $c(\gamma)$, $\gamma \in \mathcal{S} \cup \mathcal{A}$, we now define new counters $c(0,0)$, $c(0,1)$, $c(1,0)$, $c(1,1)$, and $\hat{c}(\gamma)$. The counters $c(0,0)$, $c(0,1)$, $c(1,0)$, and $c(1,1)$ are used to encode the sequence $\{I(i)\}_{i=1}^t$; initially, they are all equal to 1. The $(i+1)$th parsed phrase is encoded by the counters $\hat{c}(\gamma)$ whenever $I(i) = 0$ and $I(i+1) = 1$ and by the counters $c(\gamma)$ whenever $I(i+1) = 0$. As in the case of $c(\gamma)$, initially $\hat{c}(\gamma)$ is 1 if $\gamma \in \mathcal{A}$ and 0 if $\gamma \in \mathcal{S}$. The first three parsed phrases are encoded as in the sequential algorithm since they are $x_1$, $x_2$, and $x_3$. Also, $I(1)$, $I(2)$, and $I(3)$ are all 0 and hence there is no need to encode them. Starting from the fourth phrase, we first encode $I(i+1)$, and then use $I(i+1)$ as side information and the structure of $G_i$ as context information to encode the $(i+1)$th parsed phrase. Suppose that $x_1, x_2\cdots x_{n_2}, \cdots, x_{n_{i-1}+1}\cdots x_{n_i}$ and $I(4), \cdots, I(i)$ have been encoded and that all corresponding counters have been updated. Let $G_i$ be the corresponding irreducible grammar for $x_1\cdots x_{n_i}$. Assume that the variable set of $G_i$ is equal to $\mathcal{S}(j_i) = \{s_0, s_1, \cdots, s_{j_i-1}\}$. Let $\alpha$ be the last symbol of $G_i(s_0)$. Let $x_{n_i+1}\cdots x_{n_{i+1}}$ be parsed off as in our irreducible grammar transform and represented by $\beta \in \{s_1, \cdots, s_{j_i-1}\} \cup \mathcal{A}$. Encode $I(i+1)$ and $\beta$, and update the relevant counters according to the following steps:

Step 1: Encode $I(i+1)$ by using the probability

$$c(I(i), I(i+1))/(c(I(i), 0) + c(I(i), 1)).$$

Step 2: Increase $c(I(i), I(i+1))$ by 1.

Step 3: If $I(i+1)=0$, encode $\beta$ by using the probability

$$c(\beta) \Big/ \sum_{\gamma \in \mathcal{S}(j_i) \cup \mathcal{A} - L_2(\alpha)} c(\gamma) \tag{4.4}$$

and then increase $c(\beta)$ by 1. If $I(i)=0$ and $I(i+1)=1$, encode $\beta$ by using the probability

$$\hat{c}(\beta) \Big/ \sum_{\gamma \in L_1(\alpha)} \hat{c}(\gamma) \tag{4.5}$$

and then increase $\hat{c}(\beta)$ by 1. On the other hand, if $I(i)=1$ and $I(i+1)=1$, no information is sent since $L_1(\alpha)$ contains only one element and the decoder knows what $\beta$ is.

Step 4: Get $G_{i+1}$ from the appended $G_i$ as in our irreducible grammar transform. Update $L_1(\gamma)$ and $L_2(\gamma)$ accordingly, where $\gamma \in \mathcal{S}(j_{i+1}) \cup \mathcal{A}$.

Step 5: If $j_{i+1} > j_i$, i.e., $G_{i+1}$ includes the new variable $s_{j_i}$, increase both $c(s_{j_i})$ and $\hat{c}(s_{j_i})$ by 1.

Repeat this procedure until the whole sequence $x$ is processed and encoded.

Note that in view of Theorem 1 and its proof, one can determine $I(i+1)$ by examining whether or not $\beta$ is in $L_1(\alpha)$. Therefore, one can perform the encoding operation of $\beta$ before updating $G_i$ to $G_{i+1}$. In Step 3, when $I(i+1)=0$, $\beta$ cannot be from $L_2(\alpha)$; when $I(i+1)=1$, $\beta$ is from $L_1(\alpha)$. Once again, this follows from Theorem 1 and its proof. The alphabet used in the arithmetic coding is $\{s_1, \cdots, s_{j_i-1}\} \cup \mathcal{A} - L_2(\alpha)$ when $I(i+1)=0$, and $L_1(\alpha)$ when $I(i)=0$ and $I(i+1)=1$.

*Example 7′:* We apply the improved sequential algorithm to compress the sequence

$$x = 1001110001000111000011111111000$$

shown in Example 6. It follows from Example 6 that $x$ is parsed into

$$\{1, 0, 0, 1, 1, 1, 0, 0, 0, 100, 0, 1, 1, 1000, 1, 1, 11, 111000\}.$$

The corresponding sequence $\{I(i)\}$ is

$$000000110010110100.$$

The product of the probabilities used to encode the sequence $\{I(i)\}_{i=4}^{18}$ is

$$\tfrac{1}{2}\ \tfrac{2}{3}\ \tfrac{3}{4}\ \tfrac{1}{5}\ \tfrac{1}{2}\ \tfrac{1}{3}\ \tfrac{4}{6}\ \tfrac{2}{7}\ \tfrac{2}{4}\ \tfrac{3}{8}\ \tfrac{2}{5}\ \tfrac{3}{6}\ \tfrac{4}{9}\ \tfrac{4}{7}\ \tfrac{5}{10}.$$

The product of the probabilities used to encode the parsed phrases is

$$\tfrac{1}{2}\ \tfrac{1}{3}\ \tfrac{2}{4}\ \tfrac{2}{5}\ \tfrac{3}{3}\ \tfrac{4}{4}\ \tfrac{1}{2}\ \tfrac{3}{4}\ \tfrac{1}{6}\ \tfrac{2}{3}\ \tfrac{5}{12}\ \tfrac{1}{5}\ \tfrac{6}{13}\ \tfrac{2}{5}\ \tfrac{1}{15}\ \tfrac{1}{16}.$$

Note that the $(i+1)$th parsed phrase need not be encoded whenever $I(i)=1$ and $I(i+1)=1$.

*Remark 6:* Assume that exact arithmetic is used. Then for the binary sequence $x$ shown in Example 6, the compression rates in bits per letter given by the hierarchical algorithm, sequential algorithm, and improved sequential algorithm are 2.179, 1.179, and 1.383, respectively. In this particular case, instead of having compression, we get rather expansion. The reason for this is, of course, that the length of the sequence $x$ is quite small. We use this sequence only for the purpose of illustrating how these algorithms work. For long data sequences, simulation results presented in Section VII and in [31] show that the improved sequential algorithm is the best and yields very good compression performance

.

## V. Performance of the Hierarchical Algorithm

In this section, we analyze the performance of the hierarchical compression algorithm and provide some insights into its workings. Some of the results presented in this section will be used to analyze the performance of the sequential and improved sequential algorithms.

Let $x = x_1 x_2 \cdots x_n$ be a sequence from $\mathcal{A}$. Let $G$ be any irreducible grammar that represents $x$. Our methodology is to identify a proper parsing of $x$ induced by $G$ and then relate the compression rate of the hierarchical algorithm to the empirical entropy of the induced parsing of $x$. To ultimately evaluate the compression performance of the hierarchical algorithm against the $k$-context empirical entropy of $x$, which is defined later in this section, several bounds on the number of phrases in the induced parsing of $x$ are essential. These bounds are established via Lemmas 1–4.

Assume that the variable set of $G$ is

$$\mathcal{S}(j) = \{s_0, s_1, \cdots, s_{j-1}\}$$

for some $j \geq 1$. We first explain how $G$ induces a partition of $x$. Let $\mathcal{S}_G$ denote a dynamically changing subset of $\mathcal{S}(j)$; initially, $\mathcal{S}_G$ is empty. Let $u^{(0)} = G(s_0)$; $u^{(0)}$ is a sequence from $\mathcal{S}(j) \cup \mathcal{A}$. If $j=1$, or equivalently if there is no variable in $u^{(0)}$, then $u^{(0)}$ itself is called the *partition sequence* induced by $G$. Otherwise, do the following.

Step 1: Set $i=0$.

Step 2: For $i \geq 0$, read $u^{(i)}$ from left to right. Replace the first variable $s$ which is not in $\mathcal{S}_G$ by $G(s)$. The resulting sequence is denoted by $u^{(i+1)}$.

Step 3: Update $\mathcal{S}_G$ by inserting the variable $s$ into $\mathcal{S}_G$.

Step 4: Repeat Steps 2 and 3 for $i = 0, 1, \cdots, (j-2)$ so that each variable $s \in \{s_1, s_2, \cdots, s_{j-1}\}$ is replaced by $G(s)$ exactly once.

In the final sequence $u^{(j-1)}$, every variable is from $\mathcal{S}_G$. The final sequence $u^{(j-1)}$ is called the *partition sequence* induced by $G$. Recall from Section II that each variable $s \in \mathcal{S}(j)$ represents a distinct substring of $x$. The partition sequence $u^{(j-1)}$ induces a parsing of $x$ if each symbol in $u^{(j-1)}$ is replaced with the corresponding substring of $x$. The given sequence $x$ is the concatenation of the phrases in this parsing.

To illustrate the above idea, let us revisit Example 6.

*Example 8:* In Example 6, the sequence

$$x = 1001110001000111000011111111000$$

is represented by the irreducible grammar $G_{18}$

$$\begin{aligned} s_0 &\to s_1 s_3 s_2 s_3 s_4 s_4 s_3 \\ s_1 &\to 100 \\ s_2 &\to s_1 0 \\ s_3 &\to s_4 s_2 \\ s_4 &\to 11. \end{aligned}$$

In this example, $j = 5$. The five sequences $u^{(0)}, u^{(1)}, \cdots, u^{4)}$ are

$$u^{(0)} = s_1 s_3 s_2 s_3 s_4 s_4 s_3,$$
$$u^{(1)} = 100 s_3 s_2 s_3 s_4 s_4 s_3,$$
$$u^{(2)} = 100 s_4 s_2 s_2 s_3 s_4 s_4 s_3,$$
$$u^{(3)} = 10011 s_2 s_2 s_3 s_4 s_4 s_3,$$

and

$$u^{4)} = 10011 s_1 0 s_2 s_3 s_4 s_4 s_3.$$

The dynamic set $\mathcal{S}_G$ goes from the empty set to the set $\{s_1, s_3, s_4, s_2\}$, as shown below

$$\{\} \rightarrow \{s_1\} \rightarrow \{s_1, s_3\} \rightarrow \{s_1, s_3, s_4\} \rightarrow \{s_1, s_3, s_4, s_2\}.$$

Note that $s_1$ represents 100, $s_2$ represents 1000, $s_3$ represents 111000, and $s_4$ represents 11. The partition sequence $u^{(4)}$ partitions $x$ into

$$1, 0, 0, 1, 1, 100, 0, 1000, 111000, 11, 11, 111000.$$

It is easy to see that the concatenation of the above phrases is equal to $x$. Also, note that the length of the partition sequence is 12, which is equal to $|G_{18}| - (j-1)$.

In the case where $G$ happens to be the irreducible grammar $G_t$ furnished by our irreducible grammar transform, the parsing of $x$ induced by the partition sequence is related, but not equal, to that furnished by our irreducible grammar transform. This can be seen by comparing Example 8 with Example 6. The parsing of $x$ induced by the partition sequence can be used to evaluate the performance of the hierarchical compression algorithm while the parsing of $x$ furnished by our irreducible grammar transform can be used to evaluate the performance of the sequential algorithm. The number of phrases in the parsing of $x$ induced by the partition sequence is less than the number of phrases in the parsing of $x$ furnished by our irreducible grammar transform; the improved sequential algorithm tries to encode directly the parsing of $x$ induced by the partition sequence.

The following lemma relates the length of the partition sequence to the size of $G$.

*Lemma 1:* Let $G$ be an irreducible grammar with variable set $\mathcal{S}(j) = \{s_0, s_1, \cdots, s_{j-1}\}$. Then the length of the partition sequence $u^{(j-1)}$ induced by $G$ is equal to $|G| - j + 1$.

