THE SIXTH JAPANESE-KOREAN
JOINT CONFERENCE
ON FORMAL LINGUISTICS, 1991

Unification in Unification-based Grammar

K.S.Choi, D.J.Son, and G.C.Kim

Department of Computer Science
Korea Advanced Institute of Science and Technology
Seoul, Korea

1. Introduction

After Herbrand's algorithm [HER71] to compute a unifier of two terms in 1930, unification has been applied to the various areas; theorem proving, logic programming, computational complexity, natural language processing, and so on.

In the computational linguistics area, after Kay's adopting of feature structures and their unification to manipulate syntactic structures[KAY79], a lot of unification-based grammar formalisms have been issued such as FUG[KAY82], LFG[BRE82], GPSG[GAZ85], PATR-II[SHI86], HPSG[POL87], etc.. Nowadays, an unification of feature structures is known as one of the most effective and powerful means to process linguistic information, but few practical methodologies have been introduced to implement the unification.

In this paper, We introduce a programming language FUL which has an extended unification intrinsically. Using this programming language as a tool, we suggest an effective unification methodology.

This paper is organized as follows. Section 2 describes the definitions and examples of feature structures and unification. In section 3, some existing unification tools are surveyed and FUL is introduced. In sections 4 and 5, the practical unification processes in two unification-based grammar formalisms (LFG and HPSG) are explained where FUL is used as a unification tool. In section 6, the parsing of the unification-based grammar formalisms is discussed from the viewpoint of the unification. Finally, some conclusions and mentions of the implementations are described in section 7.

2. Feature Structures and Unification

A feature structure can be described in BNF as Figure 1.

```
<feature_structure> ::= [ <paths> ]  |  [ ]
<paths>  ::= <path> , <paths>  |  <path>
<path>  ::= <label> <delimiter> <feature_value>
<feature_value>  ::= <path> | <feature_structure> | <terminal_value>
<label>  ::= <feature_name> | <sharing_tag>
<terminal_value>  ::= <constant> | <list> | <set> | <sharing_tag>
```

Figure 1. Description of a Feature Structure

In Figure 1, '[]' indicates a feature structure which has no information and therefore corresponds to a variable. An example of a feature structure is shown in Figure 2.

```
[ cat : NP,
  agree : <1> : [ num : sing, per : 3 ],
  subj  : <1>
]
```

Figure 2. An Example of a Feature Structure (Matrix Notation)

In figure 2, '<1>' indicates the sharing tag.

Figure 1 and Figure 2 follow the matrix notation to represent the feature structures. On the other hand, we can also follow the directed graph notation, which is known as a more comprehensive and effective representation then matrix notation. The matrix notation corresponding Figure 2 is shown in Figure 3.

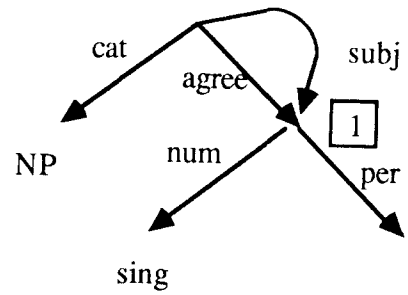

Figure 3. An Example of the Feature Structure Representation(Directed Graph Notation)

In the directed graph notation, edges correspond to the feature names, and external vertices(with child(ren)) to the terminal values. Internal vertices(without child) are pointers pointing to the part of a feature structure. Sharing tag (e.g. '<1>') corresponds to the external vertex. For example, in the Figure 3, the feature names **agree** and **subj** share the part of the feature structure to which the sharing tag '<1>' indicates.

Now we will define the conflict between feature structures which is used in later definition of an unification.

[Definition 1] Conflict between Feature Structures

For two feature structures F1 and F2, if there exists a maximally identical sub-path P, l_1 :: l_2 ::...:: l_n :: where :: is a delimiter and l_i (1 <= i <= n) is a label, which has feature values V1 and V2 in F1 and F2 correspondingly, and V1 and V2 are included to one of the following cases, then it is said that F1 and F2 conflict each other.

1) The kinds of V1 and V2 are different except the case that either of them is '[]'
   (e.g. <path, terminal value>,<path, feature structure(not '[]')>, etc..)
2) V1 and V2 are terminal values and satisfy followings.
   - The kinds of terminal values are different except the case that either of them is a sharing tag.
   (e.g. <constant, list>, <list, set>, etc..)
   - They are constants whose values are different.

3) V1 and V2 are feature structures which conflict each other.

An unification of feature structures is defined as Definition 2.

[Definition 2] Unification of Feature structures
An unification of two feature structures is an operation that takes the least information which subsumes the informations of the two unconflicting feature structures.

