Translating a Unification Grammar with Disjunctions into Logical Constraints

Mikio Nakano and **Akira Shimazu***
NTT Basic Research Laboratories
3-1 Morinosato-Wakamiya, Atsugi 243-0198 Japan
E-mail: nakano@atom.brl.ntt.co.jp, shimazu@jaist.ac.jp

Abstract

This paper proposes a method for generating a logical-constraint-based internal representation from a unification grammar formalism with disjunctive information. Unification grammar formalisms based on path equations and lists of pairs of labels and values are better than those based on first-order terms in that the former is easier to describe and to understand. Parsing with term-based internal representations is more efficient than parsing with graph-based representations. Therefore, it is effective to translate unification grammar formalism based on path equations and lists of pairs of labels and values into a term-based internal representation. Previous translation methods cannot deal with disjunctive feature descriptions, which reduce redundancies in the grammar and make parsing efficient. Since the proposed method translates a formalism without expanding disjunctions, parsing with the resulting representation is efficient.

1 Introduction

The objective of our research is to build a natural language understanding system that is based on unification. The reason we have chosen a unification-based approach is that it enables us to describe grammar declaratively, making the development and amendment of grammar easy.

Analysis systems that are based on unification grammars can be classified into two groups from the viewpoint of the ways feature structures are represented: (a) those using labeled, directed graphs (Shieber, 1984) and (b) those using first-order terms (Pereira and Warren, 1980; Matsumoto et al., 1983; Tokunaga et al., 1991).

In addition to internal representation, grammar formalisms can be classified into two groups, (i) those that describe feature structures with path equations and lists of pairs of labels and values (Mukai and Yasukawa, 1985; Aït-Kaci, 1986; Tsuda, 1994), and (ii) those that describe feature structures with first-order terms (Pereira and Warren, 1980; Matsumoto et al., 1983; Tokunaga et al., 1991). Since formalisms (i) are used in the family of the PATR parsing systems (Shieber, 1984), hereafter they will be called PATR-like formalisms.

Most of the previous systems are either ones that generate representation (a) from formalisms (i) or ones that generate representation (b) from formalisms (ii). However, representation (b) is superior, and formalism (i) is far better. Representation (b) is superior for the following two reasons. First, unification of terms is more efficient of that of graphs because the data structure of terms is simpler (Schöter, 1993).[1] Second, it is easy to represent and process *named disjunctions* (Dörre and Eisele, 1990) in the term-based representation. Named disjunctions are effective when two or more disjunctive feature values depend on each other. The treatment of named disjunctions in graph unification requires a complex process, while it is simple in our logical-constraint-based representations. Formalism (i) is better because term-based formalism is problematic in that readers need to memorize the correspondence between arguments and features and it is not easy to add new features or delete features (Gazdar and Mellish, 1989).

Therefore, it is effective to translate formalism (i) into representation (b). Previous translation methods[2] (Covington, 1989; Hirsh, 1988; Schöter, 1993; Erbach, 1995) are problematic in that they cannot deal with disjunctive feature descriptions, which reduce redundancies in grammar. Moreover, incorporating disjunctive information into internal representation makes parsing more efficient (Kasper, 1987; Eisele and Dörre, 1988; Maxwell and Kaplan, 1991; Hasida, 1986).

This paper presents a method for translating grammar formalism with disjunctive information based on path equations and lists of pairs of labels and values into term-

* Presently with Japan Advanced Institute of Science and Technology.

[1] Since unspecified features are represented by variables in term unification, when most of the features are unspecified, it is inefficient to represent feature structures by terms. In current linguistic theories such as HPSG (Pollard and Sag, 1994), however, thanks to the type specifications, the number of features that a feature structure can have is reduced, so it does not cause as much trouble.

[2] Methods that generate representation (b) after generating representation (a) are included.

based representations, without expanding disjunctions. The formalism used here is *feature-based formalism with disjunctively defined macros* (FF-DDM), an extension of the PATR-like formalisms that incorporates a description of disjunctive information. The representation used here is *logical-constraint-based grammar representation* (LCGR), in which disjunctive feature structures are represented by Horn clauses.