*Proof:* Lemma 1 follows immediately from the observation that in the whole process of deriving the partition sequence $u^{(j-1)}$, each $s_i$, $0 < i < j$, is replaced only once.

From now on, we concentrate on irreducible grammars furnished by our irreducible grammar transform. Let $x = x_1 x_2 \cdots x_n$ be a sequence from $\mathcal{A}$. Let $G_t$ be the final irreducible grammar with variable set $\mathcal{S}(j_t)$ resulting from applying our irreducible grammar transform to $x$. Then we have the following lemma.

*Lemma 2:* In the partition sequence $u^{(j_t-1)}(G_t)$ induced by $G_t$, there is no repeated pattern of length 3, where patterns are counted in the sliding-window, overlapping manner.

*Proof:* We prove Lemma 2 by induction. Clearly, Lemma 2 is true for any $G_i$ with $i \leq 3$ since in this case, the irreducible grammar $G_i$ consists of only one production rule $s_0 \rightarrow x_1 \cdots x_i$. Suppose now that Lemma 2 is true for $G_i$ with $i > 3$. We next want to show that it is also true for $G_{i+1}$. In view of Theorem 1, different reduction rules need to be applied in order to get $G_{i+1}$ from the appended $G_i$. It is easy to see that in Case 3 of Theorem 1, the partition sequence $u^{(j_{i+1}-1)}(G_{i+1})$ is the same as $u^{(j_i-1)}(G_i)$. (For example, the irreducible grammars $G_{13}$ and $G_{14}$ in Example 6 induce the same partition sequence $10011 s_1 0 s_2 s_3$.) Thus in this case, our assumption implies that Lemma 2 is true for $G_{i+1}$. In Case 2 of Theorem 1, the partition sequence $u^{(j_{i+1}-1)}(G_{i+1})$ is the same as $u^{(j_i-1)}(G_i)$ except the last symbol; in $u^{(j_{i+1}-1)}(G_{i+1})$, the last symbol is equal to the newly introduced variable $s_{j_{i+1}-1}$ which appears in $u^{(j_{i+1}-1)}(G_{i+1})$ only in the last position. (For example, in Example 6, $G_{10}$ induces a partition sequence $10011 s_1 0 s_1$ while $G_{11}$ induces a partition sequence $10011 s_1 0 s_2$.) Therefore, in this case, there is no repeated pattern of length 3 in $u^{(j_{i+1}-1)}(G_{i+1})$ and hence Lemma 2 is true for $G_{i+1}$. In Case 1 of Theorem 1, the partition sequence $u^{(j_{i+1}-1)}(G_{i+1})$ is obtained by appending the last symbol in $G_{i+1}(s_0)$ to the end of $u^{(j_i-1)}(G_i)$. To show that there is no repeated pattern of length 3 in $u^{(j_{i+1}-1)}(G_{i+1})$ in this case, we distinguish between several subcases. We need to look at how $G_i$ and $G_{i-1}$ are constructed from $G_{i-1}$ and $G_{i-2}$, respectively. If $I(i) = 1$, then the last symbol in $u^{(j_i-1)}(G_i)$ is equal to $s_{j_i-1}$ which appears only once in $u^{(j_i-1)}(G_i)$. Clearly, in this case, there is no repeated pattern of length 3 in $u^{(j_{i+1}-1)}(G_{i+1})$. If $I(i) = 0$ and $I(i-1) = 1$, then $u^{(j_i-1)}(G_i)$ is obtained by appending the last symbol in $G_i(s_0)$ to the end of $u^{(j_{i-1}-1)}(G_{i-1})$ and the last symbol in $u^{(j_{i-1}-1)}(G_{i-1})$ is equal to $s_{j_{i-1}-1}$ which appears only once in $u^{(j_{i-1}-1)}(G_{i-1})$. Once again, in this case, there is no repeated pattern of length 3 in $u^{(j_{i+1}-1)}(G_{i+1})$ since $u^{(j_{i+1}-1)}(G_{i+1})$ is obtained by appending the last symbol in $G_{i+1}(s_0)$ to the end of $u^{(j_i-1)}(G_i)$. The only case left is the case in which $I(i) = 0$ and $I(i-1) = 0$. It is easy to see that in this case, $G_{i+1}$ is obtained from $G_{i-2}$ by appending the three recent parsed symbols to the end of $G_{i-2}(s_0)$, and $u^{(j_{i+1}-1)}(G_{i+1})$ is obtained by appending the last three symbols in $G_{i+1}(s_0)$ to the end of $u^{(j_{i-2}-1)}(G_{i-2})$. Let $\beta_1\beta_2\beta_3\beta_4$ be the last four symbols in $u^{(j_{i+1}-1)}(G_{i+1})$. Clearly, $\beta_2\beta_3\beta_4$ is the only possible repeated pattern of length 3 in $u^{(j_{i+1}-1)}(G_{i+1})$. Note that for any irreducible grammar $G$, the last symbol in the partition sequence induced by $G$ is the same as the last symbol in $G(s_0)$. Thus $\beta_1$ is also the last symbol in $G_{i-2}(s_0)$ and hence $\beta_1\beta_2\beta_3\beta_4$ yields the last four symbols in $G_{i+1}(s_0)$. From this, it follows that $\beta_2\beta_3\beta_4$ cannot repeat itself in a overlapping position in $u^{(j_{i+1}-1)}(G_{i+1})$ since $G_{i+1}$ is irreducible. On the other hand, for any substring $\alpha_1\alpha_2\alpha_3$ of length 3 of $u^{(j_{i-2}-1)}(G_{i-2})$, either $\alpha_1\alpha_2$ or $\alpha_2\alpha_3$ appears in $G_{i-2}(s_j)$ for some $s_j \in \mathcal{S}(j_{i-2})$. Since $G_{i+1}$ is obtained from $G_{i-2}$ by appending $\beta_2\beta_3\beta_4$ to the end of $G_{i-2}(s_0)$ and $G_{i+1}$ is irreducible, it follows that $\beta_2\beta_3\beta_4$ cannot repeat itself in $u^{(j_{i-2}-1)}(G_{i-2})$. Thus there is no repeated pattern of length 3 in $u^{(j_{i+1}-1)}(G_{i+1})$. This completes the proof of Lemma 2.

*Remark 7:* From the proof of Lemma 2, it follows that with the help of our irreducible grammar transform, the partition sequence $u^{(j_t-1)}(G_t)$ can be constructed sequentially in one pass.

Lemma 2 enables us to establish useful upper bounds on the size of the final irreducible grammar $G_t$ and the length of the induced partition sequence in terms of the length $|x|$. These bounds are stated in Lemma 3 and will be proved in Appendix A.

*Lemma 3:* Let $x$ be a sequence from $\mathcal{A}$. Let $G_t$ be the final irreducible grammar with variable set $\mathcal{S}(j_t)$ resulting from applying our irreducible grammar transform to $x$. Let $u^{(j_t-1)}(G_t)$ be the partition sequence induced by $G_t$. Then

$$\frac{\left|u^{(j_t-1)}(G_t)\right|}{|x|} \le \frac{3\log|\mathcal{A}|}{\log|x|-3\log\log|x|-2-8\log|\mathcal{A}|}$$

and

$$\frac{|G_t|}{|x|} \le \frac{6\log|\mathcal{A}|}{\log|x|-3\log\log|x|-2-8\log|\mathcal{A}|} \quad (5.1)$$

whenever

$$|x| \ge \sum_{n=3}^{7} n\binom{n-1}{2}|\mathcal{A}|^n$$

and $\log|x|-3\log\log|x|-2-8\log|\mathcal{A}|>0$, where log stands for the logarithm with base 2.

The following lemma, which will be proved in Appendix B, gives a lower bound to the average length of the $\mathcal{A}$-sequences represented by $s_i \in \mathcal{S}(j_t)$, $i=1,\cdots,j_t-1$.

*Lemma 4:* Let $x$ be a sequence from $\mathcal{A}$. Let $G_t$ be the final irreducible grammar with variable set $\mathcal{S}(j_t)$ resulting from applying our irreducible grammar transform to $x$. Then

$$\frac{1}{j_t-1}\sum_{i=1}^{j_t-1}|s_i| \ge \frac{1}{3\log|\mathcal{A}|}\left[\log|x|-3\log\log|x|-3-9\log|\mathcal{A}|-\log 3\right]$$

whenever $|x| \ge 3(|\mathcal{A}|+|\mathcal{A}|^2)^3$, where $|s_i|$ denotes the length of the $\mathcal{A}$-sequence represented by $s_i \in \mathcal{S}(j_t)$.

We are now in position to evaluate the compression performance of the hierarchical data compression algorithm. We compare the compression performance of the hierarchical algorithm with that of the best arithmetic coding algorithm with $k$ contexts which operates letter by letter, rather than phrase by phrase. Let $\mathcal{C}$ be a finite set consisting of $k$ elements; each element $C \in \mathcal{C}$ is regarded as an abstract context. Let $p$: $\mathcal{C}\times(\mathcal{A}\times\mathcal{C}) \to [0,1]$ be a transition probability function from $\mathcal{C}$ to $\mathcal{A}\times\mathcal{C}$, i.e., $p$ satisfies

$$\sum_{a\in\mathcal{A},\, C\in\mathcal{C}} p(a, C|C') = 1$$

for any $C' \in \mathcal{C}$. Note that random transitions between contexts are allowed. For any sequence $x = x_1x_2\cdots x_n$ from $\mathcal{A}$, the compression rate in bits per letter resulting from using the arithmetic coding algorithm with transition probability function $p$ to encode $x$ is given by

$$-\frac{1}{n}\log \sum_{C_1,\cdots,C_n\in\mathcal{C}} \prod_{i=1}^{n} p(x_i, C_i|C_{i-1})$$

where $C_0 \in \mathcal{C}$ is the initial context, and log stands for the logarithm with base 2 throughout this paper. Let

$$r_k^*(x) \triangleq -\frac{1}{n}\log \max_p \max_{C_0\in\mathcal{C}} \sum_{C_1,\cdots,C_n\in\mathcal{C}} \prod_{i=1}^{n} p(x_i, C_i|C_{i-1}) \quad (5.2)$$

where the outer maximization varies over every transition probability function $p$ from $\mathcal{C}$ to $\mathcal{A}\times\mathcal{C}$. The quantity $r_k^*(x)$ represents the smallest compression rate in bits per letter among all arithmetic coding algorithms with $k$ contexts which operate letter by letter. It should be, however, emphasized that there is no single arithmetic coding algorithm with $k$ contexts which can achieve the compression rate $r_k^*(x)$ for *every* sequence $x = x_1x_2\cdots x_n$. When $k=1$, $r_k^*(x)$ is equal to the traditional empirical entropy of $x$. For this reason, we call $r_k^*(x)$ the *$k$-context empirical entropy* of $x$.

Let $r^h(x)$ be the compression rate in bits per letter resulting from using the hierarchical compression algorithm to compress $x$. We are interested in the difference between $r^h(x)$ and $r_k^*(x)$. Let

$$R_{n,k}^h \triangleq \max_{x\in\mathcal{A}^n}\left[r^h(x)-r_k^*(x)\right].$$

The quantity $R_{n,k}^h$ is called the *worst case redundancy* of the hierarchical algorithm against the $k$-context empirical entropy.

*Theorem 2:* There is a constant $d_k$, which depends only on $|\mathcal{A}|$ and $k$, such that

$$R_{n,k}^h \le d_k\frac{\log\log n}{\log n}.$$

*Remark 8:* Worst case redundancy is a rather strong notion of redundancy. For probabilistic sources, there are two other notions of redundancy: average redundancy [8] and pointwise redundancy [21]. It is expected that the average and pointwise redundancies of the hierarchical, sequential, and improved sequential algorithms are much smaller. The exact formulas of these redundancies, however, are unknown at this point, and left open for future research.

