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

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

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 with 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 for a machine translation project from Japanese into English. This was supported by the Japanese Government and was called the Mu-project.

1. Introduction

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 with 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 for a machine translation project from Japanese into English and from English into Japanese. This was supported by the Japanese Government and was called the Mu-project [Nagao 83], [Nagao 84], [Tsujii 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.

2. 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.

OBJECTIVE-1: A grammar writing language must be able to manipulate linguistic characteristics in Japanese and other languages. The linguistic structure of Japanese is very different from that of English, for instance. Japanese does not severely restrict the word order, 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.

OBJECTIVE-2: It is desirable that a grammar writing language have the same framework to write grammars in analysis, transfer, and generation phases. It is not desirable for a grammar writer to learn several different expressions for different stages of a machine translation.

OBJECTIVE-3: 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 to improve his machine translation system over a long period. An improvement of this kind is mainly done in the word dictionaries. Therefore, a grammar writing language must be able to handle grammatical rules written in word dictionaries.

OBJECTIVE-4: There is a natural sequence in a translation process. For example, a parsing of simple noun phrases is executed in advance of a parsing of more complex noun phrases which contain sentential forms. A provisional parsing of compound sentences is executed before a parsing of complex sentences. When an application sequence of grammatical rules is specified 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. In this way, a grammar writing language must be able to express different phases of a translation process in the expression explicitly.

OBJECTIVE-5: A grammar writing language must

*Department of Electrical Engineering, Kyoto University.

be able to resolve syntactic and semantic ambiguities in natural languages. But it must have some mechanisms to avoid a combinatorial explosion.

Many grammar writing languages have been developed for machine translation systems [Nakamura 85]; Augmented Transition Network (ATN) [Woods 70], which is an augmentation of the push-down automaton; LINGOL [Pratt 73] and DCG [Pereira 80], based on Augmented Context Free Grammar (ACFG); SYSTEM-Q [Colmerauer 70] and ROBRA [Boitet 79], based on tree-to-tree transformation. Since push-down automaton and CFG formalism obviously cannot support our objectives 1–4, it is not easy to write a grammar using ATN and ACFG. Tree-to-tree transformation formalisms basically can support our objectives (especially objective 2). But SYSTEM-Q does not have a function to express a sequence in a translation process (objective 4). ROBRA does not allow writing rules in word dictionaries (objective 3), and it does not have a mechanism to explicitly avoid a combinatorial explosion (objective 5). Then, we need a new formalism to support all these objectives for practical machine translation systems.

Keeping these points in mind, we developed a new software system for machine translation, which is composed of the language specification for grammar writing and its executing system. We will call it GRADE (Grammar Describer). GRADE can support all these objectives as explained in the following sections.

3. Expression of the data for processing

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

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

The annotated tree can express a lot of information such as syntactic category, number, semantic marker, and other things, and there is no limitation in the number of properties. The annotated tree can also uses a flag in its node, which control the process of a translation similar to a flag in a conventional programming language. For example, in a grammar of 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.

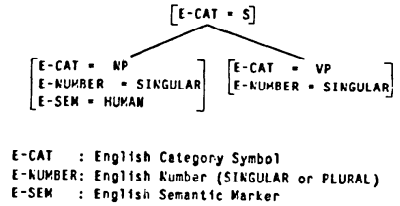


Fig. 1 An example of an annotated tree in GRADE.

4. 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. Because the tree-to-tree transformation by this rewriting rule is very powerful, it can be used in the grammars of analysis, transfer and generation phases of a machine translation, as stated in the objective 2.

A rewriting rule in GRADE consists of a declaration part, in which attributes of a rewriting rule are defined, and a transformation part, in which conditions and a result of the rule application are written. An example of the rewriting rule, CHECK_NOUNS, in GRADE is shown in Fig. 2.

