Grammar Writing System (GRADE) of Mu-Machine Translation Project and its Characteristics

Jun-ichi NAKAMURA, Jun-ichi TSUJII, Makoto NAGAO

Department of Electrical Engineering
Kyoto University
Sakyo, Kyoto, Japan

ABSTRACT

A powerful grammar writing system has been developed. This grammar writing system is called GRADE (GRAmmar DEscriber). GRADE allows a grammar writer to write grammars including analysis, transfer, and generation using the same expression. GRADE has powerful grammar writing facility. GRADE allows a grammar writer to control the process of a machine translation. GRADE also has a function to use grammatical rules written in a word dictionary. GRADE has been used for more than a year as the software of the machine translation project from Japanese into English, which is supported by the Japanese Government and called Mu-project.

1. Objectives

When we develop a machine translation system, the intention of a grammar writer should be accurately stated in the form of grammatical rules. Otherwise, a good grammar system cannot be achieved. A programming language to write a grammar, which is composed of a grammar writing language, and a software system to execute it, is necessary for the development of a machine translation system (Boitet 82).

If a grammar writing language for a machine translation system is to have a powerful writing facility, it must fulfill the following needs.

A grammar writing language must be able to manipulate linguistic characteristics in Japanese and other languages. The linguistic structure of Japanese is largely different from that of English, for instance. Japanese does not restrict the word order strongly, and allows the omission of some syntactic components. When a machine translation system translates sentences between Japanese and English, a grammar writer must be able to express such characteristics.

A grammar writing language should have a framework to write grammars in analysis, transfer, and generation phase using the same expression. It is undesirable for the grammar writer to learn several different expressions for different stages of a machine translation.

There are many word specific linguistic phenomena in a natural language. A grammar writer must be able to add word specific rules to a machine translation system one after another to deal with word specific linguistic phenomena, and improve his machine translation system over a long period. Therefore, a grammar writing language must be able to handle grammatical rules written in word dictionaries.

There is a natural sequence in a translation process. For example, a parsing of noun phrases which do not contain sentential forms is executed before a parsing of more complex noun phrases. An approximate parsing of compound sentences is executed before a parsing of complex sentences. Also, when an application sequence of grammatical rules are written explicitly, a grammar writing system can execute the rules efficiently, because the system just needs to test the applicability of a restricted number of grammatical rules. So, a grammar writing language must be able to express several phases of a translation process in the expression explicitly.

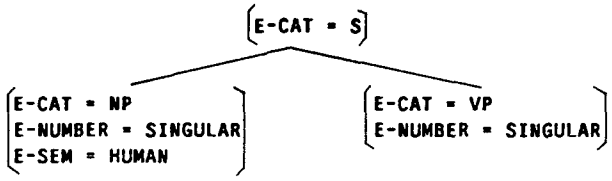
A grammar writing language must be able to treat the syntactic and semantic ambiguities in natural languages. But it must have some mechanisms to avoid a combinatorial explosion.

Keeping these points in mind, we developed a new programming system, which is composed of the grammar writing language and its executing system. We will call it GRADE (Grammar Describer).

2. Expression of the data for a processing

The form of data to express the structure of a sentence during an analysis, a transfer, and a generation process has a strong effect on the framework of a grammar writing language. GRADE uses an annotated tree structure for expressing a sentence. Grammatical rules in GRADE are described in the form of tree-to-tree transformation with annotation to each node.

The annotated tree in GRADE is a tree structure whose nodes have lists of property names and their values. Figure 1 shows an example of the annotated tree.



E-CAT : English Category Symbol
E-NUMBER: English Number (SINGULAR or PLURAL)
E-SEM : English Semantic Marker

Figure 1 An example of the annotated tree in GRADE

The annotated tree can express a lot of information such as syntactic category, number, semantic marker, and other things. The annotated tree can also express a flag in its node, which is similar to a flag in a conventional programming language, to control the process of a translation. For example, in a grammar of a generation, a grammatical rule is applied to all nodes in the annotated tree, whose processings are not finished. In such a case, a grammatical rule checks the DONE flag whether it is processed or not, and sets T to the newly processed ones.

3. Rewriting Rule in GRADE

The basic component of a grammar writing language is a rewriting rule. The rewriting rule in GRADE transforms one annotated tree into another annotated tree. The rewriting rule can be used in the grammars of analysis, transfer and generation phase in a machine translation system, because the tree-to-tree transformation by this rewriting rule is very powerful.

