LFG System in Prolog

Hideki Yasukawa
The Second Laboratory
Institute for New Generation Computer Technology (ICOT)
Tokyo, 108, Japan

ABSTRACT

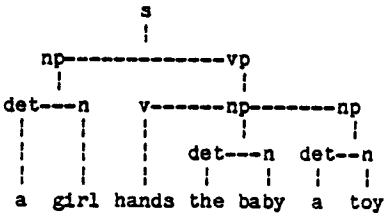
In order to design and maintain a large scale grammar, the formal system for representing syntactic knowledge should be provided. Lexical Functional Grammar (LFG) [Kaplan,Bresnan 82] is a powerful formalism for that purpose. In this paper, the Prolog implementation of LFG system is described. Prolog provides a good tools for the implementation of LFG. LFG can be translated into DCG [Pereira,Warren 80] and functional structures (f-structures) are generated during the parsing process.

I INTRODUCTION

The fundamental purposes of syntactic analysis are to check the grammaticality and to clarify the mapping between semantic structures and syntactic constituents. DCG provides tools for fulfilling these purposes. But, due to the fact that the arbitrary Prolog programs can be embedded into DCG rules, the grammar becomes too complicated to understand, debug and maintain. So, the development of the formal system to represent syntactic knowledges is needed. The main concern is to define the appropriate set of the descriptive primitives used to represent the syntactic knowledges. LFG seems to be promising formalism from current linguistic theories which satisfies these requirements. LFG is adopted for our preliminary version of the formal system and the Prolog implementation of LFG is described in this paper.

II SIMPLE OVERVIEW OF LFG

In this section, the simple overview of LFG is described (See [Kaplan,Bresnan 82] for details). LFG is an extention of context free grammar (CFG) and has two-levels of representation, i.e. c-structures (constituent structures) and f-structures (functional structures). A c-structure is generated by CFG and represents the surface word and phrase configurations in a sentence, and the f-structure is generated by the functional equations associated with the grammar rules and represents the configuration of the surface grammatical functions. Fig. 1 shows the c-structure and f-structure for the sentence "a girl handed the baby a toy" ([Kaplan,Bresnan 82]).



(a) c-structure

```
subj    spec    a
        num     sg
        pred    `girl`
tense   past
pred    `hand<(↑ subj)(↑ obj2)(↑ obj)>`
obj     spec    the
        num     sg
        pred    `baby`
obj2    spec    a
        num     sg
        pred    `toy`
```

(b) f-structure

Fig. 1 The example c-structure and f-structure

As shown in Fig. 1, f-structure is a hierarchical structure constructed by the pairs of attribute and its value. An attribute represents grammatical function or syntactic feature. Lexical entries specify a direct mapping between semantic arguments and configurations of surface grammatical functions, and grammar rules specify a direct mapping between these surface grammatical functions and particular constituent structure configurations. To represent these grammatical relations, several devices and schemata are provided in LFG as shown below.

(a) meta variables
  (i) ↑ & ↓ (immediate dominance)
  (ii) ⇑ & ⇓ (bounded dominance)
(b) functional notations
  a designator (↑ subj) indicates the value of the "subj" attribute of the mother node's f-structure.
(c) Equational schema
  (i) = (functional equation)
  (ii) ∈ (set inclusion)

(d) Constraining schema
  (i) =c (equational constraint)
  (ii) d (existential constraint)
    where d is a designator
  (iii) negation of (i) and (ii)

Fig. 2 shows the example grammar rules and lexical entries in LFG, which generate the c-structure and the f-structure in Fig. 1.

```
1. s ->     np      vp
       (↑ subj)=↓  ↑=↓
2. np -> det   n
         ↑=↓  ↑=↓
3. vp -> v       np          np
         ↑=↓  (↑ obj)=↓  (↑ obj2)=↓
4. det ->    [a]
         (↑ spec)=a  (↑ num)=sg
5. det ->    [the]
         (↑ spec)=the
6. n ->    [girl]
         (↑ num)=sg  (↑ pred)=`girl`
7. n ->    [baby]
         (↑ num)=sg  (↑ pred)=`baby`
8. n ->    [toy]
         (↑ num)=sg  (↑ pred)=`toy`
9. v ->    [handed]
         (↑ tense)=past
         (↑ pred)=`hand<(↑ subj)(↑ obj2)(↑ obj)>`
```