*Proof of Theorem 2:* Let $x \in \mathcal{A}^n$ be a sequence to be compressed. Let $G_t$ be the final irreducible grammar with variable set $\mathcal{S}(j_t)$ resulting from applying our irreducible grammar transform to $x$. Let $u^{(j_t-1)}(G_t) = u_1u_2\cdots u_m$ be the partition sequence induced by $G_t$. Recall that the hierarchical compression algorithm compresses $x$ by first converting $G_t$ into its canonical form $G_t^g$, then constructing the sequence generated from $G_t$, and finally using a zero-order arithmetic code with a dynamic alphabet to encode the generated sequence. In the process of converting $G_t$ into its canonical form $G_t^g$, one gets a permutation $\pi$ over $\mathcal{A}\cup\mathcal{S}(j_t)$ such that $G_t^g$ is obtained from $G_t$ by renaming each symbol $\beta$ as $\pi(\beta)$. For example, for the final irreducible grammar $G_{18}$ in Example 6, the permutation $\pi$ is given by $\pi(s_3)=s_2$, $\pi(s_2)=s_3$, and $\pi(\beta)=\beta$ for any other symbol $\beta$. Let $v = v_1v_2\cdots v_l$ be the sequence generated from $G_t$. Note that $v$ is from $\mathcal{S}(j_t)\cup\mathcal{A}\cup\{b, e, s\}$. Strike out symbols $s$, $b$, and $e$ in $v$. The resulting sequence is called the *content sequence* generated from $G_t$ and denoted by $v'$. (For example, the content sequence generated from $G_{18}$ in Example 6 is

$s_2s_4s_2100s_4s_3s_1011$.) It is easy to see that the content sequence $v'$ and the partition sequence $u^{(j_t-1)}(G_t)$ have the same length $|G_t| - j_t + 1$. Furthermore, for each symbol $\beta \in \mathcal{S}(j_t) \cup \mathcal{A}$, the frequency of $\beta$ in $u^{(j_t-1)}(G_t)$ is the same as that of $\pi(\beta)$ in $v'$. Thus $u^{(j_t-1)}(G_t)$ and $v'$ have the same first-order unnormalized empirical entropy, that is,

$$H\left(u^{(j_t-1)}(G_t)\right) = H(v') \tag{5.3}$$

where $H(u^{(j_t-1)}(G_t))$ is defined as

$$H\left(u^{(j_t-1)}(G_t)\right) \triangleq \sum_{\beta\in\mathcal{S}(j_t)\cup\mathcal{A}} |\{i\colon u_i = \beta,\, 1 \le i \le m\}| \cdot \log\left(\frac{\left|u^{(j_t-1)}(G_t)\right|}{|\{i\colon u_i = \beta,\, 1 \le i \le m\}|}\right)$$

and $H(v')$ is defined similarly. Below we will upper–bound the total number of bits $nr^h(x)$ in terms of $H(v')$.

Assume that exact arithmetic is used. In view of the encoding process of the hierarchical algorithm, the probability used to encode the symbol $v_i$ in $v$ is

$$\frac{c(v_i)}{|\mathcal{A}| + 2 + i + c(s)}$$

where $c(v_i)$ is the number of $v_i$ in the prefix $v_1v_2\cdots v_i$ and $c(s)$ is the number of $s$ in the prefix $v_1v_2\cdots v_i$. Thus the number of bits needed to encode $v_i$ is

$$\log\left(\frac{|\mathcal{A}| + 2 + i + c(s)}{c(v_i)}\right) \le \log\left(\frac{|\mathcal{A}| + 1 + i + j_t}{c(v_i)}\right).$$

The above inequality is due to the fact that $c(s) \le j_t - 1$ for all positions $i$. This implies that the total number of bits $nr^h(x)$ is upper-bounded by

$$\begin{aligned} nr^h(x) &\le \log\left(\frac{\prod_{i=1}^{l}(|\mathcal{A}| + 1 + j_t + i)}{\prod_{\beta\in\mathcal{S}(j_t)\cup\mathcal{A}\cup\{s,b,e\}}(c_\beta)!}\right) \\ &= \log\left(\frac{l!}{\prod_{\beta\in\mathcal{S}(j_t)\cup\mathcal{A}\cup\{s,b,e\}}(c_\beta)!}\right) \\ &\quad + \log\left(\binom{l + |\mathcal{A}| + 1 + j_t}{|\mathcal{A}| + 1 + j_t}\right) \\ &\overset{1)}{\le} H(v) + l + |\mathcal{A}| + 1 + j_t \\ &\overset{2)}{=} lH\left(\frac{c_s}{l}, \frac{c_b}{l}, \frac{c_e}{l}, \frac{l - c_s - c_b - c_e}{l}\right) \\ &\quad + H(v') + l + |\mathcal{A}| + 1 + j_t \\ &\le 3l + |\mathcal{A}| + 1 + j_t + H(v') \\ &\overset{3)}{\le} 6|G_t| + |\mathcal{A}| + 1 + j_t + H(v') \\ &\overset{4)}{=} 6|G_t| + |\mathcal{A}| + 1 + j_t + H(u^{(j_t-1)}(G_t)). \end{aligned} \tag{5.4}$$

In the above, $c_\beta$ denotes, for each $\beta \in \mathcal{S}(j_t) \cup \mathcal{A} \cup \{s, b, e\}$, the number of $\beta$ in $v$, $H(v)$ denotes the first-order unnormalized empirical entropy of $v$, and

$$H\left(\frac{c_s}{l}, \frac{c_b}{l}, \frac{c_e}{l}, \frac{l - c_s - c_b - c_e}{l}\right)$$

denotes the Shannon entropy of the distribution

$$(c_s/l,\, c_b/l,\, c_e/l,\, (l - c_s - c_b - c_e)/l).$$

The inequality 1) is due to the well-known inequality on the size of a type class [7, Theorem 12.1.3, p. 282]. The equality 2) follows from the entropy identity saying that the joint entropy of two random variables is equal to the marginal entropy of the first random variable plus the conditional entropy of the second random variable given the first random variable. The inequality 3) follows from the fact that

$$l \le |G_t| + 1 + 2(j_t - 1) \le 2|G_t|.$$

Finally, the equality 4) follows from (5.3).

To complete the proof, we next upper-bound $H(u^{(j_t-1)}(G_t))$ in terms of $r_k^*(x)$. To this end, let $p$ be a transition probability function from $\mathcal{C}$ to $\mathcal{A} \times \mathcal{C}$ for which the maximum on the right-hand side of (5.2) is achieved. Note that such $p$ exists and generally depends on the sequence $x$ to be compressed. Let $p^*$ be the probability distribution on $\mathcal{A}^+$ such that for any positive integer $r$ and any $y = y_1\cdots y_r \in \mathcal{A}^r$

$$p^*(y) = Q_k k^{-1} r^{-2} \max_{C_0\in\mathcal{C}} \sum_{C_1,\cdots,C_r\in\mathcal{C}} \prod_{i=1}^{r} p(y_i, C_i|C_{i-1}). \tag{5.5}$$

In (5.5), the constant $Q_k$ is selected so that $p^*$ is a probability distribution on $\mathcal{A}^+$; it is easy to check that $Q_k \ge 1/2$. Recall that the partition sequence $u^{(j_t-1)}(G_t)$ partitions $x$ into nonoverlapping, variable-length phrases; each symbol in $u^{(j_t-1)}(G_t)$ represents a substring of $x$, and the concatenation of all these substrings is equal to $x$. Think of each symbol $\beta \in \mathcal{S}(j_t) \cup \mathcal{A}$ as a sequence from $\mathcal{A}$. Then it makes sense to write $p^*(\beta)$ for any $\beta \in \mathcal{S}(j_t) \cup \mathcal{A}$. From (5.2) and (5.5), it then follows that

$$\begin{aligned} &\sum_{i=1}^{m} -\log p^*(u_i) \\ &\qquad \le nr_k^*(x) - m\log Q_k + m\log k + 2\sum_{i=1}^{m}\log|u_i| \end{aligned} \tag{5.6}$$

where $|u_i|$ denotes the length of the $\mathcal{A}$-sequence represented by $u_i$. In view of the information inequality [7, Theorem 2.6.3, p. 26]

$$H\left(u^{(j_t-1)}(G_t)\right) \le \sum_{i=1}^{m} -\log p^*(u_i)$$

which, together with (5.4) and (5.6), implies

$$\begin{aligned} nr^h(x) \le\, &6|G_t| + |\mathcal{A}| + 1 + j_t + nr_k^*(x) - m\log Q_k \\ &+ m\log k + 2\sum_{i=1}^{m}\log|u_i| \end{aligned}$$

$$\begin{aligned}&\overset{1)}{\le} 7|G_t| + |\mathcal{A}| + 2 + nr_k^*(x) + m\log k \\ &\quad + 2\sum_{i=1}^{m} \log|u_i| \\ &\overset{2)}{\le} 7|G_t| + |\mathcal{A}| + 2 + nr_k^*(x) + m\log k \\ &\quad + 2m\log\left(\frac{n}{m}\right). \end{aligned} \tag{5.7}$$

In the above, the inequality 1) is due to Lemma 1 and the fact that $Q_k \ge 1/2$. The inequality 2) is attributable to the concavity of the logarithm function. Note that (5.7) holds for any sequence $x \in \mathcal{A}^n$. Dividing both sides of (5.7) by $n$ and applying Lemma 3, we then get

$$R_{n,k}^h \le O\left(\frac{1}{n}\right) + O\left(\frac{1}{\log n}\right) + O\left(\frac{\log\log n}{\log n}\right).$$

This completes the proof of Theorem 2.

*Corollary 1:* For any stationary, ergodic source $X = \{X_i\}_{i=1}^{\infty}$ with alphabet $\mathcal{A}$

$$r^h(X_1X_2\cdots X_n) \to H(X)$$

with probability one as $n \to \infty$, where $H(X)$ is equal to the entropy rate of $X$.

*Proof:* Let $k = |\mathcal{A}|^q$. For any $\mathcal{A}$-sequence $y$ with length $|y| < n$, let $f(y|x^n)$ be the frequency of $y$ in $x^n = x_1\cdots x_n \in \mathcal{A}^n$, computed in a cyclic manner

$$f(y|x^n) = \frac{|\{1 \le i \le n\colon x_ix_{i+1}\cdots x_{i+|y|-1} = y\}|}{n}$$

where, as a convention, $x_i = x_j$ whenever $i = j \pmod n$. Consider the $q$th-order traditional empirical entropy defined by

$$H_q = \sum_{y\in\mathcal{A}^q}\sum_{a\in\mathcal{A}} f(ya|X^n)\log\left(\frac{f(y|X^n)}{f(ya|X^n)}\right)$$

where $X^n = X_1X_2\cdots X_n$. It is easy to verify that

$$r_k^*(X^n) \le H_q.$$

Thus from Theorem 2

$$r^h(X^n) \le H_q + d\frac{\log\log n}{\log n}.$$

Letting $n \to \infty$ and invoking the ergodic theorem, we get

$$\lim_{n\to\infty} H_q = H(X_{q+1}|X_1\cdots X_q) \quad \text{almost surely}$$

and

$$\limsup_{n\to\infty} r^h(X^n) \le H(X_{q+1}|X_1\cdots X_q) \quad \text{almost surely.}$$

In the above inequality, letting $q \to \infty$ yields

$$\limsup_{n\to\infty} r^h(X^n) \le H(X) \quad \text{almost surely.}$$

This, together with sample converses in source coding theory [2], [11], [34], implies

$$r^h(X^n) \to H(X) \quad \text{almost surely.}$$

## VI. Performance of the Sequential and Improved Sequential Algorithms

In this section, we analyze the performance of the sequential and improved sequential compression algorithms and provide some insights into their workings. We take an approach similar to that of Section V.

Let $x \in \mathcal{A}^+$ be a sequence to be compressed. Let $G_t$ be the final irreducible grammar with variable set $\mathcal{S}(j_t)$ furnished by the proposed irreducible grammar transform. Recall that $t$ is the number of phrases parsed off by the proposed irreducible grammar transform. The next lemma, which will be proved in Appendix C, upper-bounds $t$ in terms of a function of $|x|$.

*Lemma 5:* There is a constant $d_1$, which depends only on $|\mathcal{A}|$, such that for any $x \in \mathcal{A}^+$ with $|x| \ge 3$,

$$t \le d_1\frac{|x|}{\log|x|}.$$

Lemma 5 enables us to evaluate the compression performance of the sequential and improved sequential compression algorithms. Let $x = x_1x_2\cdots x_n$ be a sequence from $\mathcal{A}$ to be compressed. Let $r^s(x)$ be the compression rate in bits per letter resulting from using the sequential algorithm to compress $x$. Let $r_k^*(x)$ be defined as in Section V. We are interested in the difference between $r^s(x)$ and $r_k^*(x)$. Let

$$R_{n,k}^s \triangleq \max_{x\in\mathcal{A}^n}\left[r^s(x) - r_k^*(x)\right].$$

The quantity $R_{n,k}^s$ is called the worst case redundancy of the sequential algorithm against the $k$-context empirical entropy. Using a similar argument to the proof of Theorem 2, one can show the following theorem.

