A Grammar-Based Compression Using a Variation of Chomsky Normal Form of Context Free Grammar

Mitsuharu Arimura
Faculty of Engineering
Shonan Institute of Technology
1–1–25 Tsujido Nishi-Kaigan, Fujisawa-shi,
Kanagawa-ken, 251–8511 Japan
Email: arimura@m.ieice.org

***Abstract*—This paper proposes a new class of grammar-based lossless source code. Grammar-based code is a class of universal data compression algorithm using a context-free grammar. A Semi-Chomsky Normal Form (semi-CNF) of context free grammar, which is a modified form of the context free grammar (CNF), is newly introduced. The proposed algorithm encodes a given sequence to a binary codeword in three step. In the first step, semi-CNF of the set of production rules is constructed using repeated substitution from a pair of symbols or variables to a new variable. In the second step, semi-CNF is translated to an irreducible or smaller grammar by eliminating production rules which are used only once in the other production rules. A produced grammar is encoded to a binary codeword in the third step. LZ78, Multilevel Pattern Matching (MPM) and Byte Pair Encoding (BPE) algorithms can be treated as examples of this class of codes. LZ78 and MPM algorithms does not use the second step of this procedure. Therefore, the proposed method can improve the compression performance of these algorithms by the unified procedure. This method has an advantage that, transformation from a given sequence to the grammar is quite simple, by using the three-step algorithm through semi-CNF.**

I. Introduction

A class of grammar-based data compression algorithms using context free grammar is formulated by Kieffer and Yang [1], and the same authors [2] proposes some compression algorithms. A known sequential grammar compression method is SEQUITUR [3], [4].

There exist more simple implementations which can be treated as examples of grammar based code, which are Lempel-Ziv 78 (LZ78) [5], Multilevel Pattern Matching (MPM) [6], Byte Pair Encoding (BPE) [7] (or Digram Coding, Re-Pair algorithm [8]). A common characteristic of these algorithms is that the right-hand sides of the production rules other than the start symbol rule are formed by two symbols. These algorithm are simple. From the viewpoint of formal language theory, there exists a notion of Chomsky Normal Form (CNF) of context free grammar. In this paper, we newly introduce a Semi-Chomsky Normal Form (semi-CNF), which is a grammar that the right-hand side of the production rules other than the start symbol rule has two symbols. Note that the class of CNF is a subclass of semi-CNF. Then, all the above algorithms are classified as the compression method using the semi-CNF of the context free grammar. However, they do not generally produce irreducible grammar which is defined in the formulation of grammar-based code [1]. This means that there exists a possibility to improve the compression performance of them.

This paper proposes a new three-step data compression algorithm which constructs a semi-CNF for a given sequence and then reduce to an irreducible or smaller grammar, which is encoded to a binary codeword by the sequential arithmetic coding. We show that LZ78 and MPM algorithms described above do not use the second step of this three-step coding algorithm, and confirm that the second step of the proposed algorithm can improve the compression performance of these algorithms by the computer simulation for sample files.

This paper is organized as follows. Section II describes a grammar-based code. In Section III, three algorithms are described, and it is demonstrated that these algorithm can construct a CNF/semi-CNF of context free grammar. The algorithm of the proposed three-step method is described in Section IV. Section V presents the simulation results and show the effectiveness of the proposed algorithm experimentally.

II. Context Free Grammar

In this section the context free grammar is defined. For any set A, A^* means the set of all the finite sequence from A. The length of a string s is represented by $|s|$ (for the empty string λ, $|\lambda| = 0$).

Definition 1 (Context Free Grammar [1], [9])**.** *A context free grammar is defined as $G = (V, T, P, S)$, each of which means the following, respectively.*

- *V is a set of variables.*
- *T is a finite set of terminate symbols ($T \cap V = \emptyset$).*
- *P is a finite set of production rules $A \to \alpha$, $A \in V, \alpha \in (V \cup T)^*$. Assume that for each $A \in V$, there exists one or more production rule which has A as a left-hand side (They are called* rule A's*).*
- *S is a start symbol.*

For a context free grammar G, each of V, T, P, S is written as $V(G)$, $T(G)$, $P(G)$, $S(G)$, respectively. For simplicity, the set $P(G)$ of production rules is called grammar G. Any sequence in $(V \cup T)^*$ is called *string*.

