ICOT Technical Memorandum: TM-0084

TM-0084

GDLO : A Grammar Description Language Based on DCG

Tarou Morishita (Sharp Corp.)
and
Hideki Hirakawa

November, 1984

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

GDL0: A Grammar Description Language Based on DCG

Tarou MORISHITA
SHARP Corp. Computers and Systems Laboratory

Hideki HIRAKAWA
Institute for New Generation Computer Technology (ICOT)

Abstract

This report proposes an grammar description language for BUP system. It is called "Grammar Description Language 0" which is an extension of DCG formalism. In this language, the data structures for grammar category definitions and grammar rule definitions are described separately. This reinforces the readability and the modifiability of the grammar description. Consequently, it improves the efficiency of grammar development.

1. Introduction

Bottom-up parsing system based on logic programming language (BUP [1]) is becoming practical with the addition of a high-speed processing mechanism [2] and various functional expansions [3],[4]. The BUP system is also being enhanced to become a total system by adding morphological processing and a variety of other program components. BUP system is the basic component of our syntactic analyzer. The BUP system adopts the Definite Clause Grammar(DCG) [5] as its grammar description framework. We consider the DCG formalism is the object-level language of the higher-level grammar description language for the parsing system.

Thus far, much effort has been made to achieve a DCG-based framework for higher-level grammar description. Typical examples are the implementation of XG [6] and that of LFG [7] on the BUP system [8]. More recent ones include an effort to describe a grammar by ESP [11], a logic programming language with object-oriented functions [9], and that to implement the generalized phrase structure grammar (GPSG) [10]. Grammar Description Language 0 (GDL0) to be described as a grammar-describing framework in this report is also one of these approaches.

In this report, DCG has been reviewed from the standpoint of an end-user-oriented grammar description form suitable for grammar developers, and GDL0 which is an extension of DCG formalism is proposed to make some improvements to the DCG description form.

Section 2 describes the problems of the DCG format which were an incentive to develop GDL0. Section 3 shows the analysis of the DCG formalism. In Section 4, the framework of GDL0 is introduced. Finally, Section 4 overviews GDL0-to-DCG translation and provides some translation examples.

2 Problems of DCG Formalism

In implementing a grammar based on context free grammar (CFG) with a logic programming language, DCG formalism is a very powerful tool for describing a grammar. The grammar written in DCG format directly corresponds to the Prolog program. This guarantees the efficiency of the parsing (program execution). Also the augmentations of CFG by the argument attachment and the extra-condition in DCG provides the generality and descriptive power.

On the other hand, there are some requirements of grammar-describing frameworks from the view point of grammar development. They are high descriptive power, efficiency, readability and modifiability. In DCG, a general framework for predicate-call and argument attachment guarantee the high descriptive power, while no constraint on the ways of calling predicates and writing arguments cause its readability and modifiability to degrade. Problems of the DCG format are summarized below.

(1) All grammar data is passed as arguments of a grammar category.

In the development of large-scale grammar, difficulties in debugging are expected to increase as the volume and complexity of the grammar increase. In the present DCG framework, however, since all grammar data necessary for a grammar category is described in the argument portion of the category, all data must be expressed as arguments in writing rules. As a result, the correction or modification of a grammar in DCG formalism is not so easy task.

(2) No constraint on the way of describing arguments.

In DCG, the constraints on the CFG rules and the structure-building procedure are described in the argument part of a grammar category and in the extra-condition part. Generally, arguments with various functions exist according to various types of constraints. Furthermore, due to the argument passing problem mentioned in (1), an argument will often perform different roles depending on rules. In contrast, in the DCG framework, it is difficult to identify and clarify data usage due to the concept of data types.

3 Analysis of DCG Arguments

The problems of DCG formalism are mainly caused from the treatment of arguments. It might be helpful to examine and analyze how DCG arguments are chosen for a particular purpose before explaining GDLO framework. This section analyzes DCG arguments, while considering whether there is any effective restrictive description means for DCG arguments. The Chat80 grammar [6] is used as the main reference.

3.1 Argument rolls

As described in the previous section, arguments provides powerful descriptive power. However this feature degrades the readability of DCG because the role of the argument is not specified explicitly. Basically the argument is used for representing the following roles.

(1) Some grammatical information such as attributes of grammar categories
(2) Some control information
(3) The output of the parsing part

The examples of these roles are described in the next section with the analysis of the grammar.

3.2 Usage of DCG Arguments

Arguments are used for various purposes. The following part shows the usages of arguments.