Fig. 2 Example grammar rules and lexical entries of LFG. (from [Kaplan,Bresnan 82])

As shown in Fig. 2, the primitives to represent grammatical relations are encoded in grammar rules and lexical entries. Each syntactic node has its own f-structure and the partial value of the f-structure is defined by the Equational schema. For example, the functional equation "(↑ subj)=↓" associated with the daughter "np" node of grammar rule 1. of Fig. 2 specifies that the value of the "subj" attribute of the f-structure of the mother "s" node is the f-structure of its daughter "np" node. The value constraints on the f-structure are specified by the Constraining schema. Moreover, the grammaticality of the sentence is defined by the three conditions shown below.
(1) Uniqueness: a particular attribute may have at most one value in a given f-structure.
(2) Completeness: a f-structure must contain all the governable grammatical functions governed by its predicate.
(3) Coherence: all the governable grammatical functions that a f-structure contain must be governed by its predicates.

III IMPLEMENTATION OF LFG PRIMITIVES

As indicated in section II, two distinct schemata are employed in the constructions of f-structures. In the current implementation, f-structures are generated during the parsing process by executing the functional equations and set inclusions associated with each syntactic node. After the parsing is done, the f-structures are checked whether their value assignments are consistent with the value constraints on them. The Completeness condition on grammaticality is also checked after the parsing. The LFG primitives are realized by the Prolog programs and embedded into the DCG rules. The Equational schema is executed during the parsing process by the execution of DCG rules. The functional equation can be seen as the extension of the unification of Prolog by introducing equality on f-structures.

A. Representations of Data Types

The primitive data types constructing f-structures are symbols, semantic predicates, subsidiary f-structures, and sets of symbols, semantic predicates, or f-structures. In current implementation, these data types are represented as follows :

1) symbols ==> atom or integer
2) semantic predicates ==> sem(X)
   where X is a predicate
3) f-structure ==> Id:Obt
   where the "Id" is an identifier variable (ID-variable). Each syntactic node has unique ID-variable which is used to identify its f-structure. The "Obt" is a ordered binary tree each leaf contains the pair of an attribute and its value.
4) set ==> {element1, element2, ..., elementN}

A f-structure can be seen as a partially defined data structure, because its value is partially generated by the Equational schema during the parsing process. An ordered binary tree, obt for short, is suitable for representing partially defined data. An obt is a binary tree whose labels are ordered. A binary tree "Obt" is represented by an term of the following form.

Obt = obt(v(Attr,Value),Less,Greater)

The "v(Attr,Value)" is a leaf node of the tree. The "Attr" is an attribute name and used as the label of the leaf node, and the "Value" is its value. The "Less" and "Greater" are also binary trees. The "Obt" is ordered when the "Less" ("Greater") is also ordered and each label of its leaf nodes is less (greater) than the label of "Obt",i.e. "Attr". If none of the leaf of a tree is defined, it is represented by a logical variable. When its label is defined later, the logical variable is instantiated. The insertion of a label and its value into an obt is done by only one unification, without rewriting the tree. This is the merit in using an ordered binary tree.

For example, the f-structure for the noun phrase "a girl", the value of the "subj" in Fig.1 (b), can be graphically represented in Fig. 3.

The "Vi"'s in Fig. 3 are the variables representing the uninstantiated subtrees.

B. Functional Notation

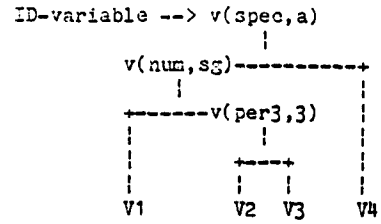


Fig. 3 the graphical representaion of an obt

The functional notations are represented by ID-variables instead of Meta variables ↑ and ↓, i.e. Meta variables must be replaced by the object level variable. For example, the designator (↑ subj) associated with the category S, is described as [subj, IdS], where IdS is the ID-variable for S. The meta variables for bounded dominance are represented by the terms controllee(Cat) and controller(Cat), where the "Cat" is the name of the syntactic category of the controller or controllee.