*Theorem 3:* There is a constant $d_k'$, which depends only on $|\mathcal{A}|$ and $k$, such that

$$R_{n,k}^s \le d_k'\frac{\log\log n}{\log n}.$$

*Proof:* In the sequential algorithm, we encode the data sequence $x \in \mathcal{A}^n$ sequentially by using a zero-order arithmetic code with a dynamic alphabet to encode the sequence of parsed phrases $x_1$, $x_2\cdots x_{n_2}$, $\cdots$, $x_{n_{t-1}+1}\cdots x_{n_t}$. Assume that exact arithmetic is used. The probability used to encode the $(i+1)$th parsed phrase, which is represented by a symbol $\beta \in \mathcal{S}(j_t) \cup \mathcal{A}$, is

$$\frac{c(\beta)}{|\mathcal{A}| + i + j_i - 1}$$

where $c(\beta)$ is the number of times the phrase $x_{n_i+1}\cdots x_{n_{i+1}}$ appears in $\{x_1, x_2\cdots x_{n_2}, \cdots, x_{n_i+1}\cdots x_{n_{i+1}}\}$. Thus the number of bits needed to encode the $(i+1)$th parsed phrase $\beta$ is

$$\log\left(\frac{|\mathcal{A}| + i + j_i - 1}{c(\beta)}\right) \le \log\left(\frac{|\mathcal{A}| + i + j_t - 1}{c(\beta)}\right).$$

This implies that the total number of bits $nr^s(x)$ is upper-bounded by

$$\begin{aligned} nr^s(x) &\le \log\left(\frac{\prod_{i=1}^{t}(|\mathcal{A}|+i+j_t-2)}{\prod_{\beta\in\mathcal{S}(j_t)\cup\mathcal{A}}(c_\beta)!}\right) \\ &= \log\left(\frac{t!}{\prod_{\beta\in\mathcal{S}(j_t)\cup\mathcal{A}}(c_\beta)!}\right) \\ &\quad + \log\left(\binom{t+|\mathcal{A}|+j_t-2}{|\mathcal{A}|+j_t-2}\right) \\ &\le H_p(x)+t+|\mathcal{A}|+j_t-2. \end{aligned} \tag{6.1}$$

In the above derivation, $c_\beta$, for each $\beta \in \mathcal{S}(j_t)\cup\mathcal{A}$, denotes the number of times the $\mathcal{A}$-sequence represented by $\beta$ appears in the sequence of parsed phrases

$$x_1,\, x_2\cdots x_{n_2},\, \cdots,\, x_{n_{t-1}+1}\cdots x_{n_t}.$$

The quantity $H_p(x)$ denotes the unnormalized empirical entropy of the sequence of parsed phrases, i.e.,

$$H_p(x) = \sum_{\beta\in\mathcal{S}(j_t)\cup\mathcal{A}} c_\beta \log\left(\frac{t}{c_\beta}\right).$$

A similar argument to the derivation of (5.6) and (5.7) can then lead to

$$H_p(x) \le nr_k^*(x) + t + t\log k + 2t\log\left(\frac{n}{t}\right)$$

which, coupled with (6.1), implies

$$nr^s(x) \le nr_k^*(x) + 3t + |\mathcal{A}| + t\log k + 2t\log\left(\frac{n}{t}\right). \tag{6.2}$$

Dividing both sides of (6.2) by $n$ and applying Lemma 5, we get

$$R_{n,k}^s \le O\left(\frac{\log\log n}{\log n}\right).$$

This completes the proof of Theorem 3.

*Corollary 2:* For any stationary, ergodic source $X=\{X_i\}_{i=1}^{\infty}$ with alphabet $\mathcal{A}$

$$r^s(X_1X_2\cdots X_n) \to H(X)$$

with probability one as $n\to\infty$, where $H(X)$ is equal to the entropy rate of $X$.

*Proof:* It follows immediately from Theorem 3 and the proof of Corollary 1.

For any $\mathcal{A}$-sequence $x=x_1x_2\cdots x_n$, let $r^{is}(x)$ be the compression rate in bits per letter resulting from using the improved sequential algorithm to compress $x$. Let

$$R_{n,k}^{is} \triangleq \max_{x\in\mathcal{A}^n}[r^{is}(x) - r_k^*(x)].$$

The quantity $R_{n,k}^{is}$ is called the worst case redundancy of the improved sequential algorithm against the $k$-context empirical entropy. Using similar arguments to the proofs of Theorems 2 and 3, one can show the following theorem.

*Theorem 4:* There is a constant $\hat{d}_k$, which depends only on $|\mathcal{A}|$ and $k$, such that

$$R_{n,k}^{is} \le \hat{d}_k \frac{\log\log n}{\log n}.$$

The following corollary follows immediately from Theorem 4 and the proof of Corollary 1.

*Corollary 3:* For any stationary, ergodic source $X=\{X_i\}_{i=1}^{\infty}$ with alphabet $\mathcal{A}$

$$r^{is}(X_1X_2\cdots X_n) \to H(X)$$

with probability one as $n\to\infty$, where $H(X)$ is equal to the entropy rate of $X$.

## VII. Simulation Results

To keep the information-theoretic flavor of this paper, this section presents only simulation results on random binary sequences. For extensive simulation results on other types of practical data, see [31].

Before presenting our simulation results, let us say a few words about the computational complexity of our compression algorithms. Let $x\in\mathcal{A}^+$ be a sequence to be compressed. From Section III, it follows that our compression algorithms have only three major operations: the parsing of $x$ into nonoverlapping substrings $\{x_1, x_2\cdots x_{n_2}, \cdots, x_{n_{t-1}+1}\cdots x_{n_t}\}$the updating of $G_i$ into $G_{i+1}$, $i = 1, 2, \cdots, t-1$, and the encoding either of $G_t$ or of the parsed substrings $\{x_1, x_2\cdots x_{n_2}, \cdots, x_{n_{t-1}+1}\cdots x_{n_t}\}$. In view of Lemmas 3 and 5, it is easy to see that the encoding operation has linear computational complexity with respect to the length $|x|$. By virtue of Lemmas 4 and 5, one can show that the average computational complexity of the parsing operation is linear with respect to $|x|$ if $x$ is drawn from a stationary source satisfying some mixing condition. To update $G_i$ into $G_{i+1}$, it follows from Theorem 1 that at most two reduction rules are involved. Therefore, the major computational complexity of the updating operation lies in finding the location at which these reduction rules are applied. Let $\alpha$ be the last symbol in $G_i(s_0)$ and let $\beta$ be the symbol representing the $(i+1)$th parsed phrase. As demonstrated in the proof of Theorem 1, $\alpha\beta$ is the only possible nonoverlapping repeated pattern of length $\ge 2$ in the appended $G_i$, and repeats itself at most once in the range of the appended $G_i$. Since $G_i$ is irreducible, one can show, by using a proper tree structure, that the total computational complexity of finding the repetition locations for all $i=1, 2, \cdots, t-1$ is linear. Hence the updating operation also has linear computational complexity with respect to $|x|$. Therefore, our compression algorithms, the hierarchical algorithm, sequential algorithm, and improved sequential algorithm, all have linear average computational complexity with respect to $|x|$. In passing, our compression algorithms are also linear in space. The argument just completed is rather brief; the implementation details of our compression algorithms, their complexity analysis, and extensive simulation results will be reported in [31]. (Experimental results [31] show that for a variety of files, the improved sequential algorithm significantly

TABLE I
RESULTS FOR MEMORYLESS BINARY SOURCES OF LENGTH 10000

| $q$ | Shannon entropy | $\|G_t\|$ | $t$ | $j_t-1$ | $r^{is}(x)$ | $r^s(x)$ | $r^h(x)$ | Unix Compress | Gzip |
|---|---|---|---|---|---|---|---|---|---|
| 0.6 | 0.9710 | 1809 | 1815 | 244 | 1.1165 | 1.1278 | 1.2041 | 1.3032 | 1.3496 |
| 0.7 | 0.8813 | 1667 | 1676 | 238 | 1.0314 | 1.0436 | 1.1076 | 1.2080 | 1.2920 |
| 0.8 | 0.7219 | 1427 | 1434 | 207 | 0.8428 | 0.8564 | 0.9206 | 1.0400 | 1.1312 |
| 0.9 | 0.4690 | 996 | 1001 | 149 | 0.5564 | 0.5645 | 0.6142 | 0.7312 | 0.8128 |

Entropy and rates are expressed in terms of bits per letter.

TABLE II
RESULTS FOR FIRST-ORDER MARKOV BINARY SOURCES OF LENGTH 10000

| $q$ | Shannon entropy | $\|G_t\|$ | $t$ | $j_t-1$ | $r^{is}(x)$ | $r^s(x)$ | $r^h(x)$ | Unix Compress | Gzip |
|---|---|---|---|---|---|---|---|---|---|
| 0.6 | 0.9710 | 1833 | 1841 | 253 | 1.1574 | 1.1692 | 1.2318 | 1.3072 | 1.3456 |
| 0.7 | 0.8813 | 1665 | 1671 | 214 | 1.0337 | 1.0445 | 1.1048 | 1.2072 | 1.2576 |
| 0.8 | 0.7219 | 1475 | 1483 | 205 | 0.9075 | 0.9177 | 0.9687 | 1.0440 | 1.1016 |
| 0.9 | 0.4690 | 1043 | 1050 | 159 | 0.6248 | 0.6327 | 0.6643 | 0.7360 | 0.7656 |

Entropy and rates are expressed in terms of bits per letter.

TABLE III
RESULTS FOR SECOND-ORDER MARKOV BINARY SOURCES OF LENGTH 10000

| $q$ | Shannon entropy | $\|G_t\|$ | $t$ | $j_t-1$ | $r^{is}(s)$ | $r^s(x)$ | $r^h(x)$ | Unix Compress | Gzip |
|---|---|---|---|---|---|---|---|---|---|
| 0.6 | 0.9710 | 1835 | 1843 | 247 | 1.1526 | 1.1648 | 1.2335 | 1.3128 | 1.3432 |
| 0.7 | 0.8813 | 1672 | 1681 | 235 | 1.0331 | 1.0436 | 1.1035 | 1.2280 | 1.2688 |
| 0.8 | 0.7219 | 1459 | 1466 | 201 | 0.8864 | 0.8963 | 0.9512 | 1.0752 | 1.1064 |
| 0.9 | 0.4690 | 1041 | 1045 | 153 | 0.6159 | 0.6215 | 0.6562 | 0.7976 | 0.7696 |

Entropy and rates are expressed in terms of bits per letter.

outperforms the Unix Compress and Gzip algorithms. For example, for some binary files with alphabet $\{0, 1, \cdots, 255\}$, the improved sequential algorithm is 255% better than the Gzip algorithm and 447.9% better than the Unix Compress algorithm. Moreover, unlike previous compression algorithms, the improved sequential algorithm can also compress short data sequences like packets moved around networks by the Internet Protocol very well.)

To see how close the compression rates given by our algorithms are to the entropy rate of a random source, we present below some simulation results for random binary sequences. In our simulation, our algorithms, like the Unix Compress and Gzip algorithms, were implemented to compress any files.

Table I lists some simulation results for memoryless binary sources of length 10000. The quantity $q$ represents the probability of symbol 1; the Shannon entropy represents the entropy rate of each binary source. The notation $|G_t|$ denotes the size of the final irreducible grammar; $t$ is the number of nonoverlapping phrases parsed off by our irreducible grammar transform; and $j_t - 1$ is the number of distinct phrases. From Table I, one can see that our algorithms are all better than the Unix Compress and Gzip algorithms. For example, on average, the improved sequential algorithm is roughly 26% more efficient than Unix Compress and 37% more efficient than Gzip. (It should be pointed out that for text files, Gzip often outperforms Unix Compress. On the other hand, for binary sequences, Unix Compress often outperforms Gzip.) Here, the efficiency of a data compression algorithm is defined as the ratio of the compression rate of the algorithm to the Shannon entropy rate of the source. Also, the number $t$ is only slightly larger than $|G_t|$; this means that the length of most $G_t(s_j)$ is 2.