3.2.1 Arguments for describing constraints

Arguments used in describing constraints are defined as "An argument is used for describing a constraint, if the grammar without the argument accepts the grater number of sentences than the original grammar". These arguments can be further classified as follows:

(1) Arguments which are used for checking the value of an attribute between categories.

(2) Arguments which are used to pass an attribute from a child node to its parent node. These arguments can be considered an auxiliary means to describe those in group <1>. When the features of the head category of an argument are inherited in a complex category at a higher bar level, the argument belongs to this group.

(3) Arguments which works as a rule selector. When the value of an argument determines the rule to be subsequently selected, the argument can function as a rule selector. The following example shows rule selecter case.

```
np_compls(proper,_,_,[],_,Nil) --> {empty(Nil)},
np_compls(common,Agmt,Case,Mods,Set0,Mask)
                                        -->
        {np_all(NPALL)},
        np_mods(Agmt,Case,Rel,Mods,Set0,Set,NPALL,Mask0),
        relative(Agmt,Rel,Set,Mask0,Mask).
```

(From Chat80)

Chat80 has a constraint to prohibit proper nouns from having a post modifier; the description of this constraint is shown above. The first argument of 'np_compls' is passed from 'np_head' in the 'np' reduction rule. its value depends upon the type of 'np_head'. That is, it has 'proper' when it is a common noun. Such an argument can be called a rule selector, since the rule it subsequently selects depends upon the passed value.

3.2.2 Arguments for heuristic control

Structure analysis based on a logic programming language uses a number of grammar descriptions aimed at increasing processing efficiency. Heuristic control is a way to control the application of rules for increased processing efficiency and other purposes. The arguments used in describing heuristic control are defined as "arguments whose existence dose not affect a set of sentences to be accepted, and whose aim is mainly to control the application of rules."

(1) Pruning

To prevent the branch which eventually fails from being searched, a rule is chosen according to the value of a previously determined argument. Pruning differs from rule selection because the result (success or fail) of the rule application is previously known in pruning.

Example

```
verb(....) --> verb_form(Root0,Time+fin,Agmt,Role),
               ...
               rest_verb(Role,Root0,Root,Voice,Aspect),
               ... .
rest_verb(aux,have,Root,Voice,[perf:Aspect]) -->
                 verb_form(Root0,past+part,_,_),
                 have(Root0,Root,Voice,Aspect).
rest_verb(aux,be,Root,Voice,Aspect) -->
                 verb_form(Root0,Tense0,_,_),
                 be(Tense0,Root0,Root,Voice,Aspect).
rest_verb(aux,do,Root,active,[]) -->
                 verb_form(Root,inf,_,_).
rest_verb(main,Root,Root,active,[]) --> [].
```

(From Chat80)

The first argument 'Role' of 'rest_verb' shows whether the first verb is an auxiliary verb or a main verb. The existence of 'Role', however, does not affect a set of verb portions to be accepted, because the value of the second argument 'Root0' controls rule selection and 'Role' is never passed to an upper node verb. Therefore, if 'Role = aux' is defined in 'verb_form', the fourth rule of 'rest_verb' is pruned, while if 'Role = main' is defined, the first to third rules of 'rest_verb' are pruned. While examples of pruning are quite limited, arguments for describing constraints sometimes show a pruning-like operation. Some examples are shown below.

```
question(QCase,subj,S) --> {subj_case(QCase)},s(S).
question(QCase,NPCase,S) --> fronted_verb(QCase,NPCase),s(S).
subj_case(subj).
subj_case(undef).
```

(From Chat80)

'Qcase' is an argument to which the value ('subj'/'compl'/'undef') is set during the analysis of interrogative phrases. Take the sentence "whom do you like?" is an example. In this case, analysis of 'whom' causes 'compl' to be assigned to 'QCase'. Then 'subj_case(QCase)' performs pruning on the first rule of 'question'. Since analysis assuming "do you like" as "s" always fails, a correct rule can be selected for a correct sentence without 'QCase'. This shows that 'QCase' can provide a pruning-like operation, although it is doubtlessly an operation to describe constraints. As this example shows, some DCG arguments may have various roles according to a particular case.

(2) Control to restrict the tree shape

In frequently used techniques, instead of generating a multivocal parse tree a single fixed-form tree is output as the analysis result for improved efficiency, and 'attachment' is determined in a subsequent stage. The Chat80 grammar also has a built-in attachment control which outputs the shape of a parse tree in the Right Most Normal Form (RMNF).