4.1 Declaration part

The declaration part has the following four components: Directory entry part, Property definition part,

```
CHECK_NOUNS.rr;
  directory_entry:
    owner(J.NAKAMURA) ;
    version(V01L02) ;
    last_update(84/4/19);
  prop_def ;
    J_CAT:
      type(u).
      value(NOUN VERB ADJ);
  ver_import:
    @NONSNP;
  var_init:
    @UME;
  matching_instruction;
    level(3,10);
    left_to_right;
    bottom_to_top;
    depth;
    order(Z,noskip);
    tree;
  matching_condition;
    %((X0   X1 BKK1 X2 BKK2 X3 BKK3 X4 #2));
      X0.NONSNP = @NONSNP.VAL;
      X1= 'N' | 'NP';
      X2= 'N' | 'NP';
      X3= 'N' | 'NP';
      X4= 'N' | 'NP';
  substructure_operation;
    if X1.J_PASS='YES';
      then @A<=call-sg(CHECK_PATTERN
            %( X2 BKK2 X3 BKK3 X4 #2) list);
      else @A<=call-sg(FOUR_OR_MORE_NOUNS
            %( X1 BKK1 X2 BKK2 X3 BKK3 X4 #2) list);
    end_if;
    @A.UME <= @UME.VAL;
  creation;
    if X1.J_PASS='YES'
      then     %((X0 X1 BKK1 @A))
      else
               %((X0 @A));
    end_if;
end_rr.CHECK_NOUNS;
```

Declaration part

Transformation part

Fig. 2 An example of a rewriting rule.

Variable declaration part, and Matching instruction part.

A directory entry part contains a name of a grammar writer who wrote the rewriting rule, a version number of the rewriting rule, and the last date of the revision. In Fig. 2, the directory entry part shows that the rule CHECK_NOUNS was written by J. NAKAMURA on 84/4/19 and the version of this rule is V01L02. This part is not used at the execution time of the rewriting rule. A grammar writer is able to see the information written in this part by using the help facility of the GRADE system, for example, when he finds something wrong in the rewriting rule. This facility is necessary for a machine translation system, when many grammar writers cooperate to develop the system.

In a property definition part, a grammar writer declares property names and their values. In Fig. 2, J_CAT (Japanese Category Symbol) is used as a property name and its value is one of NOUN, VERB, or ADJ. A grammar writer can remember what properties he uses. This part is useful to check the consistency of rewriting rules, when the grammar becomes very large.

In a variable declaration part, a grammar writer declares the names of global and local variables. In Fig. 2, this part shows that @NONSNP is a global variable, which should be declared in other rule that call this rule, and @UME is a local variable, which is used in this rule or rules called by this rule. A grammar writer can use variables to control rule applications like registars in Augmented Transition Network.

In a matching instruction part, a grammar writer specifies the mode of application of the rewriting rule to an annotated tree (See section 4.3).

4.2 Transformation part

The transformation part specifies the tree-to-tree 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 are described. (2) Substructure operation part: which specifies operations for the annotated tree that have been matched with the condition written in the matching condition part. (3) Creation part: which specifies the structure and the property values of the tranformed annotated tree.

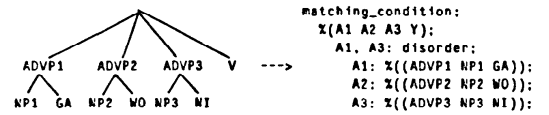
4.2.1 Matching condition part

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 specific structure for the annotated tree, but also structures which may repeat several times, structures which may be omitted, and structures for which the order of sub-structures is free. This function is used to manipulate the linguistic characteristics, especially in Japanese, which are discussed in objective 1.

For example, a structure, in which adjectives (ADJ) repeats an arbitrary number of times and is then followed by a noun (N) in English is written as the expression in Fig. 3(a).

(a) The structure, in which adjectives (ADJ) repeat arbitrary number of times and then a noun (N) follows them in English

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



(c) 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) preceed a verb (V) in no particular order

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

(d) the number agreement between a subject noun and a verb

Fig. 3 Examples of a matching condition part.

A structure in English like a combination of a verb (V) and an adverbial particle (ADVPART) in this sequence with or without a pronoun (PRON) inbetween is expressed as in Fig. 3(b).

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) preceed a verb (V) in no particular order is written as the expression in Fig. 3(c).