Table II lists some simulation results for first-order Markov binary sources of length 10000. The transition matrix of each Markov source is

$$\begin{bmatrix} q & 1-q \\ 1-q & q \end{bmatrix}$$

and the initial distribution is uniform. Once again, our algorithms are all better than the Unix Compress and Gzip algorithms. In this case, the improved sequential algorithm is, on average, roughly 19% more efficient than Unix Compress and 25% more efficient than Gzip.

Table III lists some simulation results for second-order Markov binary sources of length 10000. The second-order Markov binary sources are generated by using the following model:

$$X_i = X_{i-1} \oplus X_{i-2} \oplus Y_i$$

where $\{Y_i\}$ is an independent and identically distributed (i.i.d.) sequence with the probability of symbol 1 being $q$, and $\oplus$ denotes modulo-2 addition. Once again, our algorithms are all better than the Unix Compress and Gzip algorithms. In this case, the improved sequential algorithm is, on average, roughly 26% more efficient than Unix Compress and 27% more efficient than Gzip.

Similar phenomena hold as well for sources of length 65536. Tables IV–VI list some simulation results for memoryless,

TABLE IV
RESULTS FOR MEMORYLESS BINARY SOURCES OF LENGTH 65536

| $q$ | Shannon entropy | $\|G_t\|$ | $t$ | $j_t-1$ | $r^{is}(x)$ | $r^{s}(x)$ | $r^{h}(x)$ | Unix Compress | Gzip |
|---|---|---|---|---|---|---|---|---|---|
| 0.6 | 0.9710 | 9028 | 9038 | 993 | 1.0634 | 1.0705 | 1.1203 | 1.1694 | 1.2858 |
| 0.7 | 0.8813 | 8307 | 8323 | 938 | 0.9713 | 0.9781 | 1.0227 | 1.0747 | 1.2291 |
| 0.8 | 0.7219 | 6977 | 6987 | 819 | 0.7916 | 0.7974 | 0.8385 | 0.9000 | 1.0712 |
| 0.9 | 0.4690 | 4764 | 4773 | 578 | 0.5222 | 0.5266 | 0.5520 | 0.6111 | 0.7609 |

Entropy and rates are expressed in terms of bits per letter.

TABLE V
RESULTS FOR FIRST–ORDER MARKOV BINARY SOURCES OF LENGTH 65536

| $q$ | Shannon entropy | $\|G_t\|$ | $t$ | $j_t-1$ | $r^{is}(x)$ | $r^{s}(x)$ | $r^{h}(x)$ | Unix Compress | Gzip |
|---|---|---|---|---|---|---|---|---|---|
| 0.6 | 0.9710 | 9137 | 9152 | 1056 | 1.0862 | 1.0934 | 1.1394 | 1.1720 | 1.2788 |
| 0.7 | 0.8813 | 8392 | 8406 | 916 | 1.0000 | 1.0068 | 1.0437 | 1.0800 | 1.2087 |
| 0.8 | 0.7219 | 7123 | 7131 | 809 | 0.8334 | 0.8380 | 0.8706 | 0.9089 | 1.0383 |
| 0.9 | 0.4690 | 5099 | 5116 | 596 | 0.5905 | 0.5942 | 0.6112 | 0.6309 | 0.7219 |

Entropy and rates are expressed in terms of bits per letter.

TABLE VI
RESULTS FOR SECOND-ORDER MARKOV BINARY SOURCES OF LENGTH 65536

| $q$ | Shannon entropy | $\|G_t\|$ | $t$ | $j_t-1$ | $r^{is}(x)$ | $r^{s}(x)$ | $r^{h}(x)$ | Unix Compress | Gzip |
|---|---|---|---|---|---|---|---|---|---|
| 0.6 | 0.9710 | 9059 | 9073 | 986 | 1.0803 | 1.0871 | 1.1308 | 1.1771 | 1.2745 |
| 0.7 | 0.8813 | 8356 | 8371 | 906 | 0.9932 | 1.0001 | 1.0393 | 1.0927 | 1.2065 |
| 0.8 | 0.7219 | 7163 | 7171 | 810 | 0.8432 | 0.8491 | 0.8798 | 0.9387 | 1.0464 |
| 0.9 | 0.4690 | 5139 | 5159 | 622 | 0.5932 | 0.5983 | 0.6143 | 0.6699 | 0.7344 |

Entropy and rates are expressed in terms of bits per letter.

first-order Markov, and second-order Markov binary sources of length 65536.

## VIII. CONCLUSIONS

Within the design framework of grammar-based codes, we have presented a greedy irreducible grammar transform that constructs sequentially a sequence of irreducible context-free grammars from which the original data sequence can be recovered incrementally. Based on this grammar transform, we have developed three efficient universal lossless compression algorithms: the hierarchical algorithm, sequential algorithm, and improved sequential algorithm. These algorithms combine the power of arithmetic coding with that of string matching in a very elegant way and jointly optimize in some sense string matching and arithmetic coding capability. It has been shown that these algorithms are all universal in the sense that they can achieve asymptotically the entropy rate of any stationary, ergodic source. Moreover, it has been proved that their worst case redundancies among all individual sequences of length $n$ are upper-bounded by $c \log\log n/\log n$, where $c$ is a constant. These algorithms have essentially linear computation and storage complexity. Simulation results show that these algorithms outperform the Unix Compress and Gzip algorithms, which are based on LZ78 and LZ77, respectively.

Many problems concerning these algorithms remain open, however. To conclude this paper, in the following paragraphs, we discuss some of these problems.

1) The technique we have adopted to analyze these algorithms is a combinatorial one. It is certainly desirable to have a probabilistic analysis of these algorithms. In particular, what are the average and pointwise redundancies of these algorithms? How does the irreducible grammar $G_i$ evolve? What properties does the set consisting of substrings represented by all $G_i$-variables have as $i$ gets larger and larger?
2) As the length of the data sequence $x$ increases, the size of $G_t$ gets larger and larger so that at some point, it will reach the memory limit that software and hardware devices can handle. If this happens, one certainly needs to modify the proposed algorithms in this paper. One solution is to freeze $G_t$ at this point and reuse $G_t$ to encode the remaining data sequence; we call this version the fixed-database version. Obviously, the fixed-database version is applicable only to the sequential and improved sequential algorithms. Another solution is to discard $G_t$ and restart these algorithms for the remaining sequence. These two solutions represent two extreme cases. One may expect that to get better compression performance, it should be arranged that $G_t$ should be changed gradually.
3) Analyze the performance of the fixed-database version.
4) Extend these algorithms to high-dimensional data and analyze compression performance accordingly.

## APPENDIX A

In this appendix, we prove Lemma 3. Since each variable $s_i \in \mathcal{S}(j_t)$ represents a distinct $\mathcal{A}$-sequence, in this proof we shall identify each symbol $s_i \in \mathcal{S}(j_t)$ with the $\mathcal{A}$-sequence rep-

resented by $s_i$. Accordingly, $|s_i|$ will denote the length of that $\mathcal{A}$-sequence. Let $u^{(j_t-1)}(G_t) = u_1u_2\cdots u_m$ be the partition sequence induced by $G_t$, where $u_l \in \mathcal{A}\cup\{s_1,\cdots,s_{j_t-1}\}$ for any $1 \le l \le m$. Assume that $m \ge 3$; this is certainly true whenever $|x| \ge 3$. Since $u^{(j_t-1)}(G_t)$ is the partition sequence induced by $G_t$, it follows that

$$|x| = \sum_{l=1}^{m} |u_l|. \tag{A.1}$$

Let

$$L \triangleq \sum_{l=1}^{m-2} [|u_l| + |u_{l+1}| + |u_{l+2}|].$$

Clearly, (A.1) implies that

$$|x| < L < 3|x|. \tag{A.2}$$

In view of Lemma 2, $u_lu_{l+1}u_{l+2}$, $l = 1, 2, \cdots, m-2$, are all distinct as sequences of length 3 from $\mathcal{S}(j_t)\cup\mathcal{A}$. Since each $u_l$ represents an $\mathcal{A}$-sequence, $u_lu_{l+1}u_{l+2}$ then represents the concatenation of the $\mathcal{A}$-sequences represented by $u_l$, $u_{l+1}$, and $u_{l+2}$, respectively. Note that the $\mathcal{A}$-sequences represented by $u_lu_{l+1}u_{l+2}$, $l = 1, 2, \cdots, m-2$, may not necessarily be distinct. Nonetheless, we can upper-bound the multiplicity. The number of integers $l$ for which $u_lu_{l+1}u_{l+2}$ represents the same $\mathcal{A}$-sequence of length $n$, is less than or equal to $\binom{n-1}{2}$ since each symbol $s_i \in \mathcal{S}(j_t)$ represents a distinct $\mathcal{A}$-sequence and all $u_lu_{l+1}u_{l+2}$, as sequences of length 3 from $\mathcal{S}(j_t)\cup\mathcal{A}$, are distinct. Thus for any $n \ge 3$

$$\begin{aligned} m_n &\triangleq |\{l\colon |u_l| + |u_{l+1}| + |u_{l+2}| = n,\ 1 \le l \le m-2\}| \\ &\le \binom{n-1}{2}|\mathcal{A}|^n. \end{aligned} \tag{A.3}$$

Clearly

$$L = \sum_{n=3}^{|x|} m_n n$$

and

$$m-2 = \sum_{n=3}^{|x|} m_n.$$

Of course, $m_n$ may be 0 for some $n$. Now it is easy to see that given $L$, $m$ is maximized when all $|u_l| + |u_{l+1}| + |u_{l+2}|$ are as small as possible, subject to the constraints given by (A.3). For any $k \ge 3$, let

$$N_k \triangleq \sum_{n=3}^{k} n\binom{n-1}{2}|\mathcal{A}|^n$$

and

$$M_k \triangleq \sum_{n=3}^{k} \binom{n-1}{2}|\mathcal{A}|^n.$$

It is easy to verify that

$$M_k = \frac{|\mathcal{A}|^3}{2}\left[\frac{k(k-1)|\mathcal{A}|^{k-2}}{|\mathcal{A}|-1} - \frac{2k|\mathcal{A}|^{k-1}}{(|\mathcal{A}|-1)^2} + \frac{2(|\mathcal{A}|^k-1)}{(|\mathcal{A}|-1)^3}\right]$$

and

$$\begin{aligned} N_k &= \frac{|\mathcal{A}|^3}{2}\left[\frac{(k+1)k(k-1)|\mathcal{A}|^{(k-2)}}{|\mathcal{A}|-1} - \frac{3(k+1)k|\mathcal{A}|^{(k-1)}}{(|\mathcal{A}|-1)^2} \right. \\ &\qquad \left. + \frac{6(k+1)|\mathcal{A}|^k}{(|\mathcal{A}|-1)^3} - \frac{6(|\mathcal{A}|^{k+1}-1)}{(|\mathcal{A}|-1)^4}\right] \\ &= \frac{|\mathcal{A}|^3}{2}\,\frac{(k+1)k(k-1)|\mathcal{A}|^{(k-2)}}{|\mathcal{A}|-1} - \frac{3}{(|\mathcal{A}|-1)}\,M_{k+1} \\ &= (k+1)M_k + (k+1)|\mathcal{A}|^3\left[\frac{k|\mathcal{A}|^{k-1}}{(|\mathcal{A}|-1)^2} - \frac{(|\mathcal{A}|^k-1)}{(|\mathcal{A}|-1)^3}\right] \\ &\quad - \frac{3}{(|\mathcal{A}|-1)}\,M_{k+1} \\ &= (k+1)M_k - \frac{3}{(|\mathcal{A}|-1)}\,M_k - \frac{3}{(|\mathcal{A}|-1)}\binom{k}{2}|\mathcal{A}|^{k+1} \\ &\quad + (k+1)|\mathcal{A}|^3\left[\frac{k|\mathcal{A}|^{k-1}}{(|\mathcal{A}|-1)^2} - \frac{(|\mathcal{A}|^k-1)}{(|\mathcal{A}|-1)^3}\right] \\ &= (k+1)M_k - \frac{3}{(|\mathcal{A}|-1)}\,M_k - 3\frac{|\mathcal{A}|^3}{2} \\ &\quad \cdot\left[\frac{k(k-1)|\mathcal{A}|^{k-2}}{|\mathcal{A}|-1} - \frac{2(k+1)}{3}\,\frac{k|\mathcal{A}|^{k-1}}{(|\mathcal{A}|-1)^2} \right. \\ &\qquad \left. + \frac{2(k+1)}{3}\frac{(|\mathcal{A}|^k-1)}{(|\mathcal{A}|-1)^3}\right] \\ &\overset{1)}{\ge} (k+1)M_k - \frac{3}{(|\mathcal{A}|-1)}\,M_k - 3M_k \\ &\ge (k-5)M_k \end{aligned} \tag{A.4}$$

