An Inference Method for Non-Recursive Context-Free Grammars

Chaiyaporn Chirathamjaree

Edith Cowan University
c.chirathamjaree@ecu.edu.au

ABSTRACT

A method is presented for the generation of non-recursive context-free grammars (CFG’s) from a set of strings that the CFG’s are required to be able to produce. The method generates compact CFG’s having a near minimal number of rules and non-terminals, compatible with the requirement to be able to generate all the strings in the example set. This method produces grammars with a reasonable amount of computation compared with enumerative methods.

Keywords: Formal grammars, Context-Free grammars, Recognition systems, Grammar inference, Feature extractor.

1. INTRODUCTION

Following the common practice in the field of pattern recognition [1], a typical recognition system can be considered to consist of a feature extractor (FE) or a pre-processor of some sort followed by a recognizer or classifier (Figure 1). The FE transcribes the input into strings of symbols representing various parameters extracted from the input. A decision is made by the recognizer on this simplified representation as to which input item, if any, has been produced.

![Fig. 1: A Recognition System](image)

The classical decision-theoretic methods [1,2] from the field of pattern recognition have commonly been used to produce recognizers for processing strings of symbols generated by the FE. Another promising approach, which stemmed from the fields of mathematical linguistics and computer science, is to make use of the techniques of formal language theory [3]. The essence of this method is to generate a grammar for each class of patterns to be recognized. The classification of a pattern is done by determining which of a number of formal grammars [4] or sets of rules could have generated it. In some applications, intelligence can be applied to generate a suitable set of grammars. However, in many applications, such as automatic recognition of speech, this may be difficult due to the poorly understood underlying mechanism of generating patterns. In such cases, syntactic methods using formal grammars can be used provided that suitable sets of training strings are available.

In applications where there is a lack of proper understanding of the string generation process, the use of judgment is necessary to select the class of grammars to be used. In principle, the use of finite-state grammars (FSG’s) may be adequate to model the FE where finite-length strings are involved. However, the use of less-restricted grammars may provide models which are preferable in some way. For example, a model constructed on the basis of context-free grammars (CFG’s) may have fewer nodes and/or links than a FSG-based model using the same data. The CFG approach also enables useful complexity to be generated without requiring inordinate computing power. In some applications, such as automatic recognition of speech, where the strings may be expected to consist of concatenated substrings, the use of non-recursive CFG’s would provide a suitable model.

This paper presents the application of linguistic methods to a recognition system. It describes a method for the direct construction of non-recursive CFG’s from a set of example strings. The method generates compact CFG’s having a near minimal number of rules and non-terminals, compatible with the requirement to be able to generate all the strings in the example set. This method produces grammars with a reasonable amount of computation compared with enumerative methods. The procedure is incremental in the sense that, if a new string is added to the sample set, an updated grammar, if necessary, is generated with no need to reprocess the strings in the original sample set.

2. THE INFERENCE OF NON-RECURSIVE CFG’S

The problem of designing a recognizer in a recognition system can be broadly divided into two areas: the construction of models based on formal grammars, known as grammar inference [5,6,7], to represent characteristics of the symbol-generating source and the search for suitable decoding methods for efficiently analyzing the strings from the source using rules or grammars of the models previously created.

In outline, the basis of the linguistic method is simply explained. The first grammar, called the skeleton grammar $G_0$, is constructed from the first string in the sample set such that $G_0$ can generate only that string.
Other strings are then individually processed in the search for incompatibility between each string and the current grammar. This is done by applying the inference procedure recursively. If the nth observed string, Sn, can be derived from the (n-1)th inferred grammar Gn-1, then Gn = Gn-1 and no augmentation of Gn-1 is required. Otherwise, Gn-1 is augmented such that Gn is produced which can generate the present string. The matching process involves the computation of the weighted minimization matrix, (the wM-matrix described in section 3) whose elements reveal the shortcomings of the CFG in relation to its ability to generate the string.

Some notation and terminology are now introduced. Precise and complete definition of context-free grammars (CFG’s) are available elsewhere [3,4,8]. A CFG consists of:

1. A finite set of non-terminals A1, A2, ..., As;
2. A finite set of terminals b1, b2, ..., br;
3. A set of rewriting rules: A → B; where A is a non-terminal and B is a non-empty string of terminals or non-terminals, or both;
4. A start symbol, which is one of the non-terminals.