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 the expression in Fig. 3(d).

4.2.2 Substructure operation part

The substructure operation part specifies operations on the annotated tree which have matched 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 declared in the variable declaration part. It also allows him to call a subgrammar, a subgrammar network, a dictionary rule, a built-in function, and a LISP function, which will be explained in section 5, 6, and 7. In addition, a grammar writer can write a conditional

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

Fig. 4 An example of a substructure operation part.

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

Fig. 5 An example of a creation part.

```
ABCD.rr;
  matching_instruction;
    level(0,100)
    left_to_right;
    bottom_to_top;
    depth;
    order(2,noskip);
    tree;
  matching_condition;
    %((A #1 B C D #2));
  creation;
    %((E #1 (A B C D) #2);
  end_rr.ABCD;
```

Fig. 6 An example of a rewriting rule.

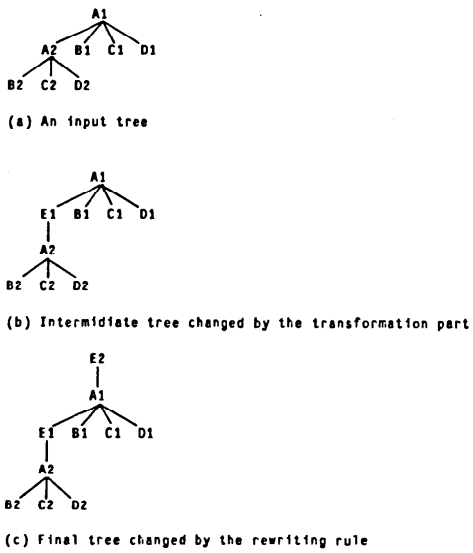


Fig. 7 An example of an application of a transformation part.

operation by using the IF-THEN-ELSE form, to control the translation process. For example, 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 expressed as in Fig. 4.

4.2.3 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 a tree whose top node is S and which has an NP subtree and a VP subtree is written as shown in Fig. 5.

4.3 Matching instruction part

There can be a case that a grammar writer needs to apply a transformation to all subtrees in a sentential structure. For example, when a grammar writer want to decide determiners of all noun phrases in a sentence, he needs not only a rule to choose a determiner of one noun phrase, but also an algorithm to traverse all subtrees in a sentential structure. The rule for one noun phrase is written in the transformation part of a rewriting rule, and the order of traversal in a tree is specified in a matching instruction part.

Consider that the rewriting rule shown in Fig. 6 is applied to the tree in Fig. 7(a). The transformation part of this rule makes a new node E over a tree whose top node is A and whose subtrees are B, C, and D. This transformation is applied to all subtrees in the tree, because the level of traversing is indicated by LEVEL (0,100) in the matching instruction part of the rewriting rule. The input tree is traversed from left to right and from bottom to top in a depth-first order. This order of traversal is specified by the keywords, LEFT_TO_RIGHT, BOTTOM_TO_TOP, and DEPTH. Therefore the tree in Fig. 7(a) is first transformed into a tree in Fig. 7(b), because the subtree whose top node is A2 is an input tree to the transformation part and is transformed into a tree whose top node is E1. Then the tree in Fig. 7(b) is changed for a tree in Fig. 7(c), becuase the subtree whose top node is A1 is transformed in this time.

There are eight types of traverse pathes, which are the combinations of ⟨left-to-right or right-to-left⟩, ⟨bottom-to-top or top-to-bottom⟩, and ⟨depth-first or breadth-first⟩. A grammar writer is able to choose one of the eight types to control the order of his rule application.

5. Control of the grammatical rule applications

A grammar writing language must be able to describe the detailed phases of a translation process in the expression explicitly (objective 4). 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 parsing 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. This control struc-

ture makes it possible to use "a procedural approach" [Tsujii 84] for resolving the ambiguities (objective 5) and selecting the most preferable reading of a sentence.

5.1 Subgrammar

A subgrammar consists of a set of rewriting rules. It is important to decide the order of rule application in a set of rewriting rules.