Subsequently, notations $\overset{G}{\to}$ and $\overset{G}{\Rightarrow}$ are defined.

- If two strings α and β satisfy $\alpha \overset{G}{\to} \beta$, there exists two strings α_1, α_2 and a production rule $A \to \gamma$ satisfying $\alpha = \alpha_1 A \alpha_2$ and $\beta = \alpha_1 \gamma \alpha_2$ (by replacing A with γ in α, we obtain β).
- If two strings α and β satisfy $\alpha \overset{G}{\Rightarrow} \beta$, there exists a sequence of strings $\alpha_1, \alpha_2, \ldots, \alpha_k$ satisfying $\alpha = \alpha_1 \overset{G}{\to} \alpha_2, \alpha_2 \overset{G}{\to} \alpha_3, \ldots, \alpha_{k-1} \overset{G}{\to} \alpha_k = \beta$.

The *language* which is generated by a context free grammar is defined by $L(G) = \{u \in T^* : S \overset{G}{\Rightarrow} u\}$.

A. Admissible Grammar

In this section, admissible grammar [1] is defined.

Definition 2 (Admissible Grammar [1])**.** *A context free grammar G is* admissible *if G satisfies the following conditions.*

a-1) *G is deterministic (for any variable $v \in V(G)$, there exists just one production rule, which has v as the left-hand side).*
a-2) *The right-hand side of any production rule is not the empty string.*
a-3) *The language $L(G)$ generated by G is not the empty set.*
a-4) *There exists no unused terminate symbol nor variable (for any symbol $Y \in V(G) \cup T(G), Y \neq S$, there exists a finite string $\alpha_1, \alpha_2, \ldots, \alpha_n$ which satisfies that Y occurs in some of them and $S = \alpha_1 \overset{G}{\to} \alpha_2, \alpha_2 \overset{G}{\to} \alpha_3, \ldots, \alpha_{n-1} \overset{G}{\to} \alpha_n \in L(G)$).*

B. Irreducible Grammar

In this section irreducible grammar [1] is defined.

Definition 3 (Irreducible Grammar [1])**.** *A context free grammar G is* irreducible *if G satisfies the following conditions.*

b-1) *G is admissible.*
b-2) *Any two different variables $v_1, v_2 \in V(G)$, $v_1 \neq v_2$ are expanded to different strings of terminate symbols.*
b-3) *Any variable other than the start symbol occurs two or more times in the right-hand side of production rules.*
b-4) *For any pair (Y_1, Y_2) of symbols (variables or terminate symbols), the string Y_1Y_2 does not occur in non-overlapping two or more positions in the right-hand side of production rules.*

If an admissible grammar G is not irreducible, G can be rewritten to another grammar G' which generates the same string as G and is nearer to the irreducible grammar than G, by applying the following *reduction rules*.

Reduction Rule 1: G is admissible. $B \to \alpha A \beta$ is the only one rule in which A occurs in the right-hand side. $A \to \gamma$ is the rule A. $\Longrightarrow$ Delete the rule $A \to \gamma$ and replace the rule $B \to \alpha A \beta$ with $B \to \alpha\gamma\beta$.

Reduction Rule 2: G is admissible. There exists a rule $A \to \alpha_1\beta\alpha_2\beta\alpha_3$ for some string β ($|\beta| \geq 2$). $\Longrightarrow$ For any variable $B \notin V(G) \cup T(G)$, replace the rule A with $A \to \alpha_1 B \alpha_2 B \alpha_3$ and create a new rule $B \to \beta$.

Reduction Rule 3: G is admissible. There exists two rules $A \to \alpha_1\beta\alpha_2$ and $B \to \alpha_3\beta\alpha_4$ for some string β ($|\beta| \geq 2$) such that $|\alpha_1|+|\alpha_2| > 0$, $|\alpha_3|+|\alpha_4| > 0$. $\Longrightarrow$ For any variable $C \notin V(G) \cup T(G)$, replace the rule A and B with $A \to \alpha_1 C \alpha_2$, $B \to \alpha_3 C \alpha_4$ and create a new rule $C \to \beta$.

Reduction Rule 4: G is admissible. There exists two rules $A \to \alpha_1\beta\alpha_2$ and $B \to \beta$ for some string β ($|\beta| \geq 2$) such that $|\alpha_1| + |\alpha_2| > 0$. $\Longrightarrow$ Replace the rule A with $A \to \alpha_1 B \alpha_2$.

