The procedure to construct a word predictor in a speech understanding system from a task-specific grammar defined in a CFG or a DCG

Yasuhisa Niimi, Shigeru Uzuhara and Yutaka Kobayashi

Department of Computer Science
Kyoto Institute of Technology
Matsugasaki, Sakyo-ku, Kyoto 606, Japan

Abstract

This paper describes a method for converting a task-dependent grammar into a word predictor of a speech understanding system. Since the word prediction is a top-down operation, left recursive rules induces an infinite looping. We have solved this problem by applying an algorithm for bottom-up parsing.

1. Introduction

In this paper we present a method for converting a task-specific grammar into a word predictor, an important component of a speech understanding system. A context free grammar (CFG) or an augmented transition network grammar (ATNG) have been used to describe task-specific constraint. When a CFG is used, Early's algorithm[1], one of the most efficient top-down parsing algorithms, has been used to make word prediction[2]. When an ATNG is used, word prediction is simply made by tentatively traveling along arcs going out from a state in an ATNG[3],[4],[5]. Since the word prediction is a top-down operation, it is difficult to avoid falling into an infinite loop if the task-specific grammar includes a left recursive rule.

F. Pereira and D. Warren have developed a definite clause grammar (DCG)[6]. The rules described in a DCG are directly converted into a set of Prolog clauses, which works as a parser with an aid of the powerful pattern matching mechanism of Prolog. Thus syntactic analysis can be done without writing a special parser working on the rules of the grammar. Since the syntactic analysis based on a DCG parser also works in top-down fashion, it shares the same difficulty as the top-down parsers have. Y. Matsumoto et al. have developed a method for converting a set of rules described in a DCG into a bottom-up parser which has overcome this difficulty without any loss of the advantages of a DCG[7].

We discuss an application of this method to a word predictor, that is, the method for transforming task-specific linguistic constraint defined in a CFG or a DCG into a Prolog program which acts as a left-to-right word predictor.

2. Word prediction in a speech understanding system

Fig.1 shows a typical configuration of a speech understanding system based on a hierarchical model. An acoustic-phonetic processor analyzes of an input uttereance and transforms it into a sequence of phonetically labeled segments. Provided that a part of an utterance has been dealt with, the controller manages its interpretations in the two kinds of trees illustrated in Fig.2; one represents word strings, of which the ends terminate at different portions on the phonetic sequence, and the other represents the sequences of syntactic categories (called category sequences), each of which is associated with one of the word strings. In this situation, the controller chooses the word string with the highest score, sends the associated category sequence to the word predictor and asks it to predict those syntactic categories which can syntactically follow the selected sequence.

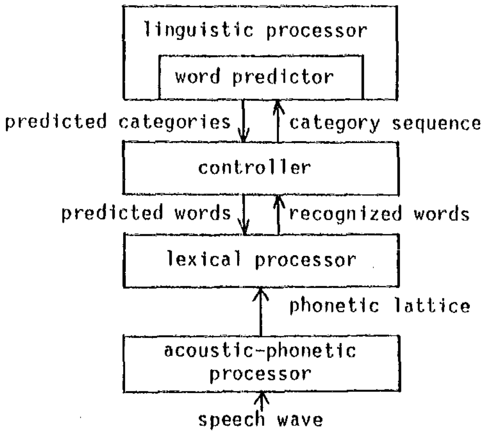

Fig. 1 A typical configuration of a speech understanding system.

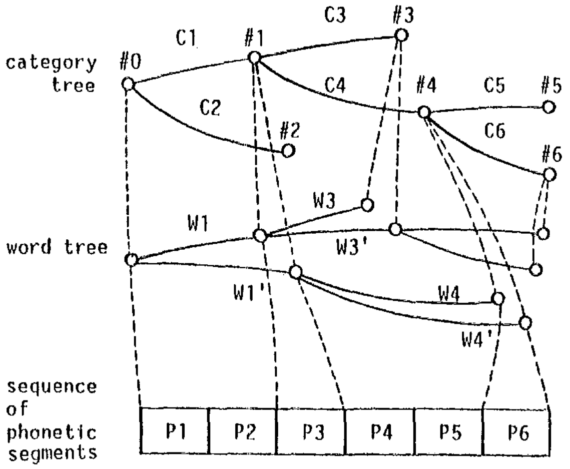

Fig. 2 A search space of a speech understanding system.

The word predictor could parse a given category sequence and predict the categories which can follow it. It is, however, inefficient to analyze the given sequence whenever asked to predict. In fact, each node of the category tree is associated with a parsing history on how rules of the grammar have been applied to analyze the category sequence. The word predictor receives a node and its parsing history from the controller and predicts the syntactic categories following the node.