C. Predicates for LFG Primitives

The predicates for each LFG primitives are as follows : (d,d1,d2 are designators, s is a set, and ~ is a negation symbol)

```
1)   d1 = d2    -> equate(d1,d2,Old,New)
2)    d ∈ s     -> include(d,s,Old,New)
3)  d1 =c d2    -> constrain(d1,d2,OldC,NewC)
4)      d       -> exist(d,OldC,NewC)
5) ~(d1 =c d2)  -> neg_constrain(d1,d2,OldC,NewC)
6)     ~d       -> not_exist(d,OldC,NewC)
```

The "Old" and "New" are global value assignments. They are used to propagate the changes of global value assignments made by the execution of each predicate. The "OldC" and "NewC" are constraint lists and used to gather all the constraints in the analysis.

Besides these predicates, the additional predicates are provided for checking a constraints during the parsing process. They are used to kill the parsing process generating inconsistent result as soon as the inconsistency is found.

The predicate "equate" gets the temporary values of the designators d1 and d2, consulting the global value assignments. Then "equate" performs the unification of their values. The unification is similar to set-theoretic union except that it is only defined for sets of nondistinct attributes. Fig. 4 shows the example trace output of the "equate" in the course of analyzing the sentence "a girl hands the baby a toy".

In order to keep grammar rules highly understandable, it would be better to hide unnecessary data, such as global value assignments or constraint lists. The macro notations similar to the original notation of LFG are provided to users for that purpose. The macro expander translates the macro notations into Prolog programs corresponding to the LFG primitives.

```
The value of the designator Det is
spec     the

The value of the designator N is
num      sg
per      3
pred     sem(girl)

Result of unification is
spec     the
num      sg
per      3
pred     sem(girl)
```

Fig. 4 Tracing results of equate.

This macro expansion results in considerable improvement of the writability and the understandability of the grammar.

The syntax of macro notations are :

```
(a) d1 = d2 -> eq(d1,d2)
(b) d ∈ s -> incl(d,s)
(c) d1 =c d2 -> c(d1,d2)
(d) d -> ex(d)
(e) ~(d1 =c d2) -> not_c(d1,d2)
(f) ~d -> not_ex(d)
```

These macro notations for LFG primitives are placed at the third argument of the each predicate in DCG rules corresponding to syntactic categories as shown in Fig. 5 (a), which corresponds to the grammar rule 1. in Fig. 2.

```
s(s(Np,Vp),Id_S,[]) -->
   np(Np,Id_Np,[eq([subj,Id_S],Id_Np]),
   vp(Vp,Id_Vp,[eq(Id_S,Id_Vp)]).
```

(a) The DCG rule with macro for LFG

```
s(s(Np,Vp),Id_S,Old,New,OldC,NewC) -->
   np(Np,Id_Np,Old,Old1,OldC,OldC1),
   {equate([subj,Id_S],Id_Np,Old1,Old2)},
   vp(Vp,Id_Vp,Old2,Old3,OldC1,NewC),
   {equate(Id_S,Id_Vp,Old3,New)}.
```

(b) The result of macro expansion

Fig. 5 Example DCG rule for LFG analysis

The variables "Id_S", "Id_Np", and "Id_Vp" are the ID-variables for each syntactic category. For example, the grammar rule in Fig. 5 (a) is translated into the one shown in Fig. 5 (b). Macro descriptions are translated into the corresponding predicate in the case of a grammar rule. In the case of a lexical entry, macro descriptions are translated into the corresponding predicate, which is executed further more and the f-structure of the lexical entry is generated.