Reduction Rule 5: G is admissible. There exists two different variables $A, B \in V(G)$ such that A and B generates the same string by applying the production rule $P(G)$. $\Longrightarrow$ Replace all the occurrences of A in the right-hand side with B, and delete unused production rules.

Using the above reduction rules as a practical data compression algorithm, there are some problems as follows.

1) In Reduction Rule 5, we need to keep the string of terminated symbols expanded from each variable, to check if all variables generate different strings. Moreover, if the coding proceeds, the strings becomes longer so that the comparison is burdensome.
2) In Reduction Rules 2 and 3, any string of length 2 or more which occurs in two positions of the right-hand sides of production rules. Since the length is not restricted, counting of all the occurrence numbers is not efficient in memory and time complexity.
3) If Reduction Rule 4 is used just after the rule B is created, it is not assured that all the occurrences of β in the right hand side of all the production rules are replaced with B all at once. This is a reason that Reduction Rule 5 is needed.

To overcome the last one, we take the following strategy.

Strategy 1: Every time after a new production rule is created using Reduction Rules 2 and 3, repeat Reduction Rule 4 as possible before applying the other reduction rule. This procedure can eliminate Reduction Rule 5.

However, taking this procedure for any length of string needs large time complexity. Consequently, we take the following additional strategy.

Strategy 2: A new production rule is created by Reduction Rules 2 and 3 only from strings β ($|\beta| = 2$).

By taking these two strategies, we can construct simple and efficient data compression algorithm.

C. Semi-Chomsky Normal Form

In this section Semi-Chomsky Normal Form (semi-CNF) is introduced. Before that, the definition of the Chomsky Normal Form (CNF) is presented.

Theorem 1 (Chomsky Normal Form Theorem of Context Free Grammar [9])**.** *Any context free language can be generated from a grammar with the production rules of the form $A \to BC$ or $A \to a$, where $A, B, C \in V$ and $a \in T$. Production rule set P of this form is called* Chomsky Normal Form (CNF).

Translating any context free grammar to a corresponding CNF is described as follows. First, for any rules including terminate symbols in the right-hand side $A \to B_0 a_1 B_1 a_2 \cdots a_i B_i$, prepare new variables and rules corresponding to each terminate symbols, and rewrite the grammar

$$A \to B_0 A_1 B_1 A_2 \cdots A_i B_i,$$
$$A_1 \to a_1,\ A_2 \to a_2,\ \ldots A_i \to a_i,$$

which is formed by the rules with the right-hand side string of variables only and the rules with the right-hand side of one terminate symbol. Next, for any rules with the right-hand side of three or more variables $A \to A_1 A_2 \cdots A_n$, prepare new variables and rewrite recursively as

$$A \to A_1 B_1,\ B_1 \to A_2 B_2,\ \ldots,$$
$$B_{n-3} \to A_{n-2} B_{n-2},\ B_{n-2} \to A_{n-1} A_n.$$

By the Reduction Rules from 2 to 4, some strings may not be converted to the CNF of the context free grammar. Even such cases, any string can be converted the following form of the grammar, which we call Semi-Chomsky Normal Form (semi-CNF) of the grammar.

Definition 4 (Semi-Chomsky Normal Form)**.** *The grammar is called* Semi-Chomsky Normal Form*, if the form of the production rules other than the start symbol rule is $A \to BC$ or $A \to a$, where $A, B, C \in V$ and $a \in T$.*

Note that the class of the CNF is a subclass of the semi-CNF. In the following section, we show that there are some algorithms generating the CNF/semi-CNF of context free grammar.

III. Data Compression Algorithms Generating CNF/semi-CNF

As a data compression algorithm which can be classified as a grammar-based code which can generate CNF/semi-CNF of context free grammar, LZ78 method, MPM coding and BPE coding has been proposed. Examples of generated grammars by these algorithms are presented in this section.

A. LZ78 Algorithm

LZ78 algorithm creates production rules of the form $A \to a$ or $A \to Ba, A, B \in V, a \in T$.