Any CFG can be transformed into an equivalent Chomsky normal form [4] in which the rules are of the following forms only:

Bielement rules: 

\[ A \rightarrow BC \]

Terminating rules: 

\[ A \rightarrow a \]

where A, B and C are non-terminals and a is a terminal.

### 3. COMPUTATION OF THE WEIGHTED MINIMIZATION MATRIX

Before the inference method can be given, it is necessary to describe the weighted minimization matrix, (wM-matrix), wM, which forms the basis of the inference process.

Firstly, the Levenshtein Distance (LD) [9] is defined as the minimum number of symbol alterations (insertions, deletions or substitutions) needed to convert an observed string x to a prototype string y. If various weights are assigned to each of the symbol alterations, the corresponding distance becomes the weighted Levenshtein Distance (wLD).

For a given CFG in Chomsky normal form and for a string, S, the weighted minimization matrix, (wM-matrix), wM, is a 3-dimensional, n x n x r matrix where n is the length of S and r is the number of non-terminals in the grammar. Let aj represent the jth symbol of S. Element mjk of wM denotes the wLD between an observed length-i substring of S, whose first symbol is aq, and a prototype string y generated from Aq, the kth non-terminal.

As an example, let A2 generates a string cd and S = dede. Given that the significance of ‘d’ and ‘e’ are -4 and -5 respectively. In this case, m123 = [significance of ‘e’] + [significance of ‘d’] = 5+4=9. Aq is arbitrarily selected from the non-terminals to represent the starting symbol, so that the element m10i is 0 if, and only if, the string S can be generated by the grammar.

The weighted hierarchy level (wHL) of a non-terminal Ak (denoted by wH(k)) is the sum of the absolute values of the significance of all symbols in a string derivable from Ak. The procedure for computing the wHL of non-terminals in a CFG is now given.

**wHL of non-terminals in terminating rules:**

\[ wH(k) = |\text{significance of } a_k | \]

for a rule \( A_k \rightarrow a_k \) in the CFG.

**wHL of non-terminals in bielement rules:**

\[ wH(k) = \min_{p, q \in P_k} \left[ wH(p) + wH(q) \right] \]

where \( P_k \) is the set of ordered pairs (p,q) such that \( A_k \rightarrow A_p A_q \)

The above follows because non-terminal \( A_k \) is replaced by non-terminals \( A_p \) and \( A_q \) via a bielement rule of the form \( A_k \rightarrow A_p A_q \), so that wHL’s corresponding to \( A_p \) and \( A_q \) need to be added. The final result is the smallest of this sum taken over all rules in the set \( P_k \).

wM-matrix can be computed iteratively by the following procedure. The procedure is in two parts; one for the terminating rules and one for the bielement rules.

**Part 1: Terminating rules**

\[ m_{ijk} = |\text{significance of } b_j - \text{significance of } a_k | \]

for a rule \( A_k \rightarrow a_k \) in the CFG.

This follows from the definitions of wM-matrix and of terminating rules.

For \( i=2,3,...,n \):

\[ m_{ijk} = \sum_{u=1}^{q} |\text{significance of } b_u | \]

\[ - |\text{significance of } b_u | \}

\[ (4) \]

The above follows because a substring of length i, whose first symbol starts at position j, can be considered as to consist of i sub-substrings, each of unit length. Only one of these sub-substrings can be matched against \( a_k \), the terminal in a rule \( A_k \rightarrow a_k \) in the CFG.

**Part 2: Bielement rules**

\[ m_{ijk} = \min_{p, q \in P_k} \left[ \min_{1 \leq u \leq l} |m_{ijp} + wH(q), m_{ijq} + wH(p)| \right] \]

\[ (5) \]

where \( P_k \) is the set of ordered pairs (p,q) such that \( A_k \rightarrow A_p A_q \)

For \( i=2,3,...,n \):

\[ m_{ijk} = \min_{p, q \in P_k} \left[ \frac{x}{1+u} \cdot \min_{1 \leq u \leq l} |m_{ijp} + m_{ijq} + wH(p)| \right] \]

\[ (6) \]

where \( x = |m_{ijp} + wH(q), m_{ijq} + wH(p)| \)
An example of the wM-matrix for the string \( S = BjC \) is given in table 1.