When attachment ambiguity exists, this RMNF control restricts the shape of a parse tree by controlling the application of rules so that modifying phrases will always attach to lower nodes. If, for example, a preposition phrase can qualify both the entire verb phrase and a particular noun phrase in that verb phrase, the RMNF control always attaches it to the lower nounphrase node. At this time, however, it leaves for the subsequent stages the information to show that the preposition phrase can also qualify the entire verb phrase. The following is the RMNF control description for the 'np' post modifier.

```
np_mods(Agmt,Case,Mods0,[Mod:Mods],Set0,Set,_,Mask) -->
            np_mod(Agmt,Case,Mod,Set0,Mask0),
            {trace(Trace),plus(Trace,Mask0,Mask1),
             minus(Set0,Mask1,Set1),plus(Mask0,Set0,Mask2)},
```

```
                np_mods(Agmt,Case,Mods0,Mods,Set1,Set,Mask2,Mask).
np_mods(_,_,Mods,Mods,Set,Set,Mask,Mask) --> [].
```

(From Chat80)

The detail of the RMNF control of Chat80 is described in [6].

3.2.3 Arguments for forming a grammar structure

Arguments are used for building structures which is an output of parsing process. The output structure varies according to the purpose of the system. In CHAT80 the output of the syntactic parsing part contains both grammatical and semantic information.

3.3 Arguments in Rules

In addition to the classification of the roles of the arguments, we can classify the arguments from the view point of the relation between the arguments and the rule in which the arguments appear. Generally speaking the arguments are classified into two types. One is the arguments which are related to the rule and the others are not related. We consider that the mixed use of these arguments degrades the readability of DCG formalism.

3.3.1 Argument significant to the rule

The first type includes arguments which have some meaning to the rule. These includes the arguments which unified with other variables for attribute checking, those which are instantiated, and those which are the target of general predicate call.

The following part shows some examples of argument use mainly taken from the Chat80 grammar.

```
np(Num) --> det(Num), noun(Num).
```

This example shows the attribute matching between categories. The matched "Num" between 'det' and 'noun', however, can be interpreted as checking for attribute matching between them, while the matched "Num" between 'noun' and 'np' can be interpreted as attribute passing to upper nodes.

```
s(...) --> subj(Subj,Agmt,Type),
           verb(Verb,Agmt,Type,Voice), ... .
subj(there,Agmt,_+be) --> [there].
subj(Subj,Agmt,_) --> {s_all(SALL)},
                      np(Subj,Agmt,subj,_,subj,SALL,_).
```

The argument 'Type' indicates the type of a verb ('be', 'have', 'trans', etc.) and its value is determined in the reduction rule of the verb. The first rule in the above example shows a constraint which represents, if 'subj' is a word 'there', the verb following it must be

of type 'be'. In this case, 'Type' is not an attribute inherent in 'subj', but has been passed from the verb, to help describe a constraint spanning more than one rule. It occurs sometimes that an attribute of a category is passed to another category as an argument and its value is restricted by another rule which is apparently unrelated to the original category.

3.3.2 Arguments not significant to the rule

The DCG rules contain arguments which have no significant meaning to the rules. One of this type of arguments is the argument which has been passed from other rules and is used only in passing data to another category within its rules. The following rules show this type of arguments.

```
np_mod(_,Case,PP,Set,Mask) --> pp(PP,Case,Set,Mask).
```

(From Chat80)

'np_mod' is the category showing 'post_modifier' of 'np'. The argument 'Case' passed in this rule represents the rule for 'np' and its value ('compl' or 'subj') was determined when 'np' was analyzed. 'np_mod' receives the argument 'Case' passed from the parent node 'np', and 'pp' passes the argument to 'np' of a child node. This represents the following constraint "If the role of the top node 'np' has been previously determined as 'subj', the role of 'np' which occurs in 'pp' (which is in the post modifier of top 'np') doesn't have 'compl' role but 'subj' role". As far as the above rule is concerned, however, such information cannot be obtained from the argument 'Case' and the argument is used only for data passing in the rule.

The second type of the arguments not significant to the rule is the arguments which are used to describe some multiple reduction rules of the same category and is not used for the remaining rules. In this case, independent or anonymous variables which are not influenced by any action are generated in a rule. It may be considered that they are generated inevitably, because generally the reduction rule for a category is defined by more than one rule.

```
np(np(Agmt,Pronoun,[]),Agmt,NPCase,def,_,Set,Nil) -->
     {is_pp(Set)},
     pers_pron(Pronoun,Agmt,NPCase),
     {empty(Nil)}.
np(np(Agmt,Kernel,Mods),Agmt,NPCase,Def,Role,Set,Mask) -->
     {is_pp(Set)},
     np_head(Kernel,Agmt,Def+Type,PostMods,Mods),
     {np_all(NPALL)},
     np_compls(Type,Agmt,Role,Postmods,NPALL,Mask).
np(part(Det,NP),3+Number,_,indef,Role,Set,Mask) -->
     {is_pp(Set)},
     determiner(Det,Number,indef),
     [of],
```

```
{s_all(SALL)},
np(NP,3+pl,compl(prep),def,Role,SALL,Mask).
```

(From CHAT80)

The argument 'NPCase' of the parent node is used only in the first rule, it has no dependency relation in the other rules. This indicates that 'case' ('compl' or 'subj') of NP is checked only in the grammar rule 'perspron'.