An example of an unification of feature structures is shown in Figure 4.

```
[ a : b : <1>,  =  [ a : <2>,      =>  [ a : <2> : b : <1>,
  c : <1>      ]     c : d : <2> ]       c : <1> : d : <2>  ]
```

Figure 4. An Example of an Unification of Feature Structures

3. Unification Tools

Two well known unification tools for computational linguists are Prolog (subsuming DCG) and PATR-II. Prolog is a programming language which provides an unification on term level. As Prolog can't directly unify the complex feature structures including sharing tags (tags to represent a structure sharing in a feature structure), etc., to implement the unification of feature structures, we must program the extended unification module with Prolog that generally requires an enormous effort.

PATR-II is not only an unification-based grammar formalism, but also a tool that implements this grammar formalism. Having a parser intrinsically, PATR-II can parse an input sentence with the grammar and lexicon written in PATR-II formalism. But as PATR-II is inadequate to implement other grammar formalisms [SHI87], it hardly seems to be a general unification tool.

Recently we developed a programming language FUL (acronym of Feature Unification Language) which has an extended unification intrinsically [SON89a].

FUL can unify various objectives such as;

1) terms	
2) lists :	'$ ::' is prefixed.
3) external paths :	'/ ::' is prefixed.
4) sharing tags :	&(N) which is a global sharing tag and #(N) which is a local sharing tag where N is a tag number.
5) feature structures	
6) functions :	pathval/1, append/2, and delete/2.
7) special symbols :	top which corresponds to a variable and bottom which corresponds to a fail.

There are three kinds of unification procedures in FUL such as

U1 = U2,
U1 = U2 => R, and
T1 == T2

where U1 and U2 are the objectives of extended unification, R is a variable, and T1 and T2 are terms. The first procedure mainly checks whether U1 and U2 are unifiable. If they conflict each other, it fails. The second procedure saves the unified result into R. When a conflict occurs, bottom is saved in R but procedure itself succeeds. The third procedure is an term-level unification.

With relations to the unification, FUL has a few of useful functions and procedures such as

```
pathval( F :: P ),
get_pathval( F :: P, V ), and
gather_wfs( F )
```

where F is a feature structure, P is an (internal) path, and V is a variable. pathval/1 is a function which returns the value of P in F. get_pathval/2 is a procedure which gets the value of P in F and saves it into V. In the above extended unifications, if one of the unificands is an external path, the unification procedures generate a fragment of a working feature structure for temporary use. gather_wfs/1 is a function which gathers these fragments of a working feature structure and saves them into F which is a (normal) feature structure.

4. Unification Process in LFG

We can describe simple LFG(Lexical Functional Grammar) grammar rules as Figure 5. (From now on, we follow the notation of FUL.)

```
s  ==>  np ( (|\ subj) = |/ ),
        vp ( |\ = |/ ).
np ==>  n ( |\ = |/ ).
vp ==>  v ( |\ = |/ ),
        np ( (|\ obj) = |/ ).
```

Figure 5. Simple LFG grammar rules

In Figure 5, '|\' corresponds to the up arrow and '|/' to the down arrow in LFG.

We also can describe several LFG lexical items as Figure 6.

```
Mary   :  n,  pred = { Mary },  num = sg.
kissed :  v,  pred = { kiss, subj, obj },  tense = past.
John   :  n,  pred = { John },  num = sg.
```

Figure 6. LFG lexical items

If the input sentence is 'Mary kissed John', the unification process may become as follows. In unification-based grammar formalisms, all linguistic information may be represented in a feature structure. For the case of LFG, we can build the top-level feature structure with the features shown in Figure 7.

1) cat (constituent)

n, vp, s, etc.
2) def_fstruct (definitional F-structure)
definitional equation (=), set inclusion (C)
3) con_fstruct (constraining F-structure)
constraining equation (=c), negation (~)
4) cstruct (C-structure)
phrase structure tree

Figure 7. Features in the top-level feature structure

Now, let's assume that noun phrase NP and verb phrase VP have been built through the unification process as shown in Figure 8.

```
NP = [ cat :: np,
       def_fstruct :: [pred :: $ :: [Mary], num :: sg]
       cstruct :: $ :: [np, [n, [Mary]]]
     ],
VP = [ cat :: vp,
       def_fstruct :: [
          pred :: $ :: [kiss, subj, obj],
          obj :: def_struct :: [pred :: $ :: [John], num :: sg],
          tense :: past ],
       cstruct :: $ :: [vp, [v, [kissed]], [np, [n, [John]]]]
     ].
```

Figure 8. The noun phrase and verb phrase built through unification process

To apply the first grammar rule (s rule) of Figure 5 to NP and VP, we can simply program using FUL as shown in Figure 9. (Here, we show only the unification process between definitional F-structures.)

```
/ :: subj :: def_fstruct = pathval(NP :: def_fstruct),
/ :: def_fstruct = pathval(VP :: def_fstruct),
gather_wfs(S).
```

Figure 9. FUL program that applies the s grammar rule

In Figure 9, the first procedure unifies the definitional F-struct in NP and the definitional F-struct of subj feature in S. The second procedure unifies the definitional F-struct in VP and the definitional F-struct in S.