One way is for the system to put the same priority on all rewriting rules, and make all possible tree structures which satisfy the constrains of each rewriting rule independently, like context free rules. This is a suitable option for linguistic research, but not useful for a machine translation system, because it usually makes so many translations that the system and users, especially post-editors cannot manage them.

The other way is for the system to put a priority ordering in rule applications, make only one possible tree structure at one time, and generates other possible tree structures by using a backtracking method, if necessary. This option allows a grammar writer to use heuristic knowledge concerning preference of rewriting rules. When a grammar writer develops a machine translation system, he must use much heuristic knowledge to output a translation which expresses the first reading of the source sentence.

Therefore, rewriting rules in a subgrammar of GRADE have a priority ordering in their application. In other word, 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.

Fig. 8 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,

```
SEARCH_CANDIDATE_OF_NOUNS.sg:
 matching_instruction:
  order(3):
 rr_in_sg:
  CANDIDATE_OF_NOUNS_1;
  UP_NP_TO_PNP;
  CANDIDATE_OF_NOUNS_2;
end_sg.SEARCH_CANDIDATE_OF_NOUNS;

CANDIDATE_OF_NOUNS_1.rr:
 matching_condition:
  %((NP #));
  NP.J_DEEP_CASE='PARAELEMENT';
 substructure_operation:
  @A<-call-sgn(CUT_NOUNS %(#) list );
 creation:
  %(@A);
end_rr.CANDIDATE_OF_NOUNS_1:
CANDIDATE_OF_NOUNS_2.rr:
 matching_condition:
  %((X Y1 # Y2));
  ((X.J_CAT='S') and (X.J_PARA=-'T'))
  or
  (X.$SYS$='ROOT');
  X.J_PASS_3=-'YES';
 substructure_operation:
  @A<-call-sgn(CUT_NOUNS %(Y1 # Y2) list );
 creation;
  %((X @A));
   X.J_PASS_3<='YES';
end_rr.CANDIDATE_OF_NOUNS_2;
UP_NP_TO_PNP.rr:
 matching_instruction:
  order(2,skip);
 matching_condition:
  %((X # Y));
  X.J_CAT='NP';
  X.J_PARA='T';
  Y.J_CAT='N'|'NP';
 creation:
  %((X # Y));
   X.J_LEX<-Y.J_LEX;
   X.J_SEM<-Y.J_SEM;
   X.J_N<-Y.J_N;
end_rr.UP_NP_TO_PNP;
```

Definition of a subgrammar

Rewriting rules in the subgrammar

Fig. 8 An example of a subgrammar and rewriting rules.

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.

5.2 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, (2) Property definition part, (3) Variable declaration part, (4) Entry part, and (5) Network part. A directory entry part, and a property definition part are the same as the ones in a rewriting rule. They are used as the default declaration in rewriting rules which are called by the subgrammar network. A variable declaration part is also the same as the one in a rewriting rule. Variables are used to control the transition of the subgrammar network. Variables are also referred to in a link specification part, which will be described later. An entry part indicates a start node of the network. A network is a body of the subgrammar network.

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 control of the processing. The link specification defines 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 the node which will be processed next.

Fig. 9 shows an example of a subgrammar network. When the executor of GRADE applies this subgrammar network to an input tree, the executor checks the variable declaration part, then puts a new variable @PRE-FLAG in a stack, and sets T to @PRE-FLAG as an initial value. After that, the executor checks the entry part and finds 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.

The analysis grammar for Japanese abstract in Mu-project uses 85 subgrammar networks, 444 subgrammars, and 1723 rewriting rules.

```
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;
```

Fig. 9 An example of a subgrammar network.

```
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;
```

Fig. 10 An example of a rewriting rule which calls a dictionary rule.

6. Handling the grammatical rule in the word dictionaries

As discussed in objective 3, a grammar writer often needs to write a specific rule for each word. 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.

When CALL-DIC operation in the substructure operation part is executed, the dictionary rule is retrieved with an entry word and a rule identifier as keys, and is applied to the annotated tree which is specified by a grammar writer. Fig. 10 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 verb as indicated by V. LEX (the value of the lexical unit of the 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.