D. Issues on the Implementation

Though f-structures are constructed during the parsing process, the execution of the Equational schema is independent of the parsing

strategy. This is necessary to keep the grammar rules highly declarative. There are some advantages of using Prolog in implementing LFG. First, the Uniqueness condition on a f-structure is fulfilled by the original unification of Prolog. Second, an ordered binary tree is a good data structure for representing a f-structure. The use of an ordered binary tree reduces the processing time by 30 percents compared with the case using a list for representing a f-structure. And third, the use of ID-variable also effective, because the sharing of a f-structure can be done only by one unification of the corresponding ID-variables.

Though the computational complexity of the Equational schema is very expensive, the LFG provides expressive and natural account for linguistic evidence. In order to overcome the inefficiency, the introduction of parallel or concurrent execution mechanism seems to be a promising approach. The computation model of LFG is similar to the constraint model of computation [Steele 80].

The Prolog implementation of LFG by Reyle and Frey [Reyle,Frey 83] aimed at more direct translation of functional equations into DCG. Although their implementation is more efficient, it does not treat the Constraining schema, set inclusions, the compound functional equation such as (↑ vcomp subj), and the bounded dominance. And their grammar rules seem to be too complex by direct encoding of f-structures into them. In order to provide an formal system having powerful description capabilities for representing syntactic knowledges, the more LFG primitives are realized than their implementation and the grammar rules are more understandable and can be more easily modified in my implementation.

IV. THE RESULT OF AN EXPERIMENT

Fig. 6 shows the result of analyzing the sentence "the girl persuaded the baby to go". LFG system is written in Dec-10 Prolog [Pereira,et.al. 78] and executed on Dec 2060.

As shown in Fig. 6, the functional control [Kaplan,Bresnan 82] is realized in the f-structure of vp. The value of the "subj" attribute of the "vcomp" is functionally controlled by the "obj" of the f-structure of the "s" node. The time used for syntactic analysis includes the time consumed by parsing process and the time consumed by Equational schema.

V. CONCLUSION

The Prolog implementation of LFG is described. It is the first step of the formal system for representing syntactic knowledges. As a result, it becomes quite obvious that Prolog is suitable for implementing LFG.

Further research on the formal system will be carried by analyzing the wider variety of actual utterances to extract the more primitives necessary for the analyses, and to give the necessary schemata for those primitives.

```
Time used in analysis is
     972 ms.(parsing)
     19 ms.(checking constraints)
     41 ms.(for checking completeness)

subj     spec     the
         num      sg
         per      3
         pred     sem(girl)
pred     sem(persuade([subj,A],[obj,A],[vcomp,A]))
obj      spec     the
         num      sg
         per      3
         pred     sem(baby)
tense    past
vcomp    subj     spec   the
                  num    sg
                  per    3
                  pred   sem(baby)
         inf      +
         pred      sem(go([subj,B]))
         to       +
```

Fig. 6 The result of analyzing the sentence, "the girl persuaded the baby to go"

VII. ACKNOWLEDGEMENTS

The author is thankful to Dr. K. Furukawa, the chief of the second research laboratory of ICOT Research Center, and the members of the natural language processing group in ICOT Research Center, both for their discussion. The author is grateful to Dr. K. Fuchi, Director of the ICOT Research Center, for providing the opportunity to conduct this research.

VIII. REFERENCES

[Kaplan,Bresnan 82] "Lexical-Functional Grammar: A Formal System for Grammatical Representation" in "Mental Representation of Grammatical Relations", Bresnan eds., MIT Press, 1982

[Reyle,Frey 83] "A Prolog Implementation of Lexical Functional Grammar", Proc. of IJCAI-83, pp. 693-695, 1983

[Pereira,et.al. 78] "User's Guide to DEC System-10 Prolog", Department of Artificial Intelligence, Univ. of Edinburgh, 1978

[Pereira,Warren 80] "Definite Clause Grammar for Language Analysis -- A Survey of the Formalism and a Comparison with Augmented Transition Networks", Artificial Intelligence, 13, pp. 231-278, 1980

[Steele 80] "The Definition and Implementation of a Computer Programming Language based on Constraints", MIT AI-TR-595, 1980