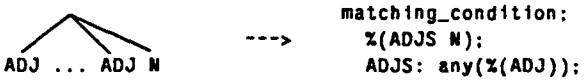
A rewriting rule in GRADE consists of a declaration part and a main part. The declaration part has the following four components. (1) Directory Entry part, which contains a grammar writer's name, a version number of the rewriting rule, and the last date of the revision. This part is not used at the execution time of the rewriting rule. A grammar writer is able to see the information by using the help facility of the GRADE system. (2) Property Definition part, where a grammar writer declares the property names and their values. (3) Variable Init. part, where a grammar writer declares the names of variables. (4) Matching Instruction part, where a grammar writer specifies the mode to apply the rewriting rule to an annotated tree.

The main part specifies the transformation in the rewriting rule, and has the following three parts. (1) Matching Condition part, where the condition of a structure and the property values of an annotated tree is described. (2) Substructure Operation part, which specifies operations for the annotated tree that has matched with the condition written in the matching condition part. (3) Creation part, which specifies the structure and the property values of the transformed annotated tree.

3.1. Matching Condition part

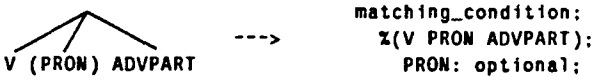
The matching condition part specifies the condition of the structure and the property values of the annotated tree. The matching condition part allows a grammar writer to specify not only a rigid structure of the annotated tree, but also structures which may repeat several times, structures which may be omitted, and structures in which the order of their sub-structures is not restricted.

For example, the structure in which adjectives (ADJ) repeat arbitrary times and a noun (N) follows them in English is expressed as follows.



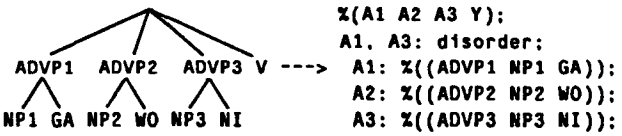

```
matching_condition;
  %(ADJS N);
  ADJS: any(%(ADJ));
```

The structure like a combination of a verb (V) and an adverbial particle (ADVPART) in this sequence with or without a pronoun (PRON) in between in English is written as follows.




```
matching_condition;
  %(V PRON ADVPART);
    PRON: optional;
```

A typical Japanese sentential structure in which three adverbial phrases (ADVP), each composed of a noun phrase (NP) and a case particle (GA, WO, or NI) proceed an verb (V) in no particular order is expressed as follows.




```
matching_condition;
 %(A1 A2 A3 Y);
 A1, A3: disorder;
 A1: %((ADVP1 NP1 GA));
 A2: %((ADVP2 NP2 WO));
 A3: %((ADVP3 NP3 NI));
```

The matching condition part allows a grammar writer to specify conditions about property names and property values for the nodes of the annotated tree. A grammar writer can compare not only a property value of a node with a constant value, but also values between two nodes in a tree.

For example, the number agreement between a subject noun and a verb is written as follows.

```
matching_condition;
  %(NP  VP);
   NP.NUMBER = VP.NUMBER;
```

3.2. Substructure Operation part

The substructure operation part specifies operations for the annotated tree which has matched with the matching condition part. The substructure operation part allows a grammar writer to set a property value to a node, and to assign a tree or a property value to a variable, which is declared in the variable init. part. It also allows him to call a subgrammar, a subgrammar network, a dictionary rule, a built-in function, and a LISP function. The subgrammar, the subgrammar network, the dictionary rule, and the built-in function will be discussed in section 4., 5., and 6. In addition, a grammar writer can write a conditional operation by using the IF-THEN-ELSE form. An operation to set 'A' to the lexical unit of the determiner node (DET.LEX), if the number of the NP node is SINGULAR, is written as follows.

```
substructure_operation;
  if NP.NUMBER = 'SINGULAR';
    then DET.LEX <= 'A';
    else DET.LEX <= 'NIL';
    end_if;
```

3.3. Creation part

The structure and the property values of the transformed annotated tree is written in the creation part. The transformed tree is described by node names such as NP and VP, which are used in the matching condition part or the substructure operation part. A creation part to create the tree whose top node is S and which has a NP sub-tree and a VP sub-tree is written as follows.

```
creation;
  %((S NP VP));
```

3.4. Matching Instruction part

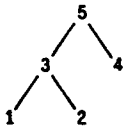
The main part of a rewriting rule in GRADE (the matching condition part, the substructure operation part, and the creation part) can be applied not only to a whole tree, but also to sub-trees. Figure 2 shows an example of the application of a main part.