In reading DCG, it is difficult to grasp the relation of passed argument through predicate call whether the argument is inherent in the category or not. This is particularly true for the description of the so-called "long distance dependency", in which data validity is checked at a node positioned away from the node to which the argument is first passed. This is a problem concerning the grammatical framework of CFG in which the entire grammar cannot be viewed from a single grammar rule since individual rules can work on a local basis. This places a limit on the DCG description. If the DCG framework is affirmed, however, arguments can be regarded as a simple, powerful means to achieve long distance dependency.

4. Framework for GDL0

4.1 Basic Explanation

TO avoid a decrease in the grammar development efficiency due to the DCG-related problems and to achieve a user-friendly grammar description, GDL0 provides a framework that allows the portion for defining and managing grammar categories and that for defining grammar rules to be separately described. This means that the data structure for the grammar category is explicitly described and the access to this data structure is explicitly expressed in the grammar rules. By introducing this feature the role of an argument is easily identified and only the arguments which is not significant to the rule are written as the explicit arguments of the rule. The other feature of GDL0 is the use of the macro notation. This makes the users to write compact rules which are easy to read.

GDL0 does not provide an approach for forming special structures, such as LFG's F-structure, or a method to access them. This is because the DCG based system provides an efficient on-line parsing. In contrast to LFG, GDL0 can be a generalized description language for forming the C-structure which represents a surface structure.

(1) Grammar-category-defining portion

GDL0 provides a description field to define the attributes inherent in each grammar category. Also, to introduce the GPSG concept of "head feature convention", it is equipped with a description field to specify

a category as the "head" of another category. This portion for defining grammar categories allows DCG arguments to be abstracted.

(2) Grammar-rule defining portion

This is the main portion of grammar description. The framework for describing rules is basically the same as that for the conventional DCGs, except for the following points which are required to add generality to a description:

1) Only one extra-condition is allowed.

2) Some macro notations are introduced for grammar attribute computation.

This two-part framework has achieved two features. One is that the arguments are separated from grammar categories. The other is that the CFG rule portion is clearly separated from the constraint-describing portion.

4.2 GDL0 Description

The grammar description part of GDL0 consists of two elements: category definitions and grammar rules.

4.2.1 Category definition part

All grammar categories are defined in the following format:

```
category(category_name,
         head_of: category_name0,
         attribute: attribute1,attribute2,...,attributen).
```

'category', a unit clause of Prolog, is a reserved word to show the description of a category definition. Each description element is explained as follows:

'category_name': the name of a grammar category to be defined.
'attribute1 ... attributen': A sequence of grammar information names contained in 'category_name'.
'category_name0': The name of a grammar category which has 'category' name as its head category.

The attributes written in the 'attribute:' field must be the names of data items which can be considered information inherent at least in a given category. There is no restriction on describing attributes, the user can define what is needed as attributes. If the user can find no attribute for that category, this field can be left blank.

The user can specify in the 'head_of' field the name of any category other than a given category. Multiple names cannot be specified. This description has been introduced to specify a category positioned among

categories of similar type at different bar levels as the head of another category to prevent the same attribute name from being repeatedly specified. The above definition shows that 'category_name' is the head category of 'category_name0'. Note that all the attributes defined in the 'attribute:' field in the category definition for 'category_name0' are also defined as attributes of 'category_name'. The attributes of a category consist of those defined in the 'attribute:' field in the category definition for that category and those defined in the 'attribute:' field of the category definition for 'category_name0'. On the other hand, the attributes of a category specified by 'category_name0' also consist of those for that category and those defined in the 'attribute:' field in the category definition for 'category_name'.

While 'head_of' can be defined only with a binomial relation, a 'head_of' hierarchy can be established among more than two categories. In a multi-level 'head_of' hierarchy, however, duplication of an attribute name is prohibited.

Example of 'attribute' use

```
category(np,
        attribute: num,per,word).
category(noun,
        head_of: np).
```

The attributes 'num', 'per', and 'word' need not be specified in the category definition of 'noun'.