3. The bottom-up parser and its application to word prediction

We give a brief explanation of the bottom-up parser proposed by Y. Matsumoto et al. Assume simply that the rules of the grammar are described in a CFG. Then, without loss of generality each of the rules can be expressed as either of the followings.

$$c \rightarrow c_1, c_2, \ldots, c_n \qquad (c,\ c_i\ (i=1,\ldots,n)\text{: nonterminals}) \qquad 1)$$

$$c \rightarrow w \qquad (w\text{: a terminal}) \qquad 2)$$

(1) These rules are transformed into the following Prolog clauses.

$$c_1(G,X_1,X) \text{ :- } \mathrm{link}(c,G), \mathrm{goal}(c_2,X_1,X_2), \ldots, \mathrm{goal}(c_n,X_{n-1},X_n),\ c(G,X_n,X). \qquad 1')$$

$$\mathrm{dict}(c,[w|X],X). \qquad 2')$$

X and X_i $(i=1,\ldots,n)$ are arguments to denote a word string to be analyzed as a list. 'link(C,G)' is a predicate to express that a string of which the left most symbol is a nonterminal C can be reduced to a nonterminal G. G is called a goal argument in this sense. 'link' is defined as follows: if the rule 1) is included in the grammar, then 'link(c_1,c)' holds, and if 'link(a,b)' and 'link(b,c)' hold, then 'link(a,c)' holds (transitive law), and 'link(c,c)' holds for every nonterminal c (reflective law). A predicate 'dict(C,X,Y)', searching the dictionary for the first word of a word string X, unifies C with its syntactic category and Y with the remaining string.

(2) A predicate goal(G,X,Z) is defined as follows.

```
goal(G,X,Z) :- dict(C,X,Y),link(C,G),
               exec(C,G,Y,Z).                 3)
```

where 'exec' is a predicate to execute a predicate 'c(G,Y,Z)'.

(3) Furthermore, for any nonterminal C, the following assertion called a terminal condition holds:

```
c(c,X,X).                                     4)
```

The parser for the given grammar consists of all these Prolog clauses.

In order to use the bottom-up parser as a left-to-right word predictor. we change the predicate 'goal' as follows:

```
goal(G,[],[]) :- link(C,G),terminal(C),
                 output(C),fail.           3'-1)
goal(G,X,Z)   :- dict(C,X,Y),link(C,G),
                 exec(C,G,Y,Z).            3'-2)
```

where 'terminal(C)' is a predicate to be true when a nonterminal C appears in the left-hand side of a production of 2).

The modified parser, receiving a word string from the controller, executes the second of 'goal' clauses in which the second argument X is unified with the given word string. Syntactic analysis of X is continued until X becomes empty. Then, the first of 'goal' clauses is invoked and predicts all the syntactic categories which make both 'link(C,G)' and 'terminal(C)' hold.

4. Word prediction under a left-to-right control

In this section we discuss the method for convertion of a set of productions defined in a CFG into a set of Prolog clauses which acts as a left-to-right word predictor. In order that this predictor can work without re-analyzing a given category sequence, we must have a table (named a history table) which contains an association of a category sequence with its parsing history, that is, a history on how productions are used to parse the sequence.

Considering a transition network depicted in Fig.3 for a production '$c \rightarrow c_1 c_2 .. c_n$', we express a parsing history with a list of pairs of a state name in a transition network and a goal argument appearing in bottom-up parsing. For the grammar shown in Fig.4, a category sequence 'N N' is parsed as shown in Fig.5(a) and the corresponding state transition is shown in Fig.5(b). A parsing history for this sequence can be expressed as a list [nps2,s]. The state name 'nps2' indicates that the last 'N' of the

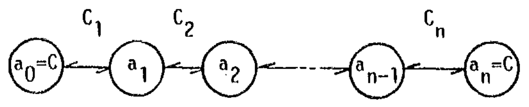

Fig. 3 A transition network for a rule $C \rightarrow C_1\ C_2\ \ldots\ C_n$.

```
S  -> NP VP        NP -> N
NP -> NP N         VP -> V NP
NP -> ART NP
```

Fig. 4 An example of context free grammar.

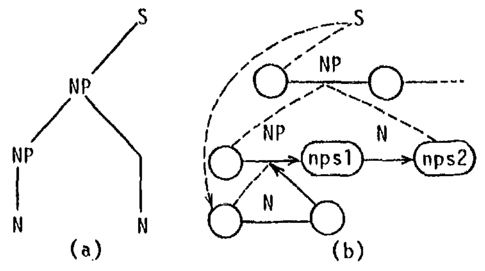

Fig. 5 The parse tree of 'N N' and the corresponding state transition.

sequence 'N N' has been parsed as 'N' in the production 'NP->NP N', and the goal argument 's' indicates that the sequence is the left most part of the string derived by the start symbol 's'.

Now we shall describe the procedure to transform a set of productions described in a CFG into a word predictor.