Transformation of main part in a rewriting rule:

Transformation of a whole annotated tree:

Figure 2 An example of an application of the main part

The matching instruction part specifies the traverse path of the annotated tree. There are four types of the traverse pathes, which are the combinations of <left-to-right or right-to-left> and <bottom-to-top or top-to-bottom>. When a grammar writer specifies left-to-right and bottom-to-top mode, the annotated tree will be traversed as follows.



4. Control of the grammatical rule applications

A grammar writing language must be able to express detailed phases of a translation process in the expression explicitly. GRADE allows a grammar writer to divide a whole grammar into several parts. Each part of the grammar is called a subgrammar. A subgrammar may correspond to a grammatical unit such as the parsing of a simple noun phrase and the paring of a compound sentence. A whole grammar is then described by a network of subgrammars. This network is called a subgrammar network. A subgrammar network allows a grammar writer to control the process of a translation in detail. When a subgrammar network in the analysis phase consists of a subgrammar for a noun-phrase (SG1) and a subgrammar for a verb-phrase (SG2) in this sequence, the executor of GRADE first applies SG1 to an input sentence, then applies SG2 to the result of an application of SG1.

4.1. Subgrammar

A subgrammar consists of a set of rewriting rules. Rewriting rules in a subgrammar have a priority ordering in their application. The n-th

rewriting rule in a subgrammar is tried before the (n+1)-th rule.

A grammar writer can specify four types of application sequence of rewriting rules in a subgrammar. Let us assume the situation that a set of rewriting rules in the subgrammar is composed of RR1, RR2, ..., and RRn, that RR1, ..., and RRi-1 cannot be applied to an input tree, and that RRi can be applied to it. When a grammar writer specifies the first type, which is called ORDER(1), the effect of the subgrammar execution is the application of RRi to the input tree. When a grammar writer specifies the second type, which is called ORDER(2), the executor of GRADE tries to apply RRi+1, ..., RRn to the result of the application of RRi. So, ORDER(2) means that rewriting rules in the subgrammar are sequentially applied to an input tree.

The third and fourth type, which are called ORDER(3) and ORDER(4), are the iteration type of ORDER(1) and ORDER(2) respectively. So, the executor of GRADE tries to apply rewriting rules until no rewriting rule is applicable to the annotated tree.

```
SEARCH-CANDIDATE-OF-NOUNS.sg;
  sg_mode; order(2);
  rr_in_sg;
    CANDIDATE-OF-NOUNS-1;
    UP-NP-TO-PNP;
    CANDIDATE-OF-NOUNS-2;
  end_sg.SEARCH-CANDIDATE-OF-NOUNS;
```

Figure 3 An example of a subgrammar

Figure 3 shows an example of a subgrammar. When this subgrammar is applied to an annotated tree, the executor of GRADE first tries to apply the rewriting rule CANDIDATE-OF-NOUNS-1 to the input tree. If the application of this rule succeeds, the input tree is transformed to the result of the application of the rewriting rule CANDIDATE-OF-NOUNS-1. Otherwise, the input tree is not modified. In either case, the executor of GRADE next tries to apply the rewriting rule UP-NP-TO-PNP to the input tree. The executor continues such a process until the application of the last rewriting rule CANDIDATE-OF-NOUNS-2 is finished.

4.2. Subgrammar Network

A subgrammar network describes the application sequence of subgrammars. The specification of a subgrammar network consists of the following five parts. (1) Directory Entry part, which is as the same as the one in a rewriting rule. (2) Property Definition part, which is the same as the one in a rewriting rule. This part is used as the default declaration in rewriting rules. (3) Variable Init. part, which is the same as the one in a rewriting rule. The variables are used to control the transition of the subgrammar network. The variables are referred to and assigned in the substructure operation part of the rewriting rule. The variables are also referred in a link specification part, which will be described later. (4) Entry part, which specifies a start node of the network. (5) Network part, which specifies a network of subgrammars.

The network part specifies the network structure of subgrammars, and consists of node specifications and link specifications. The node specification has a label and a subgrammar or a subgrammar network name, which is called when the node gets the control of the processing. The link specification specifies the transition among nodes in a subgrammar network. The link specification checks the value of a variable which is set in a rewriting rule, and decides the label of a node which will be processed next.

```
PRE.sgn;
  directory_entry;
    owner(J.NAKAMURA); version(V02L05);
    last_update(83/12/25);
  var_init;
    @PRE-FLAG init(T);
  entry;
    START;
  network;
    START: PRE-STEP-1.sg;
    LOOP : PRE-STEP-2.sg;
    A:     PRE-STEP-3.sg;
    B:     PRE-END-CHECK.sg;
             if @PRE-FLAG; then goto LOOP;
                           else goto LAST;
    LAST:  PRE-STEP-4.sg;
             exit;
  end_sgn.PRE;
```

Figure 4 An example of a subgrammar network.