Example of 'head_of'

```
category(np,
        attribute: num,per,word).
category(noun,
        head_of: np,
        attribute: type).
category(pronoun,
        head_of: np,
        attribute: anaphor)
```

In this case, each category has the following attributes:

```
pronoun --- num,per,word,anaphor
noun    --- num,per,word,type
np      --- num,per,word,anaphor,type
```

The framework of category definitions looks like defining the 'category class'. In object-oriented languages, class has the functions to handle 'class method' or 'inheritance'. We discussed the use of these functions to improve the descriptive power of GDL0. However, we have not introduced them, because of these reasons:

1) Assume that the class method is an inherent procedure to determine the attributes of a category. Then, the part for describing grammar rules overlaps the part for describing a class, because attribute calculation cannot be separated from rules. For example, describing methods only for complicated constraints requires a large number of rules and thus may result in a complicated description. In contrast, the ESP-based object-oriented parser [9] defines the rule itself as a method in a 'rule class'.

2) There were few examples of procedure-related inheritance, partly because the method definition was not introduced.

3) Hierarchical relations in knowledge representation, such as 'is_a' and 'part_of', do not adequately match a set of grammar categories. For example, although the relation 'noun is_a np' is valid, if desired information is a feature inherent in noun passed to 'np', an 'instance' node must be accessed from an upper level node.

4.2.2 Grammar rule part

A grammar rule is written in the following format:

```
c(Args) --> c1(Args1),...,cn(ArgsN),
                {extra-condition(=Prolog programs with macros)}.
```

The argument field is optional, and ci (i>1) represents a terminal or non-terminal symbol.

(1) Handling arguments

The attributes defined in the category definition for a given category cannot be written in an argument of a grammar category. This restriction has been set to distinguish the data used in data passing from the inherent data.

(2) Handling extra-conditions

In principle, a single extra-condition is written at the end of a rule to perform the entire calculation of attributes. The following macro notations are available for attribute calculation:

1) category x ! attribute y
   'category x' is a category name and 'attribute y' is an attribute name defined in the category definition for 'category x'. It can be handled as a Prolog variable.

2) category x <= category y : [attribute 1,..., attribute n]

   This is equivalent to the following:

   category x ! attribute 1 = category y ! attribute 1

```
                - - -
        category x ! attribute n = category y ! attribute n
```

This notation is used to pass an attribute of a child node to its parent node. The following notation can be used to pass all attributes of a child node to its parent node:

```
        category x <= category y
```

3) category x <=> category y : [attribute 1,..., attribute n]

This macro notation can be expanded in the same way as notation b. It is used to check the duplication of attribute names among child nodes.

Notations 2) and 3) shows attribute passing and duplication checking and can be described with ordinary equality. Prolog's cut ! cannot be used in the extra-condition.

Examples of GDL0 description

```
Category definition: category(np,
                             attribute: num,per,word,structure).
                     category(noun,
                             head_of: np,
                             attribute: type).
                     category(det,
                             attribute: num,word,structure).
Grammar rule:   np --> det,noun,
                    {np!structure=np(det!structure,noun!structure),
                     np<=noun:[num,per,word,type],
                     det!num=noun!num}.
```

Appendix A shows the simple grammar written in GDL0.

5. Translation of GDL0 to DCG

A grammar file written in GDL0 is converted into a grammar file in the DCG format by a GDL0 translator. This section describes some points of translation.

5.1 Basic translation

All attributes of each grammar category are calculated using their category definition,and ,for one attribute one Prolog variable symbol is assigned to an argument of a DCG category. Each category definition defines the argument number of the corresponding category. The attribute names in the category definition specifies the position of the argument attached to the category. Since the category definitions must be processed before the translation of the grammar rule part, the category definitions should locate the top of the grammar file.

A term in the extra-condition which becomes an argument of DCG after translation is assigned a variable which shows that argument. When these terms are connected with equality, however, unification is first performed, and the result is then assigned to category arguments. Attributes which have not been referenced within a rule remain variables after conversion and are used as DCG arguments.

These are the two basic operations for translation. Some translation examples are shown below.

Grammar in GDL0:

```
category(np,attribute:num,per,st).
category(noun,head-of:np,attribute:type).
category(det,attribute:num,st).
np-->det,noun,{ np<=noun:[num,per],
                np!st=np(det!st,noun!st),
                det!num=noun!num,
                member(noun!type,[common,number,proper])}.
```

Translation Result:

```
np(Num,Per,np(Det,Noun),Type1) -->
                        det(Num,Det),
                        noun(Num,Per,Noun,Type2),
                        {member(Type2,[common,number,proper])}.
```

5.2 Distributing term in extra-condition

Since the translation operation described in the previous section treats the argument part of grammar rules, the extra-condition part remains after the rule part in the same order. This causes the problems of efficiency. For example, an extra-condition for pruning or rule selection loses its meaning if it is not located in the correct position. Generally the position of the extra-condition has no logical meaning but it affects the parsing efficiency. GDL0 translator should generate the most efficient codes (DCG program) by arranging the position of extra-conditions. Currently GDL0 translator adopts very simple term distribution algorithm instead of the general one. The following shows the term distribution procedure.