where the inequality 1) is true for any $k \ge 2$, and the last inequality is due to the fact that $|\mathcal{A}| \ge 2$. If $L$ happens to be $N_k$ for some $k > 5$, then $m-2 \le M_k$. In view of (A.4), we then have

$$\frac{m-2}{L} \le \frac{M_k}{N_k} \le \frac{1}{k-5}. \tag{A.5}$$

If $N_k < L < N_{k+1}$, we write $L = N_k + \Delta$, where $\Delta < (k+1)\binom{k}{2}|\mathcal{A}|^{k+1}$. Then

$$m-2 \le M_k + \frac{\Delta}{k+1} \le \frac{N_k}{k-5} + \frac{\Delta}{k+1}.$$

This, together with (A.5), implies that

$$\frac{m-2}{L} \le \frac{1}{k-5} \tag{A.6}$$

whenever $N_k \le L < N_{k+1}$ for some $k > 5$. We next bound $k$ in terms of $L$. Since

$$L \ge N_k \ge k\binom{k-1}{2}|\mathcal{A}|^k$$

it follows that

$$k \le \frac{\log L}{\log|\mathcal{A}|} \tag{A.7}$$

and whenever $N_k \le L < N_{k+1}$ for some $k > 6$

$$\frac{m}{L} = \frac{m-2}{L} + \frac{2}{L} \le \frac{1}{k-6}. \tag{A.8}$$

On the other hand,

$$L < N_{k+1} < 3k^3|\mathcal{A}|^{k+2}.$$

From this

$$\log L < 3\log k + (k+2)\log|\mathcal{A}| + 2$$

which, combined with (A.7), implies that

$$(k+2)\log|\mathcal{A}| > \log L - 3\log\log L - 2. \tag{A.9}$$

Combining (A.9) with (A.8) yields

$$\frac{m}{L} \le \frac{\log|\mathcal{A}|}{\log L - 3\log\log L - 2 - 8\log|\mathcal{A}|}.$$

This, coupled with (A2), implies that

$$\begin{aligned}\frac{|u^{(j_t-1)}(G_t)|}{|x|} &\le \frac{3\log|\mathcal{A}|}{\log L - 3\log\log L - 2 - 8\log|\mathcal{A}|} \\ &\le \frac{3\log|\mathcal{A}|}{\log|x| - 3\log\log|x| - 2 - 8\log|\mathcal{A}|}\end{aligned} \tag{A.10}$$

whenever

$$|x| \ge \sum_{n=3}^{7} n\binom{n-1}{2}|\mathcal{A}|^n$$

and

$$\log|x| - 3\log\log|x| - 2 - 8\log|\mathcal{A}| > 0.$$

To upper-bound $|G_t|/|x|$, note that each variable $s_i$ in $\mathcal{S}(j_t)$ other than $s_0$ appears at least once in the partition sequence $u^{(j_t-1)}(G_t)$. Thus from Lemma 1

$$|G_t| = \left|u^{(j_t-1)}(G_t)\right| + (j_t - 1) \le 2\left|u^{(j_t-1)}(G_t)\right|$$

which, together with (A.10), implies that

$$\frac{|G_t|}{|x|} \le \frac{6\log|\mathcal{A}|}{\log|x| - 3\log\log|x| - 2 - 8\log|\mathcal{A}|}$$

whenever

$$|x| \ge \sum_{n=3}^{7} n\binom{n-1}{2}|\mathcal{A}|^n$$

and

$$\log|x| - 3\log\log|x| - 2 - 8\log|\mathcal{A}| > 0.$$

From this and (A.10), Lemma 3 follows.

## Appendix B

In this appendix, we prove Lemma 4. We use the same notation as in the proof of Lemma 3. Let $u^{(j_t-1)}(G_t) = u_1u_2\cdots u_m$ be the partition sequence induced by $G_t$, where $u_l \in \mathcal{A} \cup \{s_1, \cdots, s_{j_t-1}\}$ for any $1 \le l \le m$. As mentioned in Section V, each variable $s_i \in \mathcal{S}(j_t)$ other than $s_0$ appears at least once in $u^{(j_t-1)}(G_t)$. Let $\mathcal{A}' \subset \mathcal{A}$ consist of all $\alpha \in \mathcal{A}$ that appear in $u^{(j_t-1)}(G_t)$. Assume that $m \ge 3$; this is certainly true whenever $|x| \ge 3$. Recall from the proof of Lemma 3 that

$$L = \sum_{l=1}^{m-2}\left[|u_l| + |u_{l+1}| + |u_{l+2}|\right].$$

and

$$|x| < L < 3|x| \tag{B.1}$$

In view of Lemma 2, $u_lu_{l+1}u_{l+2}$, $l = 1, 2, \cdots, m-2$, are all distinct as sequences of length 3 from $\mathcal{A}' \cup \{s_1, \cdots, s_{j_t-1}\}$. From this, it follows that

$$\begin{aligned}L &\le \sum_{\alpha_1}\sum_{\alpha_2}\sum_{\alpha_3}\left[|\alpha_1| + |\alpha_2| + |\alpha_3|\right] \\ &= 3\left(|\mathcal{A}'| + j_t - 1\right)^2\sum_{\alpha_1}|\alpha_1| \\ &\le 3\left[\sum_{\alpha_1}|\alpha_1|\right]^3\end{aligned} \tag{B.2}$$

where $\sum_{\alpha_i}$ denotes the summation over $\mathcal{A}' \cup \{s_1, \cdots, s_{j_t-1}\}$ for any $1 \le i \le 3$. In view of (B.1) and (B.2), we have

$$L' \triangleq \sum_{\alpha_1}|\alpha_1| > \left(\frac{|x|}{3}\right)^{1/3}. \tag{B.3}$$

To estimate the average length of the $\mathcal{A}$-sequences represented by $s_i$, $i = 1, \cdots, j_t - 1$, we first evaluate

$$\frac{1}{|\mathcal{A}'| + j_t - 1}\sum_{\alpha_1}|\alpha_1|.$$

Note that each $\alpha_1 \in \mathcal{A}' \cup \{s_1, \cdots, s_{j_t-1}\}$ represents a distinct $\mathcal{A}$-sequence. Standard techniques can be used to show that

$$\frac{L'}{|\mathcal{A}'| + j_t - 1} \ge \frac{\log L' - \log\log L' - 1 - 3\log|\mathcal{A}|}{\log|\mathcal{A}|}$$

whenever $L' \ge |\mathcal{A}| + |\mathcal{A}|^2$. This, together with (B.3), implies that

$$\frac{L'}{|\mathcal{A}'| + j_t - 1} \ge \frac{\log|x| - 3\log\log|x| - 3 - 9\log|\mathcal{A}| - \log 3}{3\log|\mathcal{A}|} \tag{B.4}$$

whenever $|x| \ge 3(|\mathcal{A}| + |\mathcal{A}|^2)^3$. Note that $j_t$ must be $\ge 2$ if $|x| \ge 3(|\mathcal{A}| + |\mathcal{A}|^2)^3$. Since the length of the $\mathcal{A}$-sequence represented by each $s_i$, $i > 0$, is $\ge 2$, it follows from (B.4) that

$$\frac{1}{j_t - 1}\sum_{i=1}^{j_t-1}|s_i| \ge \frac{\log|x| - 3\log\log|x| - 3 - 9\log|\mathcal{A}| - \log 3}{3\log|\mathcal{A}|}$$

whenever $|x| \ge 3(|\mathcal{A}| + |\mathcal{A}|^2)^3$. This completes the proof of Lemma 4.

## Appendix C

In this Appendix, we prove Lemma 5.

We first establish a relationship between $t$ and $|G_t|$. Recall from Section III that the proposed irreducible grammar transform parses the sequence $x$ sequentially into nonoverlapping

substrings $\{x_1, x_2 \cdots x_{n_2}, \cdots, x_{n_{t-1}+1} \cdots x_{n_t}\}$ and builds sequentially an irreducible grammar $G_i$ with variable set $\mathcal{S}(j_i)$ for each $x_1 \cdots x_{n_i}$, where $1 \le i \le t, n_1 = 1$, and $n_t = n$. From Theorem 1, it follows that the size $|G_{i+1}|$ of $G_{i+1}$ increases by 1 in Cases 1 and 2 of Theorem 1 and remains the same as $|G_i|$ in Case 3 of Theorem 1. Thus the number $t$ is equal to $|G_t|$ plus the number of times Case 3 of Theorem 1 appears in the entire irreducible grammar transform process. Recall that $I(1) = 0$, and for any $i > 1$, $I(i)$ is equal to 0 if $G_i$ is equal to the appended $G_{i-1}$, and 1 otherwise. One can determine the number of times Case 3 of Theorem 1 appears by looking at the runs of 1's in the binary sequence $I(1)I(2)\cdots I(t)$. In view of Theorem 1, it is easy to see that the total number of runs of 1's in the binary sequence $I(1)I(2)\cdots I(t)$ is $j_t - 1$; each run of 1's is corresponding to a variable $s_j$ for some $1 \le j \le j_t - 1$. Let $[l_j, r_j]$ be the interval corresponding to the $j$th run of 1's, that is, $I(l_j)I(l_j + 1)\cdots I(r_j)$ is the $j$th run of 1's. This, of course, implies that $I(l_j - 1) = 0$ and $I(r_j + 1) = 0$ if $r_j < t$. The variable $s_j$ is introduced at the stage $l_j$, and Case 3 of Theorem 1 holds for $l_j + 1, l_j + 2, \cdots, r_j$ if $r_j > l_j$. Then one can see that the number of times Case 3 of Theorem 1 appears in the entire irreducible grammar transform process is equal to

$$\sum_{j=1}^{j_t-1} [r_j - l_j] = \sum_{r_j > l_j} [r_j - l_j]$$

where the summation $\sum_{r_j > l_j}$ is taken over all $j$ satisfying $r_j > l_j$. Thus we get the following identity:

$$t = |G_t| + \sum_{r_j > l_j} [r_j - l_j]. \tag{C.1}$$

In view of Lemma 3, it now suffices to upper-bound the sum in (C.1). To this end, let us reveal a hierarchical structure among the intervals $[l_j, r_j]$ satisfying $r_j > l_j$. An interval $[l_j, r_j]$ with $r_j > l_j$ is called a *top interval* if $G_{r_j}(s_j)$ is a substring of $G_{l_j-1}(s_0)$. In other words, for a top interval $[l_j, r_j]$, $G_{r_j}(s_j)$ is read off directly from $G_{l_j-1}(s_0)$, and $G_{r_j}$ is obtained from $G_{l_j-1}$ by repeatedly applying Reduction Rules 2 and 1. Note that the first interval $[l_j, r_j]$ with $r_j > l_j$ is a top interval. Assume that there are a total of $k$ top intervals $[l_{m_1}, r_{m_1}], [l_{m_2}, r_{m_2}], \cdots, [l_{m_k}, r_{m_k}]$, where $1 \le m_1 < m_2 < \cdots < m_k \le j_t - 1$. Since $G_i$ is irreducible for any $i$ and since $G_{r_{m_i}}(s_{m_i})$ is a substring of $G_{l_{m_i}-1}(s_0)$, a similar argument to the proof of Lemma 2 can be used to show that there is no repeated pattern of length 3 in the $k$ sequences $G_{r_{m_1}}(s_{m_1}), G_{r_{m_2}}(s_{m_2}), \cdots, G_{r_{m_k}}(s_{m_k})$, where patterns are counted in the sliding-window, overlapping manner and in all the $k$ sequences. All other intervals $[l_j, r_j]$ with $r_j > l_j$ are related to top intervals. To show this, we introduce a new concept. An interval $[l_j, r_j]$ with $r_j > l_j$ is said to be *subordinate directly to* an interval $[l_{j'}, r_{j'}]$, where $1 \le j' < j \le j_t - 1$, if $G_{r_j}(s_j)$ is a substring of $G_{l_j-1}(s_{j'})$. An interval $[l_j, r_j]$ with $r_j > l_j$ is said to be *subordinate to* an interval $[l_{j'}, r_{j'}]$, where $1 \le j' < j \le j_t - 1$, if there is a sequence of intervals $[l_{i_1}, r_{i_1}], \cdots, [l_{i_n}, r_{i_n}]$ such that $[l_{i_m}, r_{i_m}]$ is subordinate directly to $[l_{i_{m+1}}, r_{i_{m+1}}]$ for $m = 0, 1, \cdots, n$, where $i_0 = j$ and $i_{n+1} = j'$. It is easy to see that every interval $[l_j, r_j]$ with $r_j > l_j$ is subordinate to a top interval. Furthermore, for any interval $[l_j, r_j]$ with $r_j > l_j$, we have

$$r_j - l_j = |G_{r_j}(s_j)| - 2$$

where $|G_{r_j}(s_j)|$ denotes the length of $G_{r_j}(s_j)$, which is a sequence from $\mathcal{A} \cup \{s_1, \cdots, s_j\}$. This implies that

$$\sum_{r_j > l_j} [r_j - l_j] = \sum_{r_j > l_j} [|G_{r_j}(s_j)| - 2]. \tag{C.2}$$