Figure 4 shows an example of a subgrammar network. When the executor of GRADE applies this subgrammar network to an input tree, the executor checks the var-init part, then puts a new variable @PRE-FLAG on a stack, and sets T to @PRE-FLAG as an initial value. After that, the executor checks the entry part and find the label of the start node START in the network. Then the executor searches the node START and applies the subgrammar PRE-STEP-1 to the input tree. After the application, the executor applies the subgrammar PRE-STEP-2 (node name: LOOP) and PRE-STEP-3 (node name: A) to the annotated tree in this sequence. Next, the executor applies the subgrammar PRE-END-CHECK (node name: B) to the tree.

Rewriting rules in PRE-END-CHECK examine the tree and set T or NIL to the variable @PRE-FLAG. The executor checks the link specification part, which is started by IF, and examines the value of the variable @PRE-FLAG. The node in the network which will be activated next is the node LOOP if @PRE-FLAG is not NIL, otherwise, the node LAST. Thus, while @PRE-FLAG is not NIL, the executor repeats the applications of three subgrammars, PRE-STEP-2, PRE-STEP-3, and PRE-END-CHECK, to the annotated tree. When @PRE-FLAG becomes NIL, the subgrammar PRE-STEP-4 in the node LAST is applied to the tree, and the application of this subgrammar network PRE is terminated.

5. Handling the grammatical rule in the word dictionaries

GRADE allows a grammar writer to write word specific grammatical rules as a subgrammar in an entry of word dictionaries of a machine translation system. A subgrammar written in a dictionary entry is called a dictionary rule. The dictionary rule is specific to a particular word in the dictionary.

The dictionary rule is retrieved with a entry word and a rule identifier as the key, and is applied to the annotated tree which is specified by a grammar writer, when CALL-DIC operation in the substructure operation part is executed. Figure 5 shows an example of a rewriting rule which calls a dictionary rule. In this case, a dictionary rule which is written in an entry of a word as indicated by V.LEX (the value of the lexical unit of verb), and whose name is ANALYSIS, is applied to the sequence of NP1, V, NP2, and PP (noun phrase 1, verb phrase, noun phrase 2, and prepositional phrase). Then the result of the application of the dictionary rule is assigned to the variable @S.

```
CASE-FRAME.rr;
  var_init: @S;
  matching_condition:
    %(NP1 V NP2 PP);
  substructure_operation;
    @S <= call-dic(V.LEX
              ANALYSIS %(NP1 V NP2 PP));
  creation;
    %(@S);
  end_rr.CASE-FRAME;
```

Figure 5 An example of a rewriting rule which calls a dictionary rule

6. Treatment of Ambiguities

A grammar writing language must be able to treat the syntactic and semantic ambiguities in natural languages. GRADE allows a grammar writer to collect all the result of possible tree-to-tree transformations by a subgrammar. However, it must avoid a combinatorial explosion, when it encounters the ambiguities.

For instance, let us assume that a grammar writer writes a subgrammar which contains two rewriting rules to analyze the case frame of a verb, that a rewriting rules is the rule to construct VP (verb phrase) from V and NP (a verb and a noun phrase), and that the other is the rule to construct VP (verb phrase) from V, NP and PP (a verb, a noun phrase, and a prepositional phrase). When he specifies NONDETERMINISTIC_PARALLELED mode to the subgrammar, the executor of GRADE applies both rewriting rules to an input tree, constructs two transformed trees, and merges them into a new tree whose top node has a special property PARA. The top node of this structure is called a para special node, whose sub-trees are the transformed trees by the rewriting rules. Figure 6 shows an example of this mode and a para node.