### Table 1: wM-Matrix for a string \( S = BjC \) and the grammar \( G_a \):

<table>
<thead>
<tr>
<th>i</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>j</td>
<td>B</td>
<td>C</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>A_1</td>
<td>7</td>
<td>5</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>A_2</td>
<td>3</td>
<td>15</td>
<td>2</td>
<td>13</td>
</tr>
<tr>
<td>A_3</td>
<td>10</td>
<td>2</td>
<td>11</td>
<td>4</td>
</tr>
<tr>
<td>A_4</td>
<td>2</td>
<td>14</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>A_5</td>
<td>0</td>
<td>12</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>A_6</td>
<td>12</td>
<td>0</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>A_7</td>
<td>1</td>
<td>13</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>A_8</td>
<td>8</td>
<td>10</td>
<td>7</td>
<td>12</td>
</tr>
<tr>
<td>A_9</td>
<td>16</td>
<td>12</td>
<td>15</td>
<td>10</td>
</tr>
</tbody>
</table>

### Grammar \( G_a \):

\[
G_a: \quad A_1 \to c \quad A_{31} \to A_1A_2 \quad A_r \to A_{32}A_r \\
A_2 \to E \quad A_{32} \to A_3A_3 \quad A_r \to A_3A_3 \\
A_3 \to h \quad A_{33} \to A_3A_7 \\
A_4 \to D \\
A_5 \to B \\
A_6 \to j \\
A_7 \to C
\]

### 4. CFG INFECTION PROCEDURE

The symbol strings in the sample set are assumed to be arbitrarily labeled \( S_1, S_2, \ldots, S_m \). \( S_i \) is processed to generate the initial CFG, \( G_1 \). Other CFG’s are inferred recursively, as follows.

#### Formation of \( G_{i+1} \) from \( G_i \):

The first stage is the computation of the wM-matrix for the string \( S_{i+1} \) and the CFG \( G_i \). For a string \( S_{i+1} = a_{i+1}a_{i+2} \ldots a_n \), if \( m_{a_i} = 0 \), \( S_{i+1} \) can be generated by \( G_i \), and no additions to \( G_i \) are required, i.e. \( G_{i+1} = G_i \). Otherwise \( G_{i+1} \) is formed by augmenting \( G_i \) with new terminals, non-terminals and rules so that \( G_{i+1} \) can generate \( S_{i+1} \).

Terminating rules are formed by the following procedure:

For \( i = 1 \) to \( n \):

Create a new non-terminal \( A_{a_i} \) and a new terminating rule \( A_{a_i} \to a_i \), if, and only if, there does not already exist a rule with \( a_i \) at its RHS. (\( A_{a_i} \) is the non-terminal corresponding to \( a_i \)).

New bielement rules are formed as follows:

Step 1. Select a set of indices \( j \).

For \( i = n-1 \):

Select \( j_{n-1} \) as the least \( j \) for which \( m_{a_{n-1}j} \) is minimum.

For \( i = (n-2), (n-3), \ldots, 2 \):

Select \( j_i \), the least \( j \) for which \( m_{a_{i}j} \) is minimum and for which \( j_i \geq j_{i+1} \). Each \( j_i \) is the \( j \) index of \( m_{a_{i}j} \) corresponding to a substring of length \( i \).

Step 2. Form the new rules:

For \( i = 2 \):

Create a new non-terminal \( A_{a_i} \) and a new rule \( A_{a_i} \to A_{b_{i}a_{i+1}} \) unless these non-terminals and the rule have already been created.

For \( i = 3 \) to \( n \):

If \( j_{i-1} > j_i \), form a new bielement rule \( A_{a_i} \to A_{b_{i}a_{i+1}} \) unless there is already a rule of the form \( A_{a_i} \to A_{b_{i}a_{i+1}} \). (\( B \) represents an arbitrary non-terminal).

If \( j_{i-1} = j_i \), form a new bielement rule \( A_{a_i} \to A_{b_{i}A_{i-1}a_{i+1}} \) unless there is already a rule of the form \( A_{a_i} \to A_{b_{i}A_{i-1}A_{i+1}} \).

#### 5. RESULTS AND DISCUSSION

Consider the following set of sample strings:

\( S_1 = ]^{sauau} \)
\( S_2 = ]^{fsau} \)
\( S_3 = ]^{sfu} \)
\( S_4 = ]^{saiua} \)

The following illustrates the operation of the method.

Rules added at a given stage are represented by **bolded** characters.