Example 1. *The grammar transform using LZ78 algorithm generates the following grammar from the string $x =$* 10110011110000111100001100011000*.*

$$S \to A_0 A_1 A_2 A_3 A_4 A_5 A_6 A_7 A_8 A_9 A_{10} A_{11} A_1,$$
$$A_0 \to 1, A_1 \to 0, A_2 \to A_0 1, A_3 \to A_1 0, A_4 \to A_2 1,$$
$$A_5 \to A_0 0, A_6 \to A_3 0, A_7 \to A_4 1, A_8 \to A_6 0,$$
$$A_9 \to A_2 0, A_{10} \to A_3 1, A_{11} \to A_5 0.$$

This grammar is semi-CNF and not irreducible, since in the variable set of the above production rules, each of the variables $A_7, A_8, A_9, A_{10}, A_{11}$ occurs only once in the right-hand side of production rules.

In production rules with the right-hand side of two symbols, replace 1 *and* 0 *with A_0 and A_1, respectively. Expanding the right-hand side of production rule S recursively, the following CNF is obtained.*

$$S \to A_0 S_0, S_0 \to A_1 S_1, S_1 \to A_2 S_2, S_2 \to A_3 S_3,$$
$$S_3 \to A_4 S_4, S_4 \to A_5 S_5, S_5 \to A_6 S_6, S_6 \to A_7 S_7,$$
$$S_7 \to A_8 S_8, S_8 \to A_9 S_9, S_9 \to A_{10} S_{10}, S_{10} \to A_{11} A_1,$$
$$A_0 \to 1, A_1 \to 0, A_2 \to A_0 A_0, A_3 \to A_1 A_1, A_4 \to A_2 A_0,$$
$$A_5 \to A_0 A_1, A_6 \to A_3 A_1, A_7 \to A_4 A_0, A_8 \to A_6 A_1,$$
$$A_9 \to A_2 A_1, A_{10} \to A_3 A_0, A_{11} \to A_5 A_1.$$

Since each of the variables $A_7, A_8, A_9, A_{10}, A_{11}$ occurs only once in the right-hand side, this grammar is not irreducible.

B. MPM Algorithm

MPM algorithm creates production rules of $A \to A_1 A_2, A_1, A_2 \in V$, where A_1 and A_2 correspond to strings of the same length of 2^n for some $n \geq 0$.

Example 2. *The grammar transform using MPM algorithm generates the following grammar from x given in Example 1.*

$$S \to A_{00} A_{01}, A_{00} \to A_{10} A_{11}, A_{01} \to A_{11} A_{12},$$
$$A_{10} \to A_{20} A_{21}, A_{11} \to A_{22} A_{21}, A_{12} \to A_{23} A_{24},$$
$$A_{20} \to A_{30} A_{31}, A_{21} \to A_{32} A_{31}, A_{22} \to A_{31} A_{32},$$
$$A_{23} \to A_{32} A_{33}, A_{24} \to A_{30} A_{32},$$
$$A_{30} \to A_{40} A_{41}, A_{31} \to A_{40} A_{40}, A_{32} \to A_{41} A_{41},$$
$$A_{33} \to A_{41} A_{40}, A_{40} \to 1, A_{41} \to 0.$$

This grammar is CNF. Since each of the variables A_{00}, A_{01}, A_{10}, A_{12}, A_{20}, A_{22}, A_{23}, A_{33} occurs only once in the right-hand side, this grammar is not irreducible.

C. Byte Pair Encoding / Digram Encoding Algorithm

Byte pair encoding creates production rules from the most frequently occurring pair of symbols recursively.

Example 3. *The grammar transform using BPE algorithm generates the following grammar from x given in Example 1.*

$$S \to A_0 A_5 A_7 A_7 A_4 A_5 A_1, A_0 \to 1, A_1 \to 0,$$
$$A_2 \to A_1 A_1, A_3 \to A_0 A_0, A_4 \to A_3 A_2, A_5 \to A_1 A_4,$$
$$A_6 \to A_3 A_4, A_7 \to A_6 A_2$$

This grammar is semi-CNF and not irreducible, since A_6 occurs only once in the right-hand side.

Expanding the rule S recursively, the following CNF is obtained.

$$S \to A_0 S_0, S_0 \to A_5 S_1, S_1 \to A_7 S_2, S_2 \to A_7 S_3,$$
$$S_3 \to A_4 S_4, S_4 \to A_5 A_1 A_0 \to 1, A_1 \to 0,$$

$$A_2 \to A_1A_1, A_3 \to A_0A_0, A_4 \to A_3A_2, A_5 \to A_1A_4,$$
$$A_6 \to A_3A_4, A_7 \to A_6A_2$$

This grammar is CNF. Since the variable A_6 occurs only once in the right-hand side, this grammar is not irreducible.