(1) For a production '$c \to c_1 c_2 .. c_n$', the following set of Prolog clauses is generated:

$c_1([G|H])$:- link(c,G),$a_1([G|H])$.

$a_1(E)$:- pred(c_2,[a_2|E]).

$a_2(E)$:- pred(c_3,[a_3|E]).

...

$a_{n-1}(E)$:- pred(c_n,[a_n|E]).

$a_n(E)$:- c(E). 4-1)

where H and E are the arguments to store parsing histories, the first element of H is a state name and that of E is a goal argument.

(2) For a nonterminal c, the following terminal condition holds:

```
c([c,a|E]) :- exec(a,E).                        4-2)
```

(3) Corresponding to 'goal' in the bottom-up parser, a predicate 'pred' is defined as follows:

```
pred(G,H) :- link(C,G),terminal(C),
             newface(No),hand_to(No,C),
             makenode(No,C,[G|H]),fail.         4-3)
```

A predicate 'newface(No)' generates a new node number in 'No', 'hand_to(No,C)' sends a pair of a node number 'No' and a predicted syntactic category C to the controller, and 'makenode()' stores a node number and its corresponding parsing history expressed as 'C([G|H])' in the history table.

(4) The controller in a speech understanding system communicates the word predictor through a predicate 'wantword' which sends to the word predictor a node number associated with a category sequence which the controller has selected, while the word predictor returns through 'hand_to' a set of the syntactic categories which can follow the selected category sequence. The definition of 'wantword' is as follows:

```
wantword(0) :- !,pred(s,[]).                    4-4)
wantword(No) :- pick_up(No,Z),!,call(Z).        4-5)
```

The symbol s in 4-4) signifies the start symbol, and the clause 4-4) is used to make a prediction at the left most part of an utterance. The predicate 'pick_up(No,Z)' looks up the history table for a node number 'No', and picks up its associated history expressed as 'C([G|H])', the execution of which invokes the clause of 4-1) or 4-2).

5. Conclusions

In this paper we have proposed the procedure to convert a grammar defined in a CFG or a DCG into a Prolog program which functions as a word predictor. The procedure is given for the left-to-right control, but it is not difficult to expand it for the island-driven control.

To simplify the description, we have given the conversion procedure for a grammar defined in a CFG, but it is easy to expand it for a grammar defined in a DCG. As long as one concernes on a speech understanding system in which syntax and semantics are well defined, one could take an advantage of a DCG in which a nonterminal can have some arguments as parameters, and could use semantic restrictions effectively to interpret an utterance. In developing a speech understanding system of which the task is to access a database, we use semantic markers to describe semantic restrictions between an adjective and a noun, a noun phrase and a postposition (in Japanese), and case slots of a verb and its fillers. In this case a rule can be expressed as follows:

$$C(S_0) \to \{P_0(S_0,S_1)\}C_1(S_1)\{P_1(S_1,S_2)\}C_2(S_2)\ldots$$
$$\{P_{n-1}(S_{n-1},S_n)\}C_n(S_n),$$

where S_i (i=0,1,..,n) is a list of semantic markers, P_i (i=1,2,...,n) is a predicate to denote a constraint among semantic markers. Considering a transition network for this DCG rule, we associate P_i with its i-th state and let P_i function as a converter of semantic markers. Since P_i would be defined in the form of a table, this converter could work bidirectionally. In addition, stacking a pair of a syntactic goal variable and a list of semantic markers in the parsing history, we can develop a procedure to transform a grammar described in a DCG into a word predictor.

Acknowledgement

This research was supported by the grant-in-aid for the special project research 'Intelligent Processing and Integration of Knowledge Informations in Multi-Media' by the Ministry of Education, Science and Culture of Japan.

References

[1] J. Early: An efficient context-free parsing algorithm, Comm. ACM, 13-2 (1970).
[2] T. Sakai and S. Nakagawa: A speech understanding system of simple Japanese sentences in a task domain, Trans. of IECEJ, E60-1 (1977).
[3] W.A. Woods et al.: Speech understanding systems -- Final technical progress report 30 October 1974 to 29 October 1976, BBN Tech. Rep. 3438, vol. 4 (1976).
[4] D.R. Reddy et al.: Speech understanding system -- Summary of results of the five year research effort at Carnegie-Mellon Univ., Carnegie-Mellon Univ. Tech. Rep. (1977).
[5] Y. Niimi and Y. Kobayashi: A voice-input programming system using BASIC-like language, Proc. IEEE Int. Conf. ASSP (1978).
[6] F.C.N. Pereira and D.H.D. Warren: Definite clause grammar for language analysis -- A survey of the formalism and comparison with augmented transition networks, Artificial Intelligence, 13 (1980).
[7] Y. Matsumoto et al.: BUP -- A bottom-up parser embedded in Prolog, New Generation Computing, 1-2 (1983).