2 Unification Grammar Formalisms with Disjunctive Information

The main difference between PATR and FF-DDM is that there can be only one definition for one macro in PATR while multiple definitions are possible in FF-DDM. These definitions are disjuncts. If the conditions in one of the definitions of a macro are satisfied, the condition the macro represents is satisfied. In FF-DDM, the grammar is described using four kinds of elements: type definitions, phrase structure rules, lexical entries, and macro definitions.

Some examples are shown below. The first is an example of type definition.

```
(1) (deftype sign
      pos agr subj)
```

This means that there is a type named *sign* and the feature structures of type *sign* can have POS, AGR, and SUBJ features.

This is an example of a phrase structure rule.

```
(2) (defrule psr1 (s -> np vp)
      (<s pos> = sentence
       <np pos> = noun
       <vp pos> = verb
       <vp subj> = <np>
       <np agr> = <vp agr>
       <s agr> = <vp agr>))
```

Here `psr1` is the name of this rule. Variable `s` denotes the feature structure of the mother node, and `np` and `vp` are variables that denote the feature structures of the daughter nodes. Rule `psr1` denotes the relationship between three feature structures `s`, `np`, and `vp`. The fourth argument is a set of path equations. The path equation `<s pos> = sentence` indicates that the POS feature value in the feature structure represented by the variable `s` is *sentence*. The path equation `<vp subj> = <np>` means the SUBJ feature value of `vp` is identical to the feature structure `np`. A path can be a list of pairs of labels and values, although we do not explain this in detail in this paper.

Next we show an example of a lexical item.

```
(3) (defword walk (sign)
      (<sign pos> = verb
       <sign agr> = <sign subj agr>)
      (not3s <sign agr>))
```

Here `sign` is the variable that represents the lexical feature structure for *walk*. The disjunctively defined macro `(not3s <sign agr>)` in the last line shows that the AGR feature value of `sign` must satisfy one of the definitions of `not3s`.

Examples of macro definitions, or definitions of `not3s`, are shown below.

```
(4) (defddmacro not3s (agr)
      (<agr num> = sing)
      (1st-or-2nd <agr per>))
(5) (defddmacro not3s (agr)
      (<agr num> = plural))
```

If one of these is satisfied, the condition for macro `not3s` is satisfied. Two definitions, (4) and (5) stand in a disjunction relation. [3]

3 Logical-Constraint-Based Grammar Representation

3.1 Logical Constraint Representation of Disjunctive Feature Structures

We will first define logical constraints. A logical constraint (*constraint* for short) is a set of positive literals of first-order logic. Each positive literal that is an element of a constraint is called a *constraint element*.

An example of a constraint is (6). Constraint elements are written in the DEC-10 Prolog notation. The names of variables start with capital letters.

(6) $\{p(X), q(X, f(Y))\}$

A *definition clause* of a predicate is a Horn clause having that predicate as the predicate of its head. For example, (7) is a definition clause of p.[4]

(7) $p(f(X,Y)) \leftarrow \{r(X), s(Y)\}$

The bodies of definition clauses can be considered as constraints, that is, bodies can be considered to constrain the variables in the head. For example, definition clause (7) means that, for a pair of the variables X and Y, $p(f(X,Y))$ is true if the instances satisfy the constraint $\{r(X), s(Y)\}$. We omit the body when it is empty. The set of definition clauses registered in the system is called a *database*.

Feature structures that do not include any disjunctions can be represented by first-order terms. For example, (8) is described by (9).