As a result, a feature structure of S is built as shown in Figure 10.

```
[ cat :: s,
  def_fstruct :: [
     pred :: $ :: [kiss, subj, obj],
     subj :: def_struct :: [pred :: $ :: [Mary], num :: sg],
     obj :: def_struct :: [pred :: $ :: [John], num :: sg],
     tense :: past ],
  cstruct :: $ :: [
     s, [np, [n, [Mary]]], [vp, [v, [kissed]], [np, [n, [John]]]]]
```

```
]
```

Figure 10. A feature structure of S built through the unification of NP and VP

5. Unification Process in HPSG

In HPSG (Head-driven Phrase Structure Grammar), the number of grammar rules are extremely reduced. But HPSG adopts several principles which must be applied in the analyzing process as shown in Figure 11. (Here, we omitted some other principles such as adjunct principle and semantic principle and so on.)

```
HFP = [ syn :: loc :: head :: #(101),
        dtrs :: head_dtr :: syn :: loc :: head :: #(101)
      ],
SCP = [ syn :: loc :: subcat :: $ :: delete( #(103), #(102) ),
        dtrs :: [ head_dtr :: syn :: loc :: subcat :: $ :: #(102),
                  comp_dtrs :: $ :: #(103) ]
      ].
```

Figure 11. The major principles of HPSG

In Figure 11, HFP indicates the Head Feature Principle and SCP the Subcategorization Principle in HPSG.

These principles can be unified for the sake of further application as shown in Figure 12.

```
HFP = SCP => UNIFIED_PRINCIPLE.
```

(a) FUL program to unify the HPSG principles

```
[  syn :: loc :: [ head :: #(101),
                   subcat :: $ :: delete(#(103), #(102)) ],
   dtrs :: [ head_dtr :: syn :: loc :: [ head :: #(101),subcat :: $ :: #(102) ],
             comp_dtrs :: $ :: #(103) ]
]
```

(b) Unified HPSG principles (UNIFIED_PRINCIPLE)

Figure 12. Unification of HPSG principles

We can describe a primitive HPSG grammar rule as Figure 13.

```
PRIMITIVE_GRAMMAR =

[ phon :: $ :: append( #(1), #(2) ),
  syn :: loc :: subcat :: $ :: [ top ],
  dtrs :: [ head_dtr :: [ syn :: loc :: lex :: yes,  phon :: $ :: #(1) ],
            comp_dtrs :: $ :: [ phon :: $ :: #(2) ]]
].
```

Figure 13. A HPSG primitive grammar rule

The completed HPSG grammar rule can be obtained from the unification of the primitive grammar and the unified principles as shown in Figure 14.

```
PRIMITIVE_GRAMMAR = UNIFIED_PRINCIPLE => COMPLETED_GRAMMAR.
```

(a) FUL program to complete the HPSG grammar

```
[ phon:: $:: append( #(1), #(2) ),
  syn:: loc:: [ head:: #(101),  subcat:: $:: delete( #(103), #(102) ),  lex:: no ],
  dtrs:: [
    head_dtr:: [
      phon:: $:: #(1),
      syn:: loc:: [ head:: #(101), subcat:: $:: #(102), lex:: yes ]],
    comp_dtrs:: $:: #(103):: [ phon:: $:: #(2) ]]
]
```

(b) Completed HPSG grammar (COMPLETED_GRAMMAR)

Figure 14. Completion of HPSG grammar

We can describe two HPSG lexical items (lexical signs) as shown in Figure 15.

```
kissed:
  [ phon:: $:: [ kissed ],
    syn:: loc:: [
      head:: [ maj:: v,  vform:: fin ],
      subcat:: $:: [
        [ syn:: loc:: [head:: [maj:: n, case:: acc, nform:: norm], subcat:: $:: []]],
        [ syn:: loc:: [head:: [maj:: n, case:: nom, nform:: norm], subcat:: $:: []]]],
      lex:: yes ]
  ].

John:
  [ phon :: $ :: [ John ],
    syn :: loc :: [
      head :: maj :: n,
      subcat :: $ :: [],
      lex :: yes ]
  ].
```

