SIGNAL PROCESSING IV: Theories and Applications
J.L. Lacoume, A. Chehikian, N. Martin, and J. Malbos (Editors)
Elsevier Science Publishers B.V. (North-Holland)




# FINITE STATE GRAMMAR INFERENCE FOR CONNECTED WORD RECOGNITION

José B. Mariño, Climent Nadeu, Eduardo Lleida

Dpto. de Teoría de la Señal y Comunicaciones
Universidad Politécnica de Cataluña. Barcelona, Spain

Abstract

An algorithm that infers a finite state grammar from a language specified by simple enumeration of legal phrases, is introduced. The algorithm processes every phrase as a string of phonetic units (words, syllables, demisyllables, etc.), associates one or more states to each phonetic unit and establishes the set of predecessors (and the set of successors) for every state. As a fundamental feature, the algorithm includes an optimization step in order to minimize the number of states. Additionally, the algorithm can deal with languages specified by a combination of other languages or, even, by a context-free grammar.

## 1. INTRODUCTION

It has been reported /1/, /2/ that finite state grammars are a very convenient way to specify language structure to connected word recognition algorithms. Every phonetic unit (word, syllable, demisyllable, etc.) is associated to a state; thus, a phrase of the language can be described as a legal succession of transitions between states. In general, a phonetic unit is codified with several different states in order to account for the different contexts in which it may appear. In the present communication the grammar is described in terms of its states and the predecessor and successor states of all of them; although the description with predecessors or successors are equivalent, both two descriptions are used on behalf of the inference algorithm.

## 2. DEFINITIONS AND NOMENCLATURE

For clarity's sake, a brief glossary of nomenclature and definitions follows:

Phonetic unit: every recognition unit used to consider the different sounds in the language; it will be denote with $u_j$, $j=1,\ldots,N$.

Copy: every state associated to the same phonetic unit; the l-th copy of $u_j$ will be expressed by $u_j^l$.

Syntactic unit: each state of the grammar; additionally to the copies of phonetic units, it is necessary to consider the origen ($u_o$) and the end ($u_\infty$) of strings as syntactic units.

Predecessor (successor): a state k is called predecessor (successor) of another state j, if the latter (former) can be reached from the former (latter) with just one transition. The set of predecessors (successors) of a given syntactic unit $u_j^l$ will be denoted by $P_j^l(S_j^l)$. The set $P_j$ of predecessors ($S_j$ of successors) of $u_j$ is the union of the predecessor (successor) sets of its copies; hence, $P_j^l(S_j^l)$ are subsets of $P_j(S_j)$.

Additionally, the i-th phonetic unit of string k will be indicated with $u_{jki}$.

## 3. THE ALGORITHM

The algorithm operates on languages specified by simple enumeration and processes them through two steps. In the first one, it is infered a finite state grammar that generates just the specified phrases or strings of phonetic units; no string outside the language can be obtained from this grammar; the rules applied in this step address only this goal. The second step optimizes this preliminary grammar by an iterative procedure in order to minimize the number of states; this optimization is necessary because the size of the grammar built by the first step is highly dependent on the order that the strings are in the language; grammars obtained by the entire algorithm do not suffer this drawback.

The description of the algorithm will be illustrated with examples from the following language of alphabetic strings:

L= {abc,abd,abf,cabc,eabc,fabd,fabf}

STEP ONE processes sequentially the strings in order to establish the necessary states that account for every phonetic unit and its different

This work was supported by the CAICYT grant number 21096/84.

contexts (copies); at the same time, the corresponding sets of predecessors and successors are set up. The processing rules are the following:

rule 1: If a phonetic unit appears several times in a string, every occurrence must be associated to a different copy.

The reason of this rule can be well understood from the string "cabc" in L. If the state for the former and the latter "c" was the same, the grammar could generate the strings "cabc(abc)$^m$" that are not in L.

rule 2: The same copy of a phonetic unit may be shared by several strings, provided that every string exhibits the same substring from the copy to a specified ending (i.e. the shared substring must always be predecessor or always be successor of the phonetic unit).