(8) **sign** $\left[\begin{array}{ll} \text{POS} & v \\ \text{AGR} & \textbf{agr}\left[\begin{array}{ll}\text{NUM} & sing \\ \text{PER} & 3rd\end{array}\right] \\ \text{SUBJ} & \textbf{sign}\left[\text{AGR}\ \textbf{agr}\left[\begin{array}{ll}\text{NUM} & sing \\ \text{PER} & 3rd\end{array}\right]\right]\end{array}\right]$

[3] Since there is no limitation on the number of arguments of a macro, named disjunctions can be described.

[4] Horn clauses are described in a different notation from DEC-10 Prolog so as to indicate explicitly that the bodies can be recognized as constraints.

(9) $sign(v, agr(sing, 3rd), sign(\_, agr(sing, 3rd), \_))$

Feature structure (8) is a *typed feature structure* used in typed unification grammars (Emele and Zajac, 1990). The set of features that a feature structure can have is specified according to types. In this paper, we do not consider type hierarchies. Symbol "_" in (9) is an anonymous variable. The arguments of function symbol *sign* correspond to POS feature, AGR feature, and SUBJ feature values.

Disjunctions are represented by the bodies of definition clauses. A constraint element in a body whose predicate has multiple definition clauses represents a disjunction. For example, in our framework a disjunctive feature description (10)[5] is represented by (11).

(10)
$$\left\{ \begin{array}{l} \textbf{sign} \left[\begin{array}{ll} \text{POS} & v \\ \text{AGR} & *1 \left\{ \begin{array}{l} \textbf{agr} \left[\begin{array}{ll} \text{NUM} & sing \\ \text{PER} & \left\{ \begin{array}{l} 1st \\ 2nd \end{array} \right. \end{array} \right] \\ \textbf{agr} \left[\begin{array}{ll} \text{NUM} & plural \end{array} \right] \end{array} \right. \\ \text{SUBJ} & \textbf{sign} \left[\text{AGR} \quad *1 \right] \end{array} \right] \\ \textbf{sign} \left[\begin{array}{ll} \text{POS} & n \\ \text{AGR} & \textbf{agr} \left[\begin{array}{ll} \text{NUM} & sing \\ \text{PER} & 3rd \end{array} \right] \end{array} \right] \end{array} \right\}$$

(11) $p(sign(v, Agr, sign(\_, Agr, \_)))$
$\leftarrow \{not\_3s(Agr)\}$
$p(sign(n, agr(sing, 3rd), \_)) \leftarrow$
$not\_3s(agr(sing, Per)) \leftarrow \{1st\_or\_2nd(Per)\}$
$not\_3s(agr(plural, \_)) \leftarrow$
$1st\_or\_2nd(1st) \leftarrow$
$1st\_or\_2nd(2nd) \leftarrow$

Literal $p(X)$ means that variable X is a candidate for the *disjunctive feature structure* (DFS) specified by predicate p. The constraint element $1st\_or\_2nd(Per)$ in (11) constrains variable Per to be either $1st$ or $2nd$. In a similar way, $not\_3s(Agr)$ means that Agr is a term having the form $agr(Num, Per)$, and that either Num is $sing$ and Per is subject to $1st\_or\_2nd(Per)$ or that Num is $plural$. As this example shows, constraint elements in bodies represent disjunctions and each definition clause of their predicates represents a disjunct.

3.2 Unification by Logical Constraint Transformation

Unification of DFSs corresponds to logical constraint satisfaction. For example, the unification of DFSs $p(X)$ and $q(Y)$ is equivalent to obtaining all instances of X that satisfy $\{p(X), q(X)\}$.

In order to be able to use the result of one unification in another unification, it would be useful to output results in the form of constraints. Such a method of satisfaction is called *constraint transformation* (Hasida, 1986). Constraint transformation returns a constraint equivalent to the input when it is satisfiable, but it fails otherwise.

[5]Braces represent disjunctions.

The efficiency of another unification using the resulting constraint depends on which form of constraint the transformation process has returned. Obtaining compact constraints corresponds to avoiding unnecessary expansions of disjunctions in graph unification (Kasper, 1987; Eisele and Dörre, 1988). Some constraint transformation methods whose resulting constraints are compact have been proposed (Hasida, 1986; Nakano, 1991). By using these algorithms, we can efficiently analyze using LCGR.