IV. Data Compression by Reduction of CNF/semi-CNF

Generally, for any context free grammar, there exists a CNF which generates the same language [9]. Since the right-hand side of CNF is at most 2, constructing a CNF of context free grammar is more simple and more efficient than constructing a general context free grammar. However, CNF of a context free grammar may not be irreducible.

On the other hand, for any admissible context free grammar, there exists a semi-CNF of irreducible grammar which generates the same language. If a semi-CNF of context free grammar is constructed from a given string using the Reduction Rules 2 to 4, and then Reduction Rule 1 is applied, it becomes irreducible.

Therefore, we propose the following algorithm.

Step 1 (Construction of semi-CNF/CNF): For a given string x, applying Reduction Rules 2 to 4 with restriction of $|\beta| = 2$, generate a CNF/semi-CNF $C(x)$ which generates only x as its language. We produce a new rule from the pair of symbols which occurs most frequently in the right hand side of rule S. Every time after a new production rule is created using Reduction Rules 2 and 3, repeat Reduction Rule 4 as many times as possible before proceeding to the next reduction.

Step 2 (Reduction): Reduce $C(x)$ to another grammar $G(x)$ using Reduction Rule 1.

We experimented the following two variations of Step 2.

Reduction Type 1: Apply Reduction Rule 1 to the right-hand side of the rule S only (apply Reduction Rule 1 to a variable occurring only once in the right-hand side of rule S and not occurring in the other rules).

Reduction Type 2: Apply Reduction Rule 1 to the right-hand side of all the production rules (apply Reduction Rule 1 to a variable occurring only once in the right-hand side of all the rules).

Reduction Type 2 corresponds to the grammar-based code defined by [1], [2]. The grammar generated by Reduction Type 1 may not be irreducible. However, since the lengths of the right-hand side of all the rules other than the start symbol rule are 2, coding procedure to the binary codewords can be simple and effective than taking Reduction Type 2.

In the following, examples of reduction to the three algorithm are presented.

A. LZ78 Algorithm

Example 4. *Applying Reduction Rule 1 to all the production rules in Example. 1, the following grammar is obtained.*

$$S \to 1A_1A_2A_3A_4A_5A_6A_41A_60A_20A_31A_50A_1$$
$$A_1 \to 0, A_2 \to 11, A_3 \to 00, A_4 \to A_21, A_5 \to 10,$$
$$A_6 \to A_30.$$

In LZ78 algorithm, there is a case such that a new block is not parsed in the parsing of the last block. In this case, the last block is equal to another previous block. If we reduce the last A_1, we obtain the following irreducible grammar.

$$S \to 10A_2A_3A_4A_5A_6A_41A_60A_20A_31A_5A_3$$
$$A_2 \to 11, A_3 \to 00, A_4 \to A_21, A_5 \to 10, A_6 \to A_30.$$

B. MPM Algorithm

Example 5. *Applying Reduction Rule 1 to all the production rules in Example. 2, the following grammar is obtained.*

$$S \to A_{30}A_{31}A_{21}A_{11}A_{11}A_{32}A_{41}A_{40}A_{24},$$
$$A_{11} \to A_{31}A_{32}A_{21}, A_{21} \to A_{32}A_{31}, A_{24} \to A_{30}A_{32},$$
$$A_{30} \to A_{40}A_{41}, A_{31} \to A_{40}A_{40}, A_{32} \to A_{41}A_{41},$$
$$A_{40} \to 1, A_{41} \to 0.$$

Removing the rules $A_{40} \to 1$ and $A_{41} \to 0$, we have the following irreducible grammar.

$$S \to A_{30}A_{31}A_{21}A_{11}A_{11}A_{32}01A_{24}, A_{11} \to A_{31}A_{32}A_{21},$$
$$A_{21} \to A_{32}A_{31}, A_{24} \to A_{30}A_{32}, A_{30} \to 10, A_{31} \to 11,$$
$$A_{32} \to 00.$$