We next upper-bound the sum on the right-hand side of (C.2).

Let us focus on a particular top interval, say, $[l_{m_1}, r_{m_1}]$. Consider all intervals $[l_j, r_j]$ that are subordinate to the top interval $[l_{m_1}, r_{m_1}]$. Note that even though $[l_j, r_j]$ is subordinate directly to $[l_{m_1}, r_{m_1}]$, the sequence $G_{r_j}(s_j)$ is not necessarily a substring of $G_{r_{m_1}}(s_{m_1})$. The reason is as follows: 1) $G_{r_{m_1}}(s_{m_1})$ is a sequence from $\mathcal{A} \cup \{s_1, s_2, \cdots, s_{m_1-1}\}$; 2) by the definition given in the above paragraph, $m_1 < j$; and 3) before the stage $l_j - 1$, the production rule corresponding to $s_{m_1}$ may be changed, and as a result, $G_{r_j}(s_j)$ may contain some variables $s_i$, where $m_1 < i < j$. Nonetheless, as long as $[l_j, r_j]$ is subordinate to $[l_{m_1}, r_{m_1}]$, the sequence $G_{r_j}(s_j)$ is indeed generated from $G_{r_{m_1}}(s_{m_1})$. By applying a procedure similar to the parallel replacement procedure mentioned in Section II, the sequence $G_{r_j}(s_j)$ can be expanded so that the expanded sequence $\tilde{G}_{r_j}(s_j)$ is a substring of $G_{r_{m_1}}(s_{m_1})$. Using the tree structure implied implicitly by the subordinate relation, one can verify that the expanded sequences $\tilde{G}_{r_j}(s_j)$ corresponding to all intervals $[l_j, r_j]$ subordinate to the top interval $[l_{m_1}, r_{m_1}]$ satisfy the following properties.

- e.1) Every expanded sequence $\tilde{G}_{r_j}(s_j)$ is a substring of $G_{r_{m_1}}(s_{m_1})$.
- e.2) $3 \le |G_{r_j}(s_j)| \le |\tilde{G}_{r_j}(s_j)|$, where $|\tilde{G}_{r_j}(s_j)|$ denotes the length of $\tilde{G}_{r_j}(s_j)$ as a sequence from $\mathcal{A} \cup \{s_1, s_2, \cdots, s_{m_1-1}\}$.
- e.3) All expanded sequences $\tilde{G}_{r_j}(s_j)$ are distinct.
- e.4) For any two expanded sequences $\tilde{G}_{r_j}(s_j)$ and $\tilde{G}_{r_{j'}}(s_{j'})$, which correspond, respectively, to two intervals $[l_j, r_j]$ and $[l_{j'}, r_{j'}]$ that are subordinate to $[l_{m_1}, r_{m_1}]$, either $\tilde{G}_{r_j}(s_j)$ is a substring of $\tilde{G}_{r_{j'}}(s_{j'})$, or $\tilde{G}_{r_{j'}}(s_{j'})$ is a substring of $\tilde{G}_{r_j}(s_j)$, or $\tilde{G}_{r_j}(s_j)$ and $\tilde{G}_{r_{j'}}(s_{j'})$ are nonoverlapping substrings of $G_{r_{m_1}}(s_{m_1})$.
- e.5) For any three expanded sequences $\tilde{G}_{r_j}(s_j)$, $\tilde{G}_{r_{j'}}(s_{j'})$, and $\tilde{G}_{r_{j''}}(s_{j''})$, which correspond, respectively, to three distinct intervals subordinate to $[l_{m_1}, r_{m_1}]$, if both $\tilde{G}_{r_j}(s_j)$ and $\tilde{G}_{r_{j'}}(s_{j'})$ are substrings of $\tilde{G}_{r_{j''}}(s_{j''})$ and if neither $\tilde{G}_{r_j}(s_j)$ nor $\tilde{G}_{r_{j'}}(s_{j'})$ is a substring of the other, then $\tilde{G}_{r_j}(s_j)$ and $\tilde{G}_{r_{j'}}(s_{j'})$ are nonoverlapping substrings of $\tilde{G}_{r_{j''}}(s_{j''})$.

In view of these properties and the fact that there is no repeated pattern of length 3 in $G_{r_{m_1}}(s_{m_1})$, these expanded sequences can be arranged in a hierarchical way, as shown in Fig. 2. The top line segment in Fig. 2 represents the sequence $G_{r_{m_1}}(s_{m_1})$. Each of the other line segments in Fig. 2 represents a different

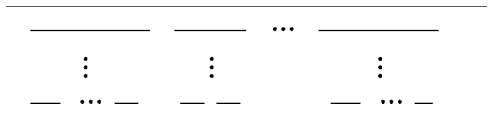

Fig. 2. Hierarchical representation of expanded sequences related to a top interval.

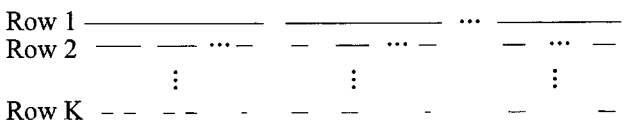


Fig. 3. Hierarchical representation of expanded sequences.

expanded sequence $\tilde{G}_{r_j}(s_j)$. For each line segment, the line segments underneath it are its nonoverlapping substrings. From Property e.3), it follows that if for some line segment, there is only one line segment underneath it, then the length of the line segment underneath it is strictly less than its own length. Here by the length of a line segment, we mean the length of the sequence from $\mathcal{S}(j_t) \cup \mathcal{A}$ it represents.

The argument in the above paragraph applies equally well to all other top intervals. Since there is no repeated pattern of length 3 in the $k$ sequences

$$G_{r_{m_1}}(s_{m_1}), G_{r_{m_2}}(s_{m_2}), \cdots, G_{r_{m_k}}(s_{m_k})$$

the expanded sequences $\tilde{G}_{r_j}(s_j)$ corresponding to all intervals $[l_j, r_j]$ with $r_j > l_j$ can be arranged in a similar fashion to Fig. 2, as shown in Fig. 3. Once again, the top $k$ line segments in Fig. 3 represent the $k$ sequences

$$G_{r_{m_1}}(s_{m_1}), G_{r_{m_2}}(s_{m_2}), \cdots, G_{r_{m_k}}(s_{m_k}).$$

Each of the other line segments in Fig. 3 represents a different expanded sequence $\tilde{G}_{r_j}(s_j)$. Line segments in Fig. 3 have a similar interpretation to line segments in Fig. 2.

Let us now go back to (C.2). In view of Property e.2)

$$\sum_{r_j > l_j} [|G_{r_j}(s_j)| - 2] \le \sum_{r_j > l_j} [|\tilde{G}_{r_j}(s_j)| - 2] \tag{C.3}$$

where $\tilde{G}_{r_j}(s_j)$ is the same as $G_{r_j}(s_j)$ whenever $[l_j, r_j]$ is a top interval. Since there is no repeated pattern of length 3 in the $k$ sequences

$$G_{r_{m_1}}(s_{m_1}), G_{r_{m_2}}(s_{m_2}), \cdots, G_{r_{m_k}}(s_{m_k})$$

it follows that there is no repeated pattern of length 3 in each row of Fig. 3 either. This implies that $|\tilde{G}_{r_j}(s_j)| - 2$ is equal to the number of patterns of length 3 appearing in the line segment corresponding to $\tilde{G}_{r_j}(s_j)$. Let $V_i$ be the set consisting of all patterns of length 3 appearing in Row $i$ of Fig. 3. Then we have

$$V_1 \supset V_2 \cdots \supset V_K \tag{C.4}$$

and

$$|V_1| > |V_2| > \cdots > |V_K| \tag{C.5}$$

where $|V_i|$ denotes the cardinality of $V_i$. Furthermore,

$$\sum_{r_j > l_j} [|\tilde{G}_{r_j}(s_j)| - 2] = \sum_{i=1}^{K} |V_i|. \tag{C.6}$$

For each $1 \le i \le K$, let

$$L_i = \sum_{\alpha_1\alpha_2\alpha_3 \in V_i} [|\alpha_1| + |\alpha_2| + |\alpha_3|]$$

where $|\alpha_j|$ $(1 \le j \le 3)$ denotes the length of the $\mathcal{A}$-sequence represented by $\alpha_j$. (Note that each $\alpha_j$ itself is a symbol in $\mathcal{S}(j_t) \cup \mathcal{A}$.) Let

$$L = \sum_{i=1}^{K} L_i.$$

At this point, we invoke the following result, which will be proved in Appendix D.

*Lemma 6:* There is a constant $d_2$, which depends only on $|\mathcal{A}|$, such that

$$\sum_{i=1}^{K} |V_i| \le d_2 \frac{L}{\log L}.$$

It is easy to see that

$$L < \sum_{r_j > l_j} 3|s_j| < 3|x|$$

where $|s_j|$ denotes the length of the $\mathcal{A}$-sequence represented by the variable $s_j$. This, together with Lemma 6, implies

$$\sum_{i=1}^{K} |V_i| \le d_2 \frac{3|x|}{\log(3|x|)}. \tag{C.7}$$

Putting (C.2), (C.3), (C.6), and (C.7) together, we get

$$\sum_{r_j > l_j} [r_j - l_j] \le d_2 \frac{3|x|}{\log(3|x|)}$$

which, coupled with (C.1) and Lemma 3, implies

$$t \le |G_t| + d_2 \frac{3|x|}{\log(3|x|)} \le d_1 \frac{|x|}{\log|x|}$$

for some constant $d_1$. This completes the proof of Lemma 5.