3.3 Grammar Representation

LCGR consists of a set of *phrase structure rules*, a set of *lexical items*, and a *database*.

Each phrase structure rule is a triplate $\langle\ V \rightarrow \xi,\ \mathbf{C}\ \rangle$, where V is a variable, ξ is a list of variables, and $\mathbf{C}$ is a constraint on V and variables in ξ. This means if instances of the variables satisfy constraint $\mathbf{C}$, they form the syntactic structure permitted by this rule. For example, $\langle\ X \rightarrow Y\ Z,\ \{psr1(X,Y,Z)\}\ \rangle$ means if there is a set of instances x, y, and z of X, Y, and Z that satisfies $\{psr1(X,Y,Z)\}$, the sequence of a phrase having feature structure y and that having feature structure z can be recognized as a phrase having feature structure x.

Each lexical item is a pair $\langle w, p\rangle$, where w is a word and p is a predicate. This means an instance of X that satisfies $\{p(X)\}$ can be a lexical feature structure for word w. For example, $\langle walk, lex\_walk\rangle$ means instances of X that satisfy $\{lex\_walk(X)\}$ are lexical feature structures for $walk$.

The database is a set of definite clauses. Predicates used in the constraints and predicates that appear in the bodies of the definite clauses in the database should have their definition clauses in the database.

4 Translation Algorithm

LCGR representation is generated from the grammar in the FF-DDM formalism as follows. (i) Predicates that represent feature values are generated from type definitions. (ii) Phrase structure rules, lexical items, and macro definitions are translated into LCGR elements. (iii) Redundancies are removed from definite clauses by reduction. Below we explain the algorithm through examples.

Creating predicates that represent feature values Let us consider the following type definition.

(12)
```
(deftype sign
    pos agr subj)
```

Then a feature structure of the type *sign* is represented by three-argument term $sign(\_, \_, \_)$, and its arguments represent POS, AGR, and SUBJ features. By using this, the following three definite clauses are created and added to the database.

(13) $pos(sign(X,\_,\_),X) \leftarrow$
$agr(sign(\_,X,\_),X) \leftarrow$
$subj(sign(\_,\_,X),X) \leftarrow$

Translation of phrase structure rules, lexical items, and macro definitions Each of the phrase structure rules, lexical items, and macro definitions is translated into a definite clause and added to the database. This is done as follows.

(I) Create a literal to be the head. In the case of a phrase structure rule and a lexical item, let a newly created symbol be the predicate and all the variables in the third element be the arguments. With macro definition, let the macro name be the predicate and all the variables in the third element be the arguments.

(II) Compute the body by using path equations and disjunctively defined macros, and add the created Horn clause to the database.

(III) By using the predicates created at the step (I), phrase structure rules and lexical items in LCGR are created.

For example, let us consider the following lexical item for verb *walk*.

(14)
```
(defword walk (sign)
    (<sign pos> = verb
     <sign agr> = <sign subj agr>)
    (not3s <sign agr>))
```

First at the step (I), a new predicate $c0$ and LCGR variable *Sign* that corresponds to `sign` are created, $c0(Sign)$ being the head. At the step (II), `<sign pos>` in the second line is replaced by the variable $X1$ and $pos(Sign, X1)$ is added to the body. The symbol `verb` is replaced by the LCGR constant *verb*. Then $eq(X1, verb)$ is added to the body, where eq is a predicate that represents the identity relation and that has the following definition clause.

$eq(X,X) \leftarrow$

As for the third line, the path `<sign agr>` at the left-hand side is replaced by $X2$, `<sign subj agr>` at the right-hand side is replaced by $X4$, and $\{agr(Sign,X2), subj(Sign,X3), agr(X3,X4)\}$ is added to the body. Then $eq(X2,X4)$ is added to the body. For macro `(not3s <sign agr>)`, `<sign agr>` is replaced by $X5$, and $agr(Sign,X5)$ and $not3s(X5)$ are added to the body. Then (15) is added to the database.