C. Byte Pair Encoding / Digram Encoding Algorithm

Example 6. *Applying Reduction Rule 1 to all the production rules in Example. 3, the following grammar is obtained.*

$$S \to A_0A_5A_7A_7A_4A_5A_1, A_0 \to 1, A_1 \to 0, A_2 \to A_1A_1,$$
$$A_3 \to A_0A_0, A_4 \to A_3A_2, A_5 \to A_1A_4, A_7 \to A_3A_4A_2.$$

Removing the rules$A_0 \to 1$ and $A_1 \to 0$, we have the following irreducible grammar.

$$S \to 1A_5A_7A_7A_4A_50, A_2 \to 00, A_3 \to 11, A_4 \to A_3A_2,$$
$$A_5 \to 0A_4, A_7 \to A_3A_4A_2.$$

D. Binary Encoding Algorithm

Reduced forms of CNF/semi-CNF grammar generated from a given string are encoded to the binary codeword using the Hierarchical Algorithm [2] as follows.

1) Arrange the production rules from top to bottom, such that, when we scan the variables in right-hand side and left-hand side, respectively, the order of the first occurrence position of each variable in the right-hand side is the same order of the left-hand side.
2) Renumber the variable in the order of the first occurrence position of the right-hand side.
3) Add the symbol $e \notin T(G) \cup V(G)$ to the tail of the right-hand side of rule S. Add the symbol $b \notin T(G) \cup V(G)$ in the head, and e in the tail, respectively, of the right-hand side of the other rules which is longer than 2. Concatenate all the right-hand sides of the production rules from top to bottom.

TABLE I
COMPRESSION RESULTS OF IRREDUCIBLE AND ORIGINAL ALGORITHMS (BYTES)

file	size	BPE (semi-CNF)	BPE CNF	BPE Red. Type 1	BPE Red. Type 2	LZ78 (semi-CNF)	LZ78 CNF	LZ78 Red. Type 1/2	MPM (CNF)	MPM Red. Type 1	MPM Red. Type 2	gzip -9	bzip2 -9
bib	111261	**30440**	35008	**30440**	31036	65479	69286	**60130**	61568	**51135**	51681	34896	27467
book1	768771	255169	295373	**255161**	256340	431809	454023	**395568**	417589	**346298**	347034	312275	232598
book2	610856	**174383**	199556	**174383**	176287	337384	354855	**310830**	327759	**274328**	275763	206152	157443
geo	102400	**60003**	73877	**60003**	60051	81651	86640	**70241**	78760	**62814**	62992	68410	56921
news	377109	**123640**	140499	123641	124916	239125	251721	**218331**	232233	**193324**	194705	144395	118600
obj1	21504	10460	12970	**10459**	10633	17969	19421	**15928**	17317	**13686**	13842	10315	10787
obj2	246814	79780	91258	**79773**	81166	170050	179206	**155119**	144090	**119997**	121357	81082	76441
paper1	53161	**17529**	20253	**17529**	17969	35518	37766	**32210**	34236	**28057**	28322	18536	16558
paper2	82199	**26641**	30976	**26641**	27120	50953	54063	**46238**	48970	**40068**	40349	29660	25041
paper3	46526	**16783**	19555	16784	17156	31140	33150	**28062**	30027	**24320**	24510	18067	15837
paper4	13286	**5472**	6521	**5472**	5616	9751	10510	**8711**	9414	**7489**	7562	5527	5188
paper5	11954	5058	6037	**5057**	5228	9163	9894	**8160**	8838	**7035**	7116	4988	4837
paper6	38105	**13215**	15322	**13215**	13516	26302	28036	**23778**	25132	**20494**	20721	13206	12292
pic	513216	51873	61815	**51869**	52371	79260	84064	**71316**	76369	**60883**	61475	52377	49759
progc	39611	**13125**	15196	**13125**	13441	27433	29221	**24765**	26277	**21430**	21649	13255	12544
progl	71646	16599	18807	**16598**	16955	40041	42524	**36702**	36838	**30548**	30941	16158	15579
progp	49379	**10875**	12328	10876	11021	28620	30461	**26180**	26732	**22170**	22550	11180	10710
trans	93695	**18605**	20630	**18605**	18663	55848	59138	**51698**	50412	**42621**	43308	18856	17899

4) For each variable $A_i \in V(G)$, replace the first occurrence of A_i with $A \notin T(G) \cup V(G)$.
5) Encode the sequence by the adaptive arithmetic coding. The initial count of symbol $a \in T(G) \cup \{A, b, e\}$ is set to 1. After the i-th occurrence of A, the count of A_i is set to 1.