7. Resolving of Ambiguities

A grammar writing language must be able to resolve the syntactic and semantic ambiguities in natural languages (objective 5). GRADE allows a grammar writer to collect all the result of possible tree-to-tree transformations by a subgrammar (not a whole grammar), when he encounters the ambiguities. This function can be used to localize non-deterministic processings for handling the ambiguities, and to avoid a combinatorial explosion.

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 rule 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 subtrees are the transformed trees by the rewriting rules. Fig. 11 shows an example of this mode and a para node.

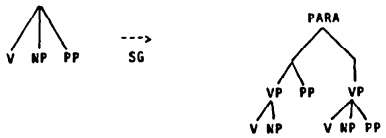


Fig. 11 An example of a para special node.

```
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);
```

Fig. 12 An example of built-in functions.

A grammar writer can select the most appropriate subtree from the subtrees 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. Fig. 12 shows an example to use these built-in functions.

In this substructure operation part, the executor of GRADE applies to the tree the dictionary rule written in a word entry which is the value of V. LEX (lexical unit of verb), 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 subtree 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 subtrees. Next, the executor calls built-in function SORT, CUT and INJECTION to get the subtree whose score is the highest one among the subtrees 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 the same as the other nodes in the current implementation of GRADE. A grammar writer can use the para node as he wants, and can select a subtree under a para node at later grammatical rule applications.

8. System configuration and the environment

The system configuration of GRADE is shown in Fig. 13. Grammatical rules written in GRADE are first translated into internal forms, which are expressed by s-expressions in LISP and are designed to be interpreted by LISP programs easily and effectively. This translation is performed by GRADE translator. Then the internal forms are applied to an input tree by GRADE executor.

GRADE system is written in UTILISP (University of Tokyo Interactive LISP) which is implemented on a

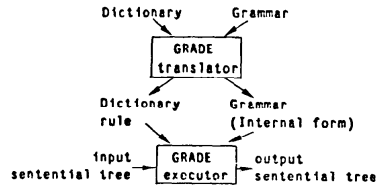


Fig. 13 The system configuration of GRADE.

```
ZERO1.rr;
  directory_entry:
    owner(J.NAKAMURA); version(V01L01); last_update(83/03/06);
  matching_instruction;
    level(0,100); order(2,skip); tree;
  matching_condition;
    %((NP #1 N #"));
      N.E_NUMBER_TYPE = 'R1';
  creation;
    %((NP ART #1 N #2));
    ART.E_LEX <= 'nil';
    ART.E_SUBCAT <= 'ZERO';
  end_rr.ZERO1;
```

(a) A grammatical rule written in GRADE

```
(rr rr_ZERO1
   (("owner       = J.NAKAMURA"
     "version     = V01L01"
     "last_update = 84/03/06")
    rrapplier
    (nil ((ART nil)) nil ((n001) (NP) (N) (#1) (#2)))
    (E_CAT (0 . 100) 1bd (2 . skip) para nil)
    (matcher
      (!cont n000 ((!nop NP NP nil)))
      (!cont NP
             ((!anythings #1)
              (!nop N N
                    (!prop-check
                      ((ueq (pvar N E_NUMBER_TYPE)
                            (pvalue ((R1)))))))
              (!nop #2 nil nil))))
    nil
    (creation
      (construct
        (prop-set
          ((uset ART E_LEX (pvalue (nil)))
           (uset ART E_SUBCAT (pvalue ((ZERO))))
           (uset ART E_CAT (pvalue ((ART))))))
        (constpatt (NP ART #1 N #2))))))
```

(b) An internal form of the rewriting rule

Fig. 14 An example of a rewriting rule and its internal form.

FACOM M382 with an additional function for handling Chinese characters. The system is also useable on Lisp Machine Symbolics 3600. The program size of GRADE system is about 10,000 lines.

8.1 GRADE translator

GRADE translator has three phases, which are lexical analysis, syntactic analysis, and internal form generation.

The lexical analysis routine reads grammatical rules from a file and makes tokens. And it collects the tokens into GRADE sentences, which are separated by ";". Then it executes a simple syntactic analysis of the sentences by using an operator precedence method.