(15) $c0(Sign) \leftarrow\{ pos(Sign,X1), eq(X1,verb),$
$agr(Sign,X2), subj(Sign,X3), agr(X3,X4),$
$eq(X2,X4), agr(Sign,X5), not3s(X5)\}$

Finally, $\langle walk, c0\rangle$ is registered as a lexical item. Phrase structure rules and macro definitions are translated in the same way. Horn clause (16) is generated from (2), and $\langle$ $S \rightarrow NP\ VP$, $\{c1(S,NP,VP)\}$ $\rangle$ is registered.

(16) $c1(S,NP,VP) \leftarrow\{ pos(S,X1), eq(X1,sentence),$
$pos(NP,X2), eq(X2,noun), pos(VP,X3),$
$eq(X3,verb), subj(VP,X4), eq(X4,NP),$
$agr(NP,X5), agr(VP,X6), eq(X5,X6),$
$agr(S,X7), agr(VP,X8), eq(X7,X8)\}$

In the same way, Horn clauses (17) are generated from the macro definitions (4) and (5).

(17) $not3s(Agr) \leftarrow\{num(Agr,X1), eq(X1,sing),$
$per(Agr,X2), 1st\_or\_2nd(X2)\}$
$not3s(Agr) \leftarrow\{num(Agr,X1), eq(X1,plural)\}$

In the above translation process, if a macro m has multiple definitions, predicate m' also has multiple definitions. This means disjunctions are not expanded during this process.

Removing Redundancy by Reduction In the definition clauses created by the above proposed method, many predicates that have only one definition clause are used, such as predicate eq, predicates representing feature values, and predicates representing macro that have only one definition. We call these predicates *definite predicates*. If these definition clauses are used in analysis as they are, it will be inefficient because the definition clause of definite predicates must be investigated every time these clauses are used.

Therefore, by using the procedure *reduce* (Tsuda, 1994) each literal whose predicate is definite in the body is replaced by the body of its definition clause.

Let us consider (18) below as an example. If the sole definition clause of $c2$ is (19), $c2(X,Y)$ in (18) is unified with the head of (19). Then, (18) is transformed into (20).

(18) $c1(f(X),Y) \leftarrow\{c2(X,Y)\}$
(19) $c2(g(A,B),Y) \leftarrow\{c3(A), c4(B)\}$
(20) $c1(f(g(A,B)),Y) \leftarrow\{c3(A), c4(B)\}$

By using this operation, Horn clause (15) above is transformed into the following one.

$c0(sign(verb,X6,sign(X7,X6,X8)))$
$\leftarrow\{not3s(X6)\}$

Since $not3s$ has two definitions, $not3s(X6)$ is not replaced. Consequently, the disjunction denoted by $not3s$ is not expanded in this translation.

5 Experiment

The advantage of this method compared to the previous methods is that it can translate without expanding disjunctions. To show this, we compared the time taken for two analyses: the first using a grammar translated

into terms after expanding disjunctions[6] and the second using a grammar translated without expanding disjunctions through our method. The computation times were measured using a bottom-up chart parser (Kay, 1980) in Allegro Common Lisp 4.3 running on Digital Unix 3.2 on DEC Alpha station 500/333MHz. It employs *constraint projection* (Nakano, 1991) as an efficient constraint transformation method. We measured the time for computing all parses. We used a Japanese grammar based on Japanese Phrase Structure Grammar (JPSG) (Gunji, 1987) that covers fundamental grammatical constructions of Japanese sentences. For all of 21 example sentences (5 to 16 words), the time taken for analysis using the grammar translated without disjunction expansion was shorter (43% to 72%). This demonstrates the advantage of our method.