As the arithmetic coding, we used the carry-free version (cf. [12]) of Range Coder [10], [11] with 64bit calculation.

V. COMPRESSION RESULT

Table I shows the compression result of the proposed algorithm using BPE, LZ78 and MPM algorithms, and gzip/bzip2 program for sample files of Calgary Corpus [13]. For each algorithms, compression results for Reduction Type 1, 2 and no reduction are presented. The boldface items are the best result in each algorithm. For LZ78 algorithm, there is no difference between the two reductions, because all the variables in the right hand side of the rules other than S also occur in the right hand side of the rule S.

By the Type 1 reduction, codeword length is smaller than or equal to the coding without reduction for almost cases. This proves the effectiveness of the Type 1 Reduction to the CNF/semi-CNF of the grammar-based code. Results of Type 2 reduction, corresponding to the grammar-based code [1], [2], are worse than Type 1 reduction, since the length of the right-hand side of some rules other than S becomes 3 or more by Type 2 reduction. In this case, we need to add the markers b and e to specify the border of the rules, which expands the number of the encoded symbols.

VI. CONCLUDING REMARKS

We proposed a new class of grammar-based data compression algorithm which reduces the Chomsky Normal Form and its variation of the context free grammar. LZ78, MPM and BPE are treated as examples of this class of codes. By indicating that the former two algorithms do not use some step of the proposed algorithm, we showed that there is a room to improve the compression performance of these algorithm.

ACKNOWLEDGMENT

This research is supported in part by MEXT Grant-in-Aid for Scientific Research(C) 15K06088.

REFERENCES

[1] John C. Kieffer and En-hui Yang, "Grammar-Based Codes: A New Class of Universal Lossless Source Codes," *IEEE Trans. Inform. Theory*, Vol.46, No.3, pp.737–754, May 2000.
[2] En-hui Yang and John C. Kieffer, "Efficient Universal Lossless Data Compression Algorithms Bases on Greedy Sequential Grammar Transform — Part One: Without Context Models," *IEEE Trans. Inform. Theory*, Vol.46, No.3, pp.755–777, May 2000.
[3] Craig G. Nevill-Manning and Ian H. Witten, "Idenfifying Hierarchical Structure in Sequences: A linear-time algorithm," *Journal of Artificial Intelligence Research*, Vol. 7, pp. 67–82, 1997.
[4] Craig G. Nevill-Manning and Ian H. Witten, "Compression and Explanation Using Hiearchical Grammars," *Computer Journal*, Vol. 40, No. 2/3, pp. 103–116, 1997.
[5] Jacob Ziv and Abraham Lempel, "Compression of Individual Sequences via Variable-Rate Coding," *IEEE Trans. Inform. Theory*, Vol.IT-24, No.5, pp.530–536, Sept. 1978.
[6] John C. Kieffer, En-hui Yang, Gregory J. Nelson and Pamela Cosman, "Universal Lossless Compression Via Multilevel Pattern Matching," *IEEE Trans. Inform. Theory*, Vol.46, No.4, pp.1227–1245, July 2000.
[7] Philip Gage, "A New Algorithm for Data Compression," *The C Users Journal*, R & D Publications, Inc. Lawrence, KS, USA, Vol.12, Issue.2, pp.23–38, Feb. 1994.
[8] N. J. Larsson and A. Moffat, "Offline Dictionary-based Compression," *Proc. of the IEEE*, Vol.88, pp.1722-1732, 2000.
[9] John E. Hopcroft and Jeffrey D. Ullman, *Formal Languages and their Relation to Automata*, Addison-Wesley, 1969.
[10] G. N. N. Martin, "Range Encoding: An Algorithm for Removing Redundancy from a Digitized Message," *Video and Data Recording Conference*, Southampton, pp.24-27, 1979.
[11] Michael Schindler, "A Fast Renormalization for Arithmetic Coding," *Proc. 1998 IEEE Data Compression Conference (DCC98)*, p.572, Snowbird, Utah, 1998.
[12] `http://www.geocities.ws/mikaellq/`
[13] The Canterbury Corpus, `http://corpus.canterbury.ac.nz/`