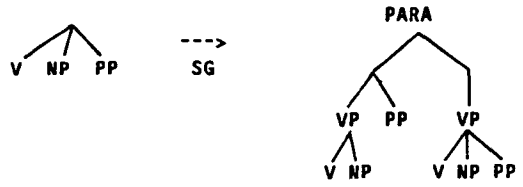


Figure 6 An example of a para special node

A grammar writer can select the most appropriate one from the sub-trees under a para special node. A grammar writer is able to use built-in functions, MAP-SG, MAP-SGN, SORT, CUT, and INJECTION in the substructure operation part to choose the most appropriate one. Figure 7 shows an example to use these built-in functions.

```
substructure_operation;
   @X <= call-dic(V.LEX CASE-FRAME %(N NP PP));
   @X <= call-built(map-sg %(@X) tree
                    EVALUATE-CASE-FRAME);
   @X <= call-built(sort %(@X) tree SCORE);
   @X <= call-built(cut %(@X) tree 1);
   @X <= call-built(injection %(@X) tree 1);
```

Figure 7 An example of built-in functions

In this substructure operation part, the executor of GRADE applies the dictionary rule written in a word which is the value of V.LEX (lexical unit of verb) to the tree, and sets the result to the variable @X. When the nondeterministic-paralleled mode is used in the dictionary rule, the value of @X is the tree whose root node is a para special node. After that, the executor calls built-in function MAP-SG to apply

the subgrammar EVALUATE-CASE-FRAME to each sub-tree of the value of @X, and sets the result to @X again. The subgrammar EVALUATE-CASE-FRAME computes the evaluation score and sets the score to the value of the property SCORE in the root node of the sub-trees. Next, the executor calls built-in function SORT, CUT, and INJECTION to get the sub-tree whose score is the highest one among the sub-trees under the para special node. This tree is then set to @X as the most appropriate result of the dictionary rule.

The para special node is treated as the same as the other nodes in the current implementation of GRADE. A grammar writer can use the para node as he want, and can select a sub-tree under a para node at the later grammatical rule application.

7. System configuration and the environment

The system configuration of GRADE is shown in Figure 8. Grammatical rules written in GRADE are first translated into internal forms, which are expressed by s-expressions in LISP. This translation is performed by GRADE translator. The internal forms of grammatical rules are applied to an input tree, which is an output of the morphological analysis program. This rule application is performed by GRADE executor. The result of rule applications is sent to the morphological generation program.

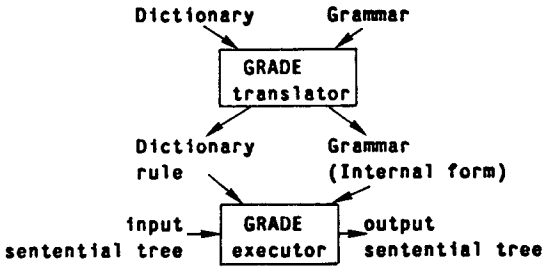


Figure 8 The system configuration of GRADE

GRADE system is written in UTILISP (University of Tokyo Interactive LISP) and implemented on FACOM M382 with the additional function of handling Chainese characters. The system is also usable on Lisp Machine Symbolics 3600. The program size of GRADE system is about 10,000 lines.

8. Conclusion

The grammar writing system GRADE is discussed in this paper. GRADE has the following features. (1) Rewriting rule is an expression in the form of tree-to-tree transformation with annotation to each node. (2) Rewriting rule has a powerful writing facility. (3) Grammar can be divided into several parts and can be linked together as a subgrammar network. (4) Subgrammar can be written in the dictionary entries to express word specific linguistic phenomena. (5) Special node is provided in a tree for embedding ambiguities.

GRADE has been used for more than a year as the software of the national machine translation project from Japanese into English. The effectiveness of GRADE has been demonstrated in this project. The linguistic parts of the project such as the morphological analysis/generation programs, the grammars for the analysis of Japanese, the transfer from Japanese into English and the generation of English, are discussed in other papers (Sakamoto 84) (Tsujii 84) (Nagao 84).

This study: "Research on the machine translation system (Japanese-English) of scientific and technological documents" is being performed through Special Coordination Funds for Promoting Science & Technology of the Science and Technology Agency of the Japanese Government.

ACKNOWLEDGEMENTS

We would like to acknowlege the contribution of M. Kogi, F. Nishino, Y. Sakane, M. Kobayashi, S. Sato, and Y. Senda, who programmed much of the system. We would also like to thank the other member of Mu-project for their useful comments.

REFERENCES

Boitet, Ch., et al, Implementation and Conversational Environment of ARIANE 78.4, Proc. COLING82, 1982.

Nagao, M., et al, Dealing with Incompleteness of Linguistic Knowledege on Language Translation, Proc. COLING84, 1984.

Sakamoto, Y., et al, Lexicon Features for Japanese Syntactic Analysis in Mu-Project-JE, Proc. COLING84, 1984.

Tsujii, J., et al, Analysis Grammar of Japanese in Mu-Project, Proc. COLING84, 1984.