Figure 15. HPSG lexical items

To apply the grammar in Figure 14(b) to the lexical signs of Figure 15, we can program as shown in Figure 16(a). As a result, we can obtain the phrasal sign as shown in Figure 16(b). (We omitted here the dtrs features and sharing tags.)

```
COMPLETED_GRAM :: dtrs :: head_dtr = lexical sign of kissed,
COMPLETED_GRAM :: dtrs :: comp_dtrs = lexical sign of John.
```

(a) (Abstract) FUL program to get a phrasal sign

```
[ phon:: $:: [ kissed, John ],
  syn:: loc:: [
    head:: [ maj:: v,  vform:: fin ],
    subcat:: $:: [
      [ syn:: loc:: [head:: [maj:: n, case:: nom, nform:: norm], subcat:: $:: []]]],
    lex:: no ]
]
```

(b) An unified phrasal sign

Figure 16. Unification of HPSG grammar and lexical signs

6. Parsing of the Unification-based Grammar Formalisms

A parsing of the unification-based grammar formalisms has following features.
Firstly, besides the constituent information of CFG, complex linguistic informations (morphological, syntactic, semantic, and other informations) are included in the parsing objetives and a parser should manipulate the unification of these informations. Therefore, there are more constraints in the parsing of the unification-based grammar formalisms.
Secondly, because of the high computational complexity of the feature structures and unification of them, the parsing and preprocessing accompanying them have a high time complexity. Some examples of the preprocessing are computations of the reachability relation [WIR87] and LR parsing table [TOM87].
Lastly, in the unification-based grammar formalisms, most linguistic informations exist in the lexical items and they are combined through unification, therfore it's natural to adopt the bottom-up parsing philosophy.

7. Concluding Remarks

Though the core in the unification-based grammar formalisms is the (extended) unification, few of general tools providing this extended unification have been introduced. The programming language FUL can be used as the very tool. FUL itself is implemented using NU-Prolog. Both interpreter and compiler have been developed for FUL. For the convenience of a FUL program debugging, they have tracer intrinsically.

We have implemented prototype English LFG and HPSG parsers with FUL [SON89a]. And now we are developing practical LFG Korean analyzing system with it [SON89b]. All the parsers could be very compact-sized. (For example, each size of the two prototype parsers not exceeds 600 lines in FUL, but they can process almost all functions needed by the grammar formalisms.)

References

[BRE82] J.Bresnan ed., "The Mental Representation of Grammatical Relations," MIT press, 1982.

[GAZ85] G.Gazdar et.al., "Generalized Phrase Structure Grammar," Basil Blackwell, 1985.

[HER71] J.Herbrand, "Recherches sur la theorie de la demonstration," Ph.D. dissertation, in Logical Writings, W.Goldfarb, ed, Harvard University Press, 1971.

[KAY79] M.Kay, "Functional Grammar," in 5th annual meeting of the Berkeley Linguistic Society, 1979.

[KAY82] M.Kay, "Parsing in Functional Unification Grammar," in Natural Language Parsing, D.R.Dowty et.al. ed., Cambridge Univ. Press, pp.251-278, 1982.

[KNI89] K.Knight, "Unification: A Multidisciplinary Survey," ACM Computing Surveys, Vol.21, No.1, pp.93-124, March 1989.

[POL87] C.Pollard and I.A.Sag, "Information-Based Syntax and Semantics," Vol.1, CSLI Lecture Notes No.12, CSLI, 1987.

[SHI86] S.M.Shieber, "An Introduction to Unification-Based Approaches to Grammar," CSLI Lecture Notes No.4, CSLI, 1986.

[SHI87] S.M.Shieber, "Separating Linguistic Analyses from Linguistic Theories," in Linguistic Theory and Computer Applications, P.Whitelock et.al. ed., academic press, pp.1-36, 1987.

[SON89a] D.J.Son, "Design of a Feature Unification Language FUL and Implementations of LFG and HPSG Parsers Using FUL," M.S. Thesis, KAIST, 1989.

[SON89b] D.J.Son, K.S.Choi, G.C.Kim, "An Implementation of a Korean LFG Analyzing System Using FUL," Proc. of Conference of Korean Information Science Society, Vol.16, No.2, 1989.

[TOM87] M.Tomita, "An Efficient Augmented-Context-Free Parsing Algorithm," Computational Linguistics, vol.13, No.1-2, pp.31-46, 1987.

[WIR87] M.Wiren, "A Comparison of Rule-Invocation Strategies in Context-Free Chart Parsing," LiTH-IDA-R-87-13, Univ. of Linkoping, 1987.