In L the strings "abc", "abd" and "abf" may share the same copy of unit "b" (and of unit "a", too). On the contrary, the syntactic unit for "b" in the string "cabc" must be a copy different from the previous one; if the same state accounted for "b" in those four strings, it is easy to realize that the grammar would additionally issue the strings "cabd" and "cabf", not necessarily legal members of the language (indeed, these strings are not included in our example).

$a^o$ $b^o$ $c^o$
$a^o$ $b^o$ $d^o$
$a^o$ $b^o$ $f^o$
$c^1$ $a^1$ $b^1$ $c^o$
$e^o$ $a^1$ $b^1$ $c^o$
$f^1$ $a^2$ $b^2$ $d^o$
$f^1$ $a^2$ $b^2$ $f^o$

| P | copy | S |
|---|---|---|
| $u_o$ | $a^o$ | $b^o$ |
| $c^1, e^o$ | $a^1$ | $b^1$ |
| $f^1$ | $a^2$ | $b^2$ |
| $a^o$ | $b^o$ | $c^o, d^o, f^o$ |
| $a^1$ | $b^1$ | $c^o$ |
| $a^2$ | $b^2$ | $d^o, f^o$ |

Figure 1. Copy labeling after applying step 1 of the algorithm to language L and the resulting sets of predecessors P and successors S for the copies of units "a" and "b" ($u_o$ denotes the origen of strings).

In order to be operative, these two rules have to be arranged in a algorithmic way. A straight forward implementation processes every string in the following way:

1.- At first, the copy with label l=0 is assigned to every phonetic unit.

2.- The string is scanned from left to right. For every syntactic unit $u^l_{jki}$ the following operations are carried out:

a) It is tested whether this unit has come out when processing previous strings, or not. If the syntactic unit is new, the algorithm leaves the scanning and passes to 3 ; the position of the unit in the string is conserved by the variable "left-end". Otherwise, the algorithm goes on the following pass.

b) According to rule 1, if the same syntactic unit has already appeared in the string, the copy label is incremented by one and the algorithm comes back to 2.a . In other case, the next operation is performed.

c) In order to apply rule 2, the current predecessor subset of $u^l_{jki}$ is tested looking for $u^l_{jk(i-1)}$, the unit that preceder $u^l_{jki}$ in the string. If it is not found, the label l is increased by one and the algorithm addresses back to 2.a . When $u^l_{jk(i-1)}$ is found, it must be checked if $u^l_{jki}$ has been shared on its right (share-r $(u^l_{jki})$=1); if this is the case, the label is incremented and the process goes back to 2.a ; otherwise (share-l $(u^l_{jki})$=0), it is concluded that $u^l_{jki}$ is the state corresponding the phonetic unit under processing, share-r $(u^l_{jki})$ is set to one in order to remember that the unit has been shared on its left, and the algorithm goes back to 2 to processes the following unit in the string.

3.- The string is processed from right to left in a similar way to that performed in pass 2 . Now, the successors are analized instead of the predecessors. The scanning finishes at the position "right-end". The algorithm advances to the final pass 4 .

4.- The substring from "left-end" to "right-end" contains the new syntactic units; theirs respective sets of predecessors and successors are initialized. Lastly, the set of successors of the unit in position "left-end-1" and the set of predecessors of the unit in position "right-end+1" are actualized.

The application of this algorithm to the language L provides the copy labeling showed in Figure 1; the same Figure includes the predecessor set and successor set of every copy of units "a" and "b".

STEP TWO is designed to reduce the number of copies that the aplication of step one has produced. Since step one processes every string sequentially without considering the structure of the entire

| | | | |
|---|---|---|---|
| $a^o$ | $b^o$ | $c^o$ | |
| $a^o$ | $b^o$ | $d^o$ | |
| $c^1$ | $a^1$ | $b^1$ | $c^o$ |
| $e^o$ | $a^1$ | $b^1$ | $c^o$ |
| $f^o$ | $a^2$ | $b^2$ | $d^o$ |

| P | copy | S |
|---|---|---|
| $u_o$ | $a^o$ | $b^o$ |
| $c^1,e^o$ | $a^1$ | $b^1$ |
| $f^o$ | $a^2$ | $b^2$ |
| $a^o$ | $b^o$ | $c^o,d^o$ |
| $a^1$ | $b^1$ | $c^o$ |
| | $a^2$ | $b^2$ | $d^o$ |