1) Search categories which appear in a given term and have an argument (or an attribute), and find the category positioned at the extreme right in reference to the rule head.

2) Distribute the term by bracing and placing it immediately after the category found in 1). If the searched category is the rule head, place the braced term immediately after the right arrow.

The following example is an GDL0 version of Chat80 description which was used as the example in 2.2 (2).

```
question(QCase,subj)--> s,
                        {subj_case(QCase),
                         question!struc=s!struc}.
```

As already mentioned in Section 2,the processing efficiency will decrease unless 'subj_case(QCase)' is performed before 's' is analyzed. Translation according to the term distribution procedure allows the example above to be converted into the format the same as the DCG rules of the original Chat80.

```
question(QCase,subj,St)--> {subj_case(QCase)},
                           s(St).
```

This procedure, however, cannot handle a case in which a description such as pruning exists in the body of a rule. Distributing an appropriate term in the expansion part of a rule requires a sophisticated algorithm, because this involves the evaluation of execution order in Prolog.

When operations such as pruning and rule selection must be described using general predicates, an extra-condition can be inserted at positions other than the end of a rule, although this violates the description specification described in the previous section. Arguments which can be used within the inserted extra-condition, are those which are defined in the categories before the inserted position. During translation term distribution is performed before the category immediately before the inserted position.

6. Conclusion

GDL0 expands the description format of DCG and separates the part for defining and managing arguments from the part for defining grammar rules to improve readability and modifiability.

While the GDL0-based grammar description is currently being experimented for evaluation, the descriptive power seems to have been enhanced for the following points:

(1) Readability: It is now easier to find what description has been made to what data of what grammar category. Also, it is now possible to make a description by taking into consideration the application purpose of grammar data, when attribute calculation is described.

(2) Modifiability: It is now possible to perform correction, modification and addition with a small amount of rewriting.

The research on grammar description language covers both engineering and linguistic area. We consider GDL0 is one step toward our goal. We consider that the following targets should be achieved to enforce GDL0.

A framework which permits constraints to be written at a desired place without describing an argument used in data passing.

Support functions for a technique for constructing a structure and an access method to a structure.

Addition of useful macro notations in the sense of linguistics and engineering.

We investigate more flexible, convenient framework for an integrated grammar description by checking and developing various grammars.

References

[1] Matsumoto, Y., et. al.; BUP: A Bottom-Up Parser Embedded in Prolog, New Generation Computing, vol.2, OHM-Springer, 1983.

[2] Matsumoto,Y. and Tanaka,H. and Kiyono,M ; The Refinement of the Efficiency of the BUP system, Society for the Study of Natural Language Processing (in Information Processing Society of Japan) 39-7, 1983/9.

[3] Kiyono,M. and Matsumoto,Y. ; The Augmentation of the BUP system and its application to Japanese Processing, 28th National Conference, Information Processing Society of Japan, 1984/3 (in Japanese).

[4] Tanaka,H. and Koyama,H. and Okumura,M. ; The Extension of the BUP system and the Development of Japanese Grammar, Proceedings of The Logic Programming Conference, 12.3, 1984/3 (in Japanese)

[5] Pereira, F. and Warren, D. ;Definite Clause Grammar for Language Analysis - A Survey of the Formalism and a Comparison with Argumented Transition Networks, Artificial Intelligence, 13, 1980, pp 231 - 238.

[6] Pereira,F. ;Logic for Natural Language Analysis, Technical Note 275, SRI International, January, 1983.

[7] Kaplan, R. and Bresnan, J. ; Lexical-Functional Grammar: A Formal System for Grammatical Representation, in "Mental Representation of Grammatical Relations, Bresnan eds., MIT Press, 1982.

[8] Yasukawa,H. and Furukawa,K. ; LFG in Prolog - Toword a formal system for representing grammatical relations -, ICOT TR-019, 1983.

[9] Miyoshi,H.and Furukawa,K. ;Object-Oriented Parser in the Logic Programing Language ESP, ICOT 1984.

[10] Gazdar, G. and Pullum, G. K. ;Generalized Phrase Structure Grammar: A Theoretical Synopsis, Indiana University Linguistics

Club, 1985/8

[11] Chikayama,T. ;ESP Reference Manual, ICOT Technical Report, No44, 1984/2.