Stage 1. Formation of \( G_1 \) directly from \( S_1 \):

\( G_1: \quad A_1 \to s \quad A_{12} \to A_1A_2 \quad A_r \to A_{14}A_3 \\
A_2 \to a \quad A_{12} \to A_2A_3 \\
A_3 \to u \quad A_{14} \to A_3A_2 \\
A_4 \to f \)

Stage 2:

\( G_2: \quad A_1 \to s \quad A_{12} \to A_1A_2 \quad A_r \to A_{14}A_3 \\
A_2 \to a \quad A_{13} \to A_2A_3 \\
A_3 \to u \quad A_{14} \to A_3A_2 \\
A_4 \to f \\
A_{16} \to A_4A_1 \\
A_{16} \to A_4A_3 \)

At the end of this stage, \( G_4 \) predicts three additional strings, namely, saau, saiau, and safu.

An incremental method for the construction of non-recursive CFG’s has been described. This is appropriate for applications such as the recognition of isolated words, where only finite-length strings are involved. It is possible to generate grammars for non-finite length strings by modifying the way new terminals and rules are created. One way is to remove the restriction on the form of production allowed, for example by allowing productions of the form \( A_{a_i} \to A_{a_i}A_{a_i} \).

The method presented yields non-recursive CFG’s that are able to generate all the given strings, irrespective of the order in which they are presented. Other strings
generated by the grammar will resemble those in the training set. The method produces compact CFG’s having near minimal number of rules and non-terminals. These are the consequences of the way in which the grammar is augmented at each stage so that the additions represent minimal change.

The method has been applied to the recognition of isolated spoken words. The vocabulary consists of ten digits ‘ZERO’ to ‘NINE’ uttered by a single speaker. The speech signal of the spoken digits is of telephone-grade quality. This is obtained from a normal telephone set via a circuit representing two limiting local lines. Ten CFG’s for the recognition of the spoken digits ‘ZERO’ to ‘NINE’ were generated from a total of 100 strings, of average length 3.6 symbols. The testing set consists of 500 strings in total with 50 strings for each of the ten digits spoken. The average length of testing strings is 3.4 symbols per string. The complexity measure of the inferred CFG’s is shown in table 2. Recognition performance was equal to that obtained using FSG’s. This is expected as the processing involved short strings with no significant underlying syntactic structure.

The representation of strings by a set of rules of formal grammars instead of direct storage of strings make possible the ‘generalization’ of strings in the training set. This reduces the size of the training set needed compared with the approach of using template matching techniques in order to cover the same number of strings.

<table>
<thead>
<tr>
<th>Words spoken</th>
<th>No. of terminals</th>
<th>No. of non-terminals</th>
<th>No. of rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>ONE</td>
<td>13</td>
<td>4</td>
<td>24</td>
</tr>
<tr>
<td>TWO</td>
<td>10</td>
<td>3</td>
<td>22</td>
</tr>
<tr>
<td>THREE</td>
<td>11</td>
<td>3</td>
<td>21</td>
</tr>
<tr>
<td>FOUR</td>
<td>19</td>
<td>14</td>
<td>42</td>
</tr>
<tr>
<td>FIVE</td>
<td>27</td>
<td>29</td>
<td>65</td>
</tr>
<tr>
<td>SIX</td>
<td>18</td>
<td>26</td>
<td>56</td>
</tr>
<tr>
<td>SEVEN</td>
<td>18</td>
<td>19</td>
<td>46</td>
</tr>
<tr>
<td>EIGHT</td>
<td>21</td>
<td>21</td>
<td>53</td>
</tr>
<tr>
<td>NINE</td>
<td>12</td>
<td>3</td>
<td>23</td>
</tr>
<tr>
<td>ZERO</td>
<td>15</td>
<td>39</td>
<td>63</td>
</tr>
</tbody>
</table>

6. CONCLUSION
Techniques of formal language theory or the syntactic methods provide a useful approach to the solution of classification and description in a recognition system where only a finite number of features (represented by symbols) are generated for each input item. The method presented for the generation of CFG’s from sample strings enables grammars of useful complexity to be generated without requiring inordinate computing power. Extension of the method to the construction of stochastic CFG’s, by counting the frequency of use of rules, is straightforward. Future work is planned to exploit the capabilities of CFG’s, in particular, the facility that processing may be started from any point in the string and may proceed in either direction. This may be appropriate for the recognition of words in connected speech.

7. REFERENCES