Grammar A

| | | | |
|---|---|---|---|
| $a^o$ | $b^o$ | $c^o$ | |
| $f^o$ | $a^1$ | $b^1$ | $d^o$ |
| $a^1$ | $b^1$ | $d^o$ | |
| $c^1$ | $a^o$ | $b^o$ | $c^o$ |
| $e^o$ | $a^o$ | $b^o$ | $c^o$ |

| P | copy | S |
|---|---|---|
| $u_o,c^1,e^o$ | $a^o$ | $b^o$ |
| $u_o,f^o$ | $a^1$ | $b^1$ |
| $a^o$ | $b^o$ | $c^o$ |
| $a^1$ | $b^1$ | $d^o$ |

Grammar B

Figure 2. Effect of processing order on copy labeling of unit "b".

language, the resulting copy labeling of phonetic units can be far from providing a grammar with the lowest number of states. In fact, different orders to process the strings can lead to grammars with different number of states; Figure 2 points out an example of this effect with a subset of L. This example suggests the basic idea that allows to reduce the number of grammar states; it is easy to realize that we can redefine the copies of a phonetic unit $u_j$ in such a way that we make correspond a new copy with each one of the predecessors (or successors) of $u_j$ without altering the language: each new copy is defined with just one predecessor (successor) and with a set of successors (predecessors) determined as the union of the successor (predecessor) sets of every old copy that exhibits the predecessor (successor) defining the new copy. As a consequence, if the number of predecessors (successors) of a phonetic unit is lower than the number of its copies, we can achieve a reduction of grammar size by a straight forward way. Obviously, this redefinition of copies of a phonetic unit makes update the predecessors and successors of the others units. Figure 3 illustrates the grammar optimization afforded by this idea; it is shown that it allows to pass from grammar A in Figure 2 to the minimized grammar B.

| P | copy | S |
|---|---|---|
| $u_o$ | $a^o$ | $b^o,b^1$ |
| $c^1,e^o$ | $a^1$ | $b^o$ |
| $f^o$ | $a^2$ | $b^1$ |
| $a^o,a^1$ | $b^o$ | $c^o$ |
| $a^o,a^2$ | $b^1$ | $d^o$ |

Figure 3. Redefinition of copies of "b" in Grammar A; the subsets of successors of copies of "a" have to be updated. Redefinition of copies of "a" provides Grammar B.

Although the underlying principle of Step Two has been introduced associating each new copy with one predecessor (or successor), it can be generalized by considering subsets of predecessors (or successors) when relabeling the copies. For every phonetic unit $u_j$, we can examine the predecessor sets $P_j^1$ (or successor sets $S_j^1$) of its copies $u_j^1$ looking for expressing each one of them in terms of the union of a reduced number of subsets of $P_j$ ($S_j$); for instance, the successor sets of unit "b" in Figure 1

$$P_b^o = \{c^o, d^o, f^o\} \quad P_b^1 = \{c_o\}, \quad P_b^2 = \{d_o, f_o\}$$

can be written as follows

$$P_b^o = P_b'^0 \cup P_b'^1 \quad P_b^1 = P_b'^0 \quad P_b^2 = P_b'^1$$

Let $P_j'^i$, $i=0,\ldots,n_{pj}-1$ (or $S_j'^i$, $i=0,\ldots,n_{sj}-1$) be those subsets generating all of $P_j^1$ (or $S_j^1$); Step Two of grammar inference algorithm can be stated by the following rule:

rule 3: Let n be the lowest of $n_{pj}$ and $n_{sj}$; it is possible to redefine the copies in such a way that their number will not exceed n. In effect, let n be equal to $n_{pj}$ ($n_{sj}$); each new copy i is defined with the predecessor subset $P_j'^i$ (successor subset $S_j'^i$) and with a set of successors (predecessors) determined as the union of the successor (predecessor) set of every old copy that presents the predecessor subset $P_j'^i$ (successor subset $S_j'^i$) defining the new copy.