Appendix A: Example of Simple grammar and Translation

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%  SIMPLE GRAMMAR  %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% CATEGORY DEFINITION

    category( sentence,
              attribute : st1, st2  ).

    category( np,
              attribute : num, per, spec, st  ).

    category( noun,
              headof : np,
              attribute : word, type, kind  ).

    category( det,
              attribute : spec, num, word  ).

    category( verb,
              headof : vp,
              attribute : word, type, num, per  ).

    category( vp,
              attribute : tense, st  ).

%% DICTIONARY

    verb --> [walks],
             { verb!word = walk,
               verb!type = iv,
               verb!per = 3,
               verb!num = sg,
               verb!tense = pres }.

    verb --> [am],
             { verb!word = be,
               verb!num = sg,
               verb!type = be,
               verb!per = 1,
               verb!tense = pres }.

    verb --> [is],
             { verb!word = be,
               verb!num = sg,
               verb!type = be,
               verb!per = 3,
               verb!tense = pres }.

    verb --> [are],
```

```
        { verb!word = be,
          ( verb!num = pl, verb!per = 3 ;
            verb!num = sg, verb!per = 2 ),
          verb!type = be,
          verb!tense = pres }.

verb --> [was],
        { verb!word = be,
          verb!num = sg,
          verb!type = be,
          ( verb!per = 3 ; verb!per = 1 ),
          verb!tense = past }.

verb --> [were],
        { verb!word = be,
          ( verb!num = sg, verb!per = 2 ;
            verb!num = pl, verb!per = 3 ),
          verb!type = be,
          verb!tense = past }.

verb --> [liked],
        { verb!word = like,
          verb!type = tv,
          verb!tense = past }.

noun --> [john],
        { noun!word = john,
          noun!num = sg,
          noun!per = 3,
          noun!type = proper,
          noun!spec = def(noun!kind),
          noun!kind = personname }.

noun --> [doctor],
         { noun!word = doctor,
           noun!num = sg,
           noun!per = 3,
           noun!type = common,
           noun!kind = none }.

noun --> [doctors],
        { noun!word = doctor,
          noun!num = pl,
          noun!per = 3,
          noun!type = common,
          noun!kind = none  }.

noun --> [girl],
         { noun!word = girl,
           noun!num = sg,
           noun!per = 3,
           noun!type = common,
```

```
                noun!kind = none  }.

    noun --> [woman],
             { noun!word = woman,
               noun!num = sg,
               noun!per = 3,
               noun!type = common,
               noun!kind = none  }.

    noun --> [women],
             { noun!word = woman,
               noun!num = pl,
               noun!per = 3,
               noun!type = common,
               noun!kind = none  }.

    det --> [the],
            { det!spec = def(the) }.

    det --> [this],
            { det!num = sg,
              det!spec = def(this) }.

    det --> [these],
            { det!num = pl,
              det!spec = def(these) }.

    det --> [a],
            { det!num = sg,
              det!spec = indef(a) }.

    det --> [some],
            { det!num = pl,
              det!spec = indef(some) }.


%% GRAMMAR

  sentence --> np,vp(X),
               { sentence!st1 = pred(vp!word,[subj,np!word],[obj,X]),
                 sentence!st2 = sentence(np!st,vp!st),
                 np <=> vp : [num,per]                          }.

  np --> det,noun,
         { np!spec = det!spec,
           np <= noun : [num,per,word],
           det!num = noun!num,
           np!st = np(np!spec,np!word,[num,np!num],[per,np!per]) }.

  np --> noun,
         { noun!type = proper,
           np <= noun : [num,per,word,spec],
```

```
            np!st = np(np!spec,np!word,[num,np!num],[per,np!per]) }.

  np --> noun,
         { noun!num = pl,
           np <= noun : [num,per,word],
           np!spec = indef(pl),
           np!st = np(np!spec,np!word,[num,np!num],[per,np!per]) }.

  vp(X) --> verb,np,
         { verb!type = be,
           verb!tense = pres,
           verb!num = np!num,
           vp <= verb : [num,per,word],
           X = np!word,
           vp!st = vp(verb(verb!word,[tense,verb!tense]),np(np!st)) }.

  vp(X) --> verb,np,
         { verb!type = tv,
           verb!tense = pres,
           vp <= verb :[num,per,word],
           X = np!word,
           vp!st = vp(verb(verb!word,[tense,verb!tense]),np(np!st)) }.

  vp(nil) --> verb,
               { verb!type = iv,
                 verb!tense = pres,
                 vp <= verb : [num,per,word],
                 vp!st = vp(verb(verb!word,[tense,verb!tense])) }.

  vp(X) --> verb,np,
             { verb!type = tv,
               verb!tense = past,
               vp <= verb :[word],
               X = np!word,
               vp!st = vp(verb(verb!word,[tense,verb!tense]),np(np!st))
}.