The syntactic analysis and internal form generation routine work after the lexical analysis phase. One function of these routines is to find the relation between a

tree structure and conditions of properties, which is written in the matching condition part implicitly for the readability of grammatical rules, and to decide the timing of property checking during pattern matching. In the rewriting rule shown in Fig. 14(a), the property condition of N node [N.E__NUMBER__TYPE='R1';] is written independently of the tree structure [%((NP #1 N #2));]. The syntactic analysis routine determines that the property condition check will be done after the pattern matching of N node, and the internal form generation routine makes the s-expression shown in Fig. 14(b) [(!nop N N (!prop-check ((ueq (pvar N E__NUMBER__TYPE) (pvalue (R1)))))))]. This internal form means that if there is a tree and the value of the property E__CAT (English Category Symbol) in the root node is N, assign it to the variable N, get the value of the property E__NUMBER__TYPE of the tree assigned to N, and compare the value with R1, whether equal or not. When the internal form is interpreted, it does not need to be tested by the program with the timing of property condition checking.

8.2 GRADE executor

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.

The GRADE executor consists of three parts: subgrammar network and subgrammar application routine, rewriting rule application routine, and tree transformation routine.

The subgrammar network and subgrammar application routine is a toplevel function. It receives an input tree structure, calls the rewriting rule application routine by interpreting the internal forms of subgrammar network and subgrammar, and outputs the transformed tree structure. It uses a backtracking mechanism to support the para special node explained in section 4.

The rewriting rule application routine traverses an input tree in accordance with the specification written in a matching instruction part, and calls the tree transformation routine.

The tree transformation routine consists of a pattern matcher, a program to execute a substructure operation part, and a program to make a transformed tree. This routine changes an annotated tree by interpreting the transformation part in a rewriting rule.

9. 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 capability to handle sophisticated linguistic phenomena. (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 nodes are 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 between Japanese and English. The effectiveness of GRADE has been demonstrated in this project.

Acknowledgements

We would like to acknowlege the contribution of Mr. M. Kogi (TIS), Mr. F. Nishino (Fujitsu), Mr. Y. Sakane (TIS), Mr. M. Kobayashi (NEC), Mr. S. Sato (Kyoto University) and Mr. Y. Senda (Sumitomo Electric), who programmed the major part of the system. We would also like to thank the other members of Mu-project for their useful comments.

References

1. Boitet, C. Automatic Production of CF and CS Analizers using A General Tree Transducer, Universite Scientifique et Medicale de Grenoble (1979).
2. Boitet, C., et al. Implementation and Conversational Environment of ARIANE 78.4, Proc. of COLING82 (1982).
3. Colmerauer, A. LES SYSTEM-Q—Ou Un Formalisme pour Analyser et Synthetiser des Phrases sur Ordinatur, Universite de Montreal (1970).
4. Nagao, M. Outline of the machine translation project of the Japanese Government, WGNL (July 1983) (in Japanese).
5. Nagao, M., et al. Dealing with Incompleteness of Linguistic Knowledege on Language Translation, Proc. of COLING84 (1984).
6. Nakamura, J. Software Environment in Machine Translation, Journal of IPSJ, Vol. 26, No. 10 (1985) (in Japanese).
7. Pereira, F., et al. Definit Clause Grammars for Language Analysis—A Survey of the Formalism and a Comparison with Augmented Transition Networks, Artificial Intelligence, Vol. 13 (1980), 231-78.
8. Pratt, V. R. A Linguistics Oriented Programming Language, Proc. of 3rd IJCAI (1973), 327-82.
9. Sakamoto, Y., et al. Lexicon Features for Japanese Syntactic Analysis in Mu-Project-JE, Proc. of COLING84 (1984).
10. Tsujii, J., et al. Analysis Grammar of Japanese in Mu-Project, proc. of COLING84 (1984).
11. Woods, W. A. Transition Network Grammars for Natural Language Analysis, CACM, Vol. 13 (1970), 591-606.

(Received May 21, 1984; revised July 23, 1985)