The application of rule 3 to a phonetic unit makes us to update the predecessor and successor sets of every state in the grammar. Step Two ends when no additional reduction can be accomplished. As an example, let us consider again the grammar in Figure 1 corresponding to the language L; Step Two optimizes it through the processing shown in Figure 4.

| P | copy | S |
|---|---|---|
| $u_o$ | $a^o$ | $b^o, b^1$ |
| $c^1, e^o$ | $a^1$ | $b^o$ |
| $f^1$ | $a^2$ | $b^1$ |
| $a^o, a^1$ | $b^o$ | $c^o$ |
| $a^o, a^2$ | $b^1$ | $d^o, f^o$ |

iteration 1

| P | copy | S |
|---|---|---|
| $u_o, c^1, e^o$ | $a^o$ | $b^o$ |
| $u_o, f^1$ | $a^1$ | $b^1$ |
| $a^o$ | $b^o$ | $c^o$ |
| $a^1$ | $b^1$ | $d^o, f^o$ |

iteration 2

Figure 4. Optimization of grammar in Figure 1 by Step Two of inference algorithm.

## 4. COMBINATIONS OF LANGUAGES

In many applications the language can be expressed by combinating properly more simple languages. Undoubtly, it is an interesting matter to contemplate the possibility of obtaining the grammar of the resulting language from the grammars of the combined smaller languages. Now, we are going to consider some typical situations.

Let $V_1$ and $V_2$ be two languages with grammars $G_1$ and $G_2$; and let us suppose that both grammars have been arranged so that copies of the same phonetic unit have different labels in $G_1$ and $G_2$. The grammar of the language V union of $V_1$ and $V_2$ can be obtained by applying Step Two to the union of $G_1$ and $G_2$; reduction is possible because origen ($u_o$) and end ($u_\infty$) are the same state in both grammars, and phonetic units that have $u_o$ as predecessor or $u_\infty$ as successor, can reduce the number of their copies according to rule 3. The grammar of the language V cartesian product of $V_1$ and $V_2$ is obtained by modifying the predecessor set of every state in $G_2$ successor of $u_o$ and the successor set of every state in $G_1$ predecessor of $u_\infty$; this modification must set these latter states as predecessors of the former ones.

These results can be utilized in a formal language framework /3/. A context-free grammar in Chomsky normal form has its productions either of the form $A \to a$ or $A \to BC$, where capital letters denote nonterminals and lowercase letters indicate terminal symbols (in our case, strings of phonetic units). A little thought allows to realize that:

a) the set of productions $A \to a_1, \ldots, A \to a_n$ defines a sublanguage $L_A = \{a_1, \ldots, a_n\}$;

b) a pair of productions

$$A \to A_1 = a_1 \quad \text{or} \quad B_1\ C_1$$
$$A \to A_2 = a_2 \quad \text{or} \quad B_2\ C_2$$

can be contemplated as the union of two sublanguages $L_{A1}$ and $L_{A2}$; and

c) the production $A \to BC$ is assimilable to the cartesian product of two sublanguages $L_B$ and $L_C$.

Thus, the introduced inference algorithm could also deal with languages of finite cardinal, specified by context-free grammars. In a forthcoming paper, an algorithmic implementation of this suggestion will be described.

## 5. CONCLUSION

The inference of a finite state grammar from a language specified by a finite set of strings of photenic units has been considered, and an inference algorithm arranged in two steps has been introduced. The first step infers the basic structure of the grammar and the second one minimizes the number of states. This algorithm can also cope with union and cartesian product of languages.

## ACKNOWLEDGMENT

The authors would like to thank Miss Inma Hernáez and Miss Clara Centeno who have implemented the inference algorithm by a FORTRAN program.

## REFERENCES

/1/ H. Ney, "The use of a One-stage dynamic programming algorithm for connected word recognition", IEEE Trans. ASSP-32, no. 2, pp. 263-271, (1984).

/2/ L.R. Rabiner, S.E. Levinson, "A speaker-independent, Syntax-oriented, connected word recognition system ...", IEEE Trans. ASSP-33, no. 3, pp. 561-573, (1985).

/3/ R.C. González, M.G. Thomason, "Syntactic Pattern Recognition: An Introduction", Addison-Wesley Pub. Co., 1978.