      %%%%%%%%%%%%%%%%%%%%%%%%%
      %% TRANSLATED DCG FILE %%
      %%%%%%%%%%%%%%%%%%%%%%%%%

verb(walk,iv,sg,3,pres,_23452) -->
          [walks].

verb(be,be,sg,1,pres,_24678) -->
          [am].

verb(be,be,sg,3,pres,_25904) -->
          [is].
```

```
verb(be,be,pl,3,pres,_27130) -->
          [are].

verb(be,be,sg,2,pres,_28356) -->
          [are].

verb(be,be,sg,3,past,_29582) -->
          [was].

verb(be,be,sg,1,past,_30808) -->
          [was].

verb(be,be,sg,2,past,_32034) -->
          [were].

verb(be,be,pl,3,past,_33260) -->
          [were].

verb(like,tv,_34298,_34306,past,_34322) -->
          [liked].

noun(john,proper,person_name,sg,3,def(person_name),_35516) -->
          [john].

noun(doctor,common,none,sg,3,_36890,_36898) -->
          [doctor].

noun(doctor,common,none,pl,3,_38145,_38153) -->
          [doctors].

noun(girl,common,none,sg,3,_39400,_39408) -->
          [girl].

noun(woman,common,none,sg,3,_40648,_40656) -->
          [woman].

noun(woman,common,none,pl,3,_41903,_41911) -->
          [women].

det(def(the),_42830,_42838) -->
          [the].

det(def(this),sg,_43440) -->
          [this].

det(def(these),pl,_44112) -->
          [these].

det(indef(a),sg,_44784) -->
          [a].

det(indef(some),pl,_45456) -->
```

```
          [some].

sentence(pred(_46893,[subj,_46622],[obj,_45793]),
                              sentence(_46614,_46885)) -->
          np(_46590,_46598,_46606,_46614,_46622,_46630,_46638),
          vp(_46877,_46885,_46893,_46901,_46590,_46598,_45793).

np(_48724,_48732,_48740,np(_48740,_48756,[num,_48724],[per,_48732]),
                              _48756,_48764,_48772) -->
          det(_48740,_48724,_49012),
          noun(_48756,_49168,_49176,_48724,_48732,_49200,_49208).

np(_51047,_51055,_51063,np(_51063,_51079,[num,_51047],[per,_51055]),
                              _51079,_51087,_51095) -->
          noun(_51079,proper,_51325,_51047,_51055,_51063,_51357).

np(pl,_53161,indef(pl),np(indef(pl),_53185,[num,pl],[per,_53161]),
                              _53185,_53193,_53201) -->
          noun(_53185,_53425,_53433,pl,_53161,_53457,_53465).

vp(_55502,vp(verb(_55518,[tense,pres]),np(_56019)),
                              _55518,_55526,_55534,_55542,_54548) -->
          verb(_55518,be,_55534,_55542,pres,_55808),
          np(_55534,_56003,_56011,_56019,_54548,_56035,_56043).

vp(_58375,vp(verb(_58391,[tense,pres]),np(_58889)),
                              _58391,_58399,_58407,_58415,_57535) -->
          verb(_58391,tv,_58407,_58415,pres,_58678),
          np(_58865,_58873,_58881,_58889,_57535,_58905,_58913).

vp(_60915,vp(verb(_60931,[tense,pres])),
                              _60931,_60939,_60947,_60955,nil) -->
          verb(_60931,iv,_60947,_60955,pres,_61215).

vp(_63074,vp(verb(_63090,[tense,past]),np(_63588)),
                              _63090,_63098,_63106,_63114,_62249) -->
          verb(_63090,tv,_63353,_63361,past,_63377),
          np(_63564,_63572,_63580,_63588,_62249,_63604,_63612).
```

Appendix B: Execution Procedure of Translator

1) Run the file "eudl.exe" in the execution form to enter Prolog mode.

2) Initiate a top-level predicate 'cat' (:?-cat.). The following prompts appear in turn:

```
> Assign Input File = "Input file name".
                      (Name of a file in GDLO format)
> Assign Output File = "Output file name".
                      (Name of a translated file in DCG format)
```

3) Specify the I/O file to start the execution. During execution, names of actual attributes assigned to individual categories are listed on the screen. When the execution is completed, 'runtime' is displayed and the screen returns to step 2).