## APPENDIX D

In this appendix, we prove Lemma 6. Recall that each $V_i$ is a subset of $(\mathcal{S}(j_t) \cup \mathcal{A})^3$ and the sequence $\{V_i\}_{i=1}^{K}$ satisfies (C.4) and (C.5). For convenience, we also write a pattern of length 3, $\alpha_1\alpha_2\alpha_3 \in (\mathcal{S}(j_t) \cup \mathcal{A})^3$, as a vector $(\alpha_1, \alpha_2, \alpha_3)$. As in the proof of Lemma 3, since each symbol $\alpha \in \mathcal{S}(j_t)$ represents a distinct $\mathcal{A}$-sequence, we shall identify $\alpha \in \mathcal{S}(j_t)$ with the $\mathcal{A}$-sequence represented by $\alpha$. It is easy to see that for any $n \ge 3$

$$\left|\left\{\alpha_1\alpha_2\alpha_3 \colon \alpha_1\alpha_2\alpha_3 \in (\mathcal{S}(j_t) \cup \mathcal{A})^3, |\alpha_1| + |\alpha_2| + |\alpha_3| = n\right\}\right| \le \binom{n-1}{2} |\mathcal{A}|^n \tag{D.1}$$

where $|\alpha_i| (1 \le i \le 3)$ denotes the length of the $\mathcal{A}$-sequence represented by $\alpha_i$. The number $L$ is defined as

$$L = \sum_{i=1}^{K} \sum_{\alpha_1\alpha_2\alpha_3 \in V_i} [|\alpha_1| + |\alpha_2| + |\alpha_3|].$$

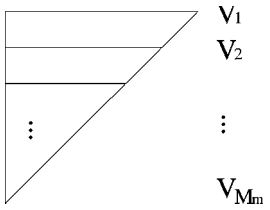


Fig. 4. Triangle structure of the sets $V_i$ in the worst case.

We want to show that $\sum_{i=1}^{K}|V_i|$ is upper-bounded by $L/\log L$ multiplied by some constant. Since the function $L/\log L$ is strictly increasing for $L \geq 3$, it is enough for us to consider worst cases. Clearly, given $\sum_{i=1}^{K}|V_i|$, $L$ is minimized when all $|\alpha_1|+|\alpha_2|+|\alpha_3|(\alpha_1\alpha_2\alpha_3 \in V_i$ and $i = 1, 2, \cdots, K)$ are as small as possible, subject to the constraints (C.4), (C.5), and (D.1). For any $m \geq 3$, let

$$N_m \triangleq \sum_{n=3}^{m} n\binom{n-1}{2}|\mathcal{A}|^n$$

and

$$M_m \triangleq \sum_{n=3}^{m} \binom{n-1}{2}|\mathcal{A}|^n.$$

Note that $M_m$ is equal to the number of string vectors $(\alpha_1, \alpha_2, \alpha_3)$, where $\alpha_i \in \mathcal{A}^+$, $i = 1, 2, 3$, such that $|\alpha_1|+|\alpha_2|+|\alpha_3| \leq m$. From the proof of Lemma 3, it follows that

$$M_m = \frac{|\mathcal{A}|^3}{2}\left[\frac{m(m-1)|\mathcal{A}|^{m-2}}{|\mathcal{A}|-1} - \frac{2m|\mathcal{A}|^{m-1}}{(|\mathcal{A}|-1)^2} + \frac{2(|\mathcal{A}|^m-1)}{(|\mathcal{A}|-1)^3}\right] \quad \text{(D.2)}$$

and

$$N_m = \frac{|\mathcal{A}|^3}{2}\left[\frac{(m+1)m(m-1)|\mathcal{A}|^{(m-2)}}{|\mathcal{A}|-1} - \frac{3(m+1)m|\mathcal{A}|^{(m-1)}}{(|\mathcal{A}|-1)^2} + \frac{6(m+1)|\mathcal{A}|^m}{(|\mathcal{A}|-1)^3} - \frac{6(|\mathcal{A}|^{m+1}-1)}{(|\mathcal{A}|-1)^4}\right]. \quad \text{(D.3)}$$

If $\sum_{i=1}^{K}|V_i|$ happens to be equal to $M_m(M_m+1)/2$, then

$$L \geq \sum_{n=3}^{m}\left[N_{n-1}\binom{n-1}{2}|\mathcal{A}|^n + \frac{1}{2}n\binom{n-1}{2}|\mathcal{A}|^n\left(\binom{n-1}{2}|\mathcal{A}|^n+1\right)\right] \quad \text{(D.4)}$$

where $N_2$ is set to 0 as a convention. In (D.4), the equality holds when $K = M_m$, $V_1$ consists of all string vectors $(\alpha_1, \alpha_2, \alpha_3)$, where $\alpha_i \in \mathcal{A}^+, i = 1, 2, 3$, such that $|\alpha_1|+|\alpha_2|+|\alpha_3| \leq m$, and $V_i$, for $2 \leq i \leq M_m$, is obtained from $V_{i-1}$ by deleting a string vector $(\alpha_1, \alpha_2, \alpha_3)$ with the largest $|\alpha_1|+|\alpha_2|+|\alpha_3|$. In other words, $L$ is minimized when the sets $V_i$ are packed into a tight triangle, as shown in Fig. 4. In Fig. 4, the $i$th line segment counted from the top represents the set $V_i$. Denote the sum on the right-hand side of (D.4) by $\tilde{N}_m$. It follows from (D.4) that

$$\tilde{N}_m > \frac{1}{8}m(m-1)^2(m-2)^2|\mathcal{A}|^{2m}. \quad \text{(D.5)}$$

Clearly, $L$ is strictly greater than $\tilde{N}_m$ if

$$\sum_{i=1}^{K}|V_i| > M_m(M_m+1)/2.$$

Thus whenever

$$M_m(M_m+1)/2 \leq \sum_{i=1}^{K}|V_i| < M_{m+1}(M_{m+1}+1)/2$$

one has

$$\begin{aligned}\sum_{i=1}^{K}|V_i| &\leq M_{m+1}(M_{m+1}+1)/2 \\ &= \tilde{N}_m\frac{M_{m+1}(M_{m+1}+1)/2}{\tilde{N}_m} \\ &\overset{1)}{\leq} \tilde{N}_m\frac{4M_{m+1}(M_{m+1}+1)}{m(m-1)^2(m-2)^2|\mathcal{A}|^{2m}} \\ &\overset{2)}{\leq} d_3\frac{\tilde{N}_m}{m} \\ &\overset{3)}{\leq} d_4\frac{\tilde{N}_m}{\log(\tilde{N}_m)} \\ &\leq d_4\frac{L}{\log L}\end{aligned}$$

where $d_3$ and $d_4$ are some constants depending only on $|\mathcal{A}|$. The inequality 1) is due to (D.5). The inequality 2) follows from the observation that from (D.2), $M_m = O(m^2|\mathcal{A}|^m)$, and, as a result,

$$\frac{M_{m+1}(M_{m+1}+1)}{m(m-1)^2(m-2)^2|\mathcal{A}|^{2m}} = O\left(\frac{1}{m}\right).$$

The inequality 3) is due to the fact that $m = O(\log(\tilde{N}_m))$. Finally, the last inequality follows from the fact that the function $L/\log L$ is increasing and $L \geq \tilde{N}_m$. This completes the proof of Lemma 6.

## REFERENCES

[1] N. Abramson, *Information Theory and Coding*. New York: McGraw-Hill, 1963.
[2] A. Barron, "Logically smooth density estimation," Ph.D. dissertation, Stanford University, Stanford, CA, 1985.
[3] J. Bentley, D Sleator, R. Tarjan, and V. K. Wei, "A locally adaptive data compression scheme," *Commun. Assoc. Comput. Mach.*, vol. 29, pp. 320–330, 1986.
[4] J. G. Cleary and I. H. Witten, "Data compression using adaptive coding and partial string matching," *IEEE Trans. Commun.*, vol. COM-32, pp. 396–402, 1984.
[5] G. V. Cormack and R. N. S. Horspool, "Data compression using dynamic Markov modeling," *Computer J.*, vol. 30, pp. 541–550, 1987.
[6] T. M. Cover, "Enumerative source encoding," *IEEE Trans. Inform. Theory*, vol. IT-19, pp. 73–77, 1973.
[7] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. New York: Wiley, 1991.
[8] L. D. Davisson, "Universal noiseless coding," *IEEE Trans. Inform. Theory*, vol. IT-19, pp. 783–795, 1973.
[9] P. Elias, "Interval and recency rank source coding: Two on-line adaptive variable length schemes," *IEEE Trans. Inform. Theory*, vol. IT-33, pp. 1–15, 1987.
[10] R. G. Gallager, "Variations on a theme by Huffman," *IEEE Trans. Inform. Theory*, vol. IT-24, pp. 668–674, 1978.
[11] J. C. Kieffer, "Sample converses in source coding theory," *IEEE Trans. Inform. Theory*, vol. 37, pp. 263–268, 1991.

[12] J. C. Kieffer and E.-H. Yang, "Grammar based codes: A new class of universal lossless source codes," *IEEE Trans. Inform. Theory*, submitted for publication.
[13] J. C. Kieffer, E.-H. Yang, G. Nelson, and P. Cosman, "Universal lossless compression via multilevel pattern matching," *IEEE Trans. Inform. Theory*, submitted for publication.
[14] J. C. Kieffer and E.-H. Yang, "Lossless data compression algorithms based on substitution tables," in *Proc. IEEE 1998 Canadian Conf. Electrical and Computer Engineering*, Waterloo, Ont., Canada, May 1998, pp. 629–632.
[15] ——, "Ergodic behavior of graph entropy," *ERA Amer. Math. Soc.*, vol. 3, no. 1, pp. 11–16, 1997.
[16] A. Lempel and J. Ziv, "On the complexity of finite sequences," *IEEE Trans. Inform. Theory*, vol. IT-22, pp. 75–81, 1976.
[17] D. L. Neuhoff and P. C. Shields, "Simplistic universal coding," *IEEE Trans. Inform. Theory*, vol. 44, pp. 778–781, Mar. 1998.
[18] C. Nevill-Manning and I. H. Witten, "Compression and explanation using hierarchical grammars," *Computer J.*, vol. 40, pp. 103–116, 1997.
[19] D. S. Ornstein and P. C. Shields, "Universal almost sure data compression," *Ann. Probab.*, vol. 18, pp. 441–452, 1990.
[20] R. Pasco, "Source coding algorithms for fast data compression," Ph.D. dissertation, Stanford Univ., Stanford, CA, 1976.
[21] 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.
[22] J. Rissanen, "Generalized Kraft inequality and arithmetic coding," *IBM J. Res. Develop.*, vol. 20, pp. 198–203, 1976.
[23] J. Rissanen and G. G. Langdon, "Arithmetic coding," *IBM J. Res. Develop.*, vol. 23, pp. 149–162, 1979.
[24] J. Rissanen, "A universal data compression system," *IEEE Trans. Inform. Theory*, vol. IT-29, no. 5, pp. 656–664, Sept. 1983.
[25] B. Y. Ryabko, "Data compression by means of a 'book stack'," *Probl. Inform. Transm.*, vol. 16, no. 4, pp. 16–21, 1980.
[26] M. J. Weinberger, A. Lempel, and J. Ziv, "A sequential algorithm for the universal coding of finite memory sources," *IEEE Trans. Inform. Theory*, vol. 38, pp. 1002–1014, May 1992.
[27] F. M. J. Willems, "The context-tree weighting method: Extensions," *IEEE Trans. Inform. Theory*, vol. 44, pp. 792–798, Mar. 1998.
[28] F. M. J. Willems, Y. M. Shtarkov, and T. J. Tjalkens, "The context-tree weighting method: Basic properties," *IEEE Trans. Inform. Theory*, vol. 41, pp. 653–664, May 1995.
[29] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Commun. Assoc. Comput. Mach.*, vol. 30, pp. 520–540, 1987.
[30] E.-h. Yang, "Universal almost sure data compression for abstract alphabets and arbitrary fidelity criterions," *Probl. Contr. Inform. Theory*, vol. 20, pp. 397–408, 1991.
[31] E.-h. Yang and Y. Jia, "Efficient grammar-based data compression algorithms: Complexity, implementation, and simulation results," paper, in preparation.
[32] E.-h. Yang and J. C. Kieffer, "On the redundancy of the fixed database Lempel-Ziv algorithm for $\phi$-mixing sources," *IEEE Trans. Inform. Theory*, vol. 43, pp. 1101–1111, July 1997.
[33] ——, "On the performance of data compression algorithms based upon string matching," *IEEE Trans. Inform. Theory*, vol. 44, pp. 47–65, Jan. 1998.
[34] E.-h. Yang and S. Shen, "Chaitin complexity, Shannon information content of a single event and infinite random sequences (I)," *Science in China*, ser. A, vol. 34, pp. 1183–1193, 1991.
[35] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. Inform. Theory*, vol. IT-23, pp. 337–343, 1977.
[36] ——, "Compression of individual sequences via variable rate coding," *IEEE Trans. Inform. Theory*, vol. IT-24, pp. 530–536, 1978.