6 Conclusion

This paper presented a method for translating a grammar formalism with disjunctive information that is based on path equations and lists of pairs of labels and values into logical-constraint-based grammar representations, without expanding disjunctions. Although we did not treat type hierarchies in this paper, we can incorporate them by using the method proposed by Erbach (1995).

Acknowledgments

We would like to thank Dr. Ken'ichiro Ishii, Dr. Takeshi Kawabata, and the members of the Dialogue Understanding Research Group for their comments. Thanks also go to Ms. Mizuho Inoue and Mr. Yutaka Imai who helped us to build the experimental system.

References

Hassan Aït-Kaci. 1986. LOGIN: A logic programming language with built-in inheritance. *Journal of Logic Programming*, 3:185–215.

Michael Covington. 1989. GULP 2.0: An extension of Prolog for unification-based grammar. Technical Report AI-1989-01, The University of Georgia.

Jochen Dörre and Andreas Eisele. 1990. Feature logic with disjunctive unification. In *COLING-90*, volume 2, pages 100–105.

A. Eisele and J. Dörre. 1988. Unification of disjunctive feature descriptions. In *ACL-88*, pages 286–294.

Martin C. Emele and Rémi Zajac. 1990. Typed unification grammars. In *COLING-90*, volume 3, pages 293–298.

Gregor Erbach. 1995. ProFIT: Prolog with features, inheritance and templates. In *EACL-95*, pages 180–187.

[6]Note that disjunctions whose elements are all atomic values are not expanded.

Gerald Gazdar and Chris Mellish. 1989. *Natural Language Processing in Lisp: An Introduction to Computational Linguistics*. Addison-Wesley.

Takao Gunji. 1987. *Japanese Phrase Structure Grammar*. Reidel, Dordrecht.

Kôiti Hasida. 1986. Conditioned unification for natural language processing. In *COLING-86*, pages 85–87.

Susan Hirsh. 1988. P-PATR: A compiler for unification-based grammars. In V. Dahl and P. Saint-Dizier, editors, *Natural Language and Logic Programming, II*, pages 63–78. Elsevier Science Publishers.

Robert T. Kasper. 1987. A unification method for disjunctive feature descriptions. In *ACL-87*, pages 235–242.

Martin Kay. 1980. Algorithm schemata and data structures in syntactic processing. Technical Report CSL-80-12, Xerox PARC.

Yuji Matsumoto, Hozumi Tanaka, Hideki Hirakawa, Hideo Miyoshi, and Hideki Yasukawa. 1983. BUP: A bottom-up parser embedded in Prolog. *New Generation Computing*, 1:145–158.

John T. Maxwell and Ronald M. Kaplan. 1991. A method for disjunctive constraint satisfaction. In Masaru Tomita, editor, *Current Issues in Parsing technology*, pages 173–190. Kluwer.

Kuniaki Mukai and Hideki Yasukawa. 1985. Complex indeterminates in Prolog and its application to discourse models. *New Generation Computing*, 3(4):145–158.

Mikio Nakano. 1991. Constraint projection: An efficient treatment of disjunctive feature descriptions. In *ACL-91*, pages 307–314.

Fernando C. N. Pereira and David H. D. Warren. 1980. Definite clause grammars for language analysis—a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 13:231–278.

Carl J. Pollard and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. CSLI, Stanford.

Andreas Schöter. 1993. Compiling feature structures into terms: an empirical study in Prolog. Technical Report EUCCS/RP-55, Centre for Cognitive Science, University of Edinburgh.

Stuart M. Shieber. 1984. The design of a computer language for linguistic information. In *COLING-84*, pages 362–366.

Takenobu Tokunaga, Makoto Iwayama, and Hozumi Tanaka. 1991. Handling gaps in logic grammars. *Trans. of Information Processing Society of Japan*, 32(11):1355–1365. (in Japanese).

Hiroshi Tsuda. 1994. cu-Prolog for constraint-based natural language processing. *IEICE Transactions on Information and Systems*, E77-D(2):171–180.