인지과학, 제 15 권 제 1 호, 2004년

# Extended LR Methods for Efficient Parsing with Feature-based Grammars*

이 강 혁*
(Kang-Hyuk Lee)

**요 약** 본 논문은 자질기반 문법의 LR 파싱 테이블 생성과 관련된 두 가지 문제점을 논의한다. 먼저 자질기반 문법에 LR 파싱 방법을 적용할 경우 발생하는 비종결성 및 비결정성의 문제를 살펴본 후, 이 문제를 해결할 수 있는 LR 방법을 제시한다. 또한 파싱 테이블을 생성하기에는 충분한 정보를 가지지 못하는 하위명시 규칙의 문제점을 지적하고, 이러한 규칙들을 LR 파싱 방법으로 처리할 수 있도록 하는 규칙추론 알고리듬을 제안한다.

***Abstract*** This paper discusses two problems with LR parsing with regard to constructing parsing tables with feature-based grammars. First, we show that traditional LR parsing methods suffer from nontermination and nondeterminism problems when they are applied to feature-based grammars. We then present an LR method for feature-based grammars that avoids both nontermination and nondetermisim by making use of partial information of a feature structure. Second, we describe the problem of adapting LR parsing to feature-based grammars with schematic rules (*i.e.*, rules that do not contain enough information to construct parsing tables). To remedy this problem, we propose a rule inference algorithm which instantiates underspecified rules into more specified ones containing enough information.

## 1 Introduction

With the currency of feature-based grammar formalisms, various parsing methods such as Earley-style chart parsing [13] and Left-corner parsing [15] have been used for natural language processing with them. In this paper, we propose an extended LR-parsing algorithm as an alternative parsing method for efficient parsing with feature-based grammars.

In chapter 2, we present the preliminary algorithm for constructing sets of LR(0) items, and discuss a critical problem with LR-parsing, *i.e.*, the nontermination problem that arises from using feature-based categories. In chapter 3, we provide an intermediate algorithm which gets around the problem addressed in section 3.1, and state another problem of "nondeterminism" even with the extended algorithm given in chapter 2. Chapter 4 presents the complete parsing table construction algorithm for feature-based grammars that avoids both nontermination an nondeterminism by making use of partial information of a feature structure. Chapter 5 discusses a problem with LR-parsing caused by underspecified grammar productions which the radical lexicalization of the grammar brings about [6, 11, 12], and provides an algorithm for rule inference that turns schematic grammar rules into more specified ones so that the parsing table construction algorithm correctly collects sets of items. The brief summary and conclusion of the paper is given in chapter 6.

* 본 연구는 2002년도 동명정보대학교 학술연구비 지원에 의하여 수행됨.
** Department of Game Engineering
Tongmyong University of Information Technology

## 2 Item Set Construction with Feature-based Grammars (I)

### 2.1 Preliminary Algorithm for Constructing Sets of Items

With a grammar with monadic categories, i.e., a pure CFG, sets of items are constructed by consulting categories in LHS and RHS of grammar productions. Given the grammar rules in [Figure 1], the CLOSURE function will compute all possible items (*i.e.*, dotted productions) from an item, say, S → • NP VP, as shown in [Figure 2].

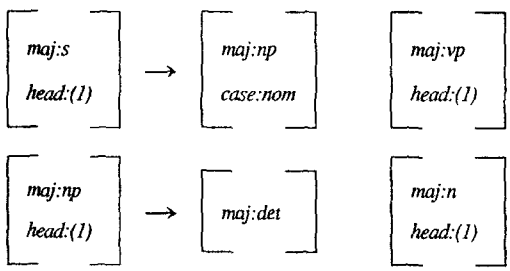
S → NP VP
NP → Det N
NP → NP PP
NP → NP RelCl
VP → V NP
RelCl → RelPro VP
. . .

[Figure 1] Example Productions

S → • NP VP
NP → • Det N
NP → • NP PP
NP → • NP RelCl

[Figure 2] Items created from the production S → NP VP

With feature-based grammar productions, however, this kind of prediction is no longer possible because unlike monadic categories, a new item is added only if there is a grammar production whose LHS is unifiable with the right-of-dot category from which the new prediction is made. The need for unification in predicting items is mainly because in feature-based grammars, categories of the same kind may be different in their internal feature structures. Let us consider the feature-based grammar productions in [Figure 3], which roughly correspond to the first two rules in figure 1. The tags (1), (2), ... etc. indicate *reentrancy*, which means that features tagged by the same integer share their values. As can be clearly seen, the NP feature structure in the RHS of the first production differs from the NP feature structure in the LHS of the second production in which *case* is unspecified. Thus, if the NP in the RHS is the right-of-dot category, we need to unify the feature structure of this category with the feature structure of the *LHS* category in the second production.




[Figure 3] Feature-based Grammar Productions

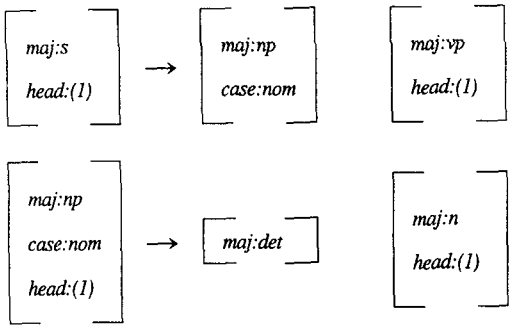
The preliminary CLOSURE/GOTO functions in which unification plays a central role in predicting items are given in [Figure 4]. (∪ indicates the unification operator.) The revised CLOSURE function unifies with the LHS of a production to be predicted (*i.e.*, $C \rightarrow \gamma$) with the category from which that prediction is made (*i.e.*, $B$). Notice that the newly created item is not $C \rightarrow \gamma$, but $C \cup B \rightarrow \delta$ which is the result of unification. $\gamma$ is also changed to $\delta$, because C may be reentrant with respect to some member(s) of $\gamma$.

**procedure** CLOSURE(*I*)
**begin**
**repeat**
**for** each item $[A \rightarrow \alpha \bullet B \beta]$, in $I$, each production $C \rightarrow \gamma$ such that $C$ is unifiable with $B$ and $[C \cup B \rightarrow \bullet \delta]$ is not in $I$ **do**
add $[C \cup B \rightarrow \bullet \delta]$ to $I$;
**until** no more items can added to $I$;
**return** $I$
**end**

**procedure** GOTO(*I*, *C*)
**begin**
let $J$ be the set of items $[A \rightarrow \alpha X \bullet \beta]$, such that $[A \rightarrow \alpha \bullet X \beta]$ is in $I$, and $X$ is unifiable with $C$;
**return** CLOSURE(*J*)
**end**

[Figure 4] LR(0) CLOSURE/GOTO functions for feature-based grammars

With this revised algorithm, we are now able to construct dotted productions through unification as in [Figure 5]. Note



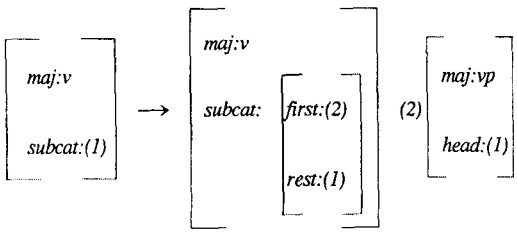

[Figure 5] Items created from the right-of-the-dot category NP

that the LHS of the second item which is predicted from the right-of-the-dot category in the first item is instantiated to have the *case* feature, which is the result of unifying with the right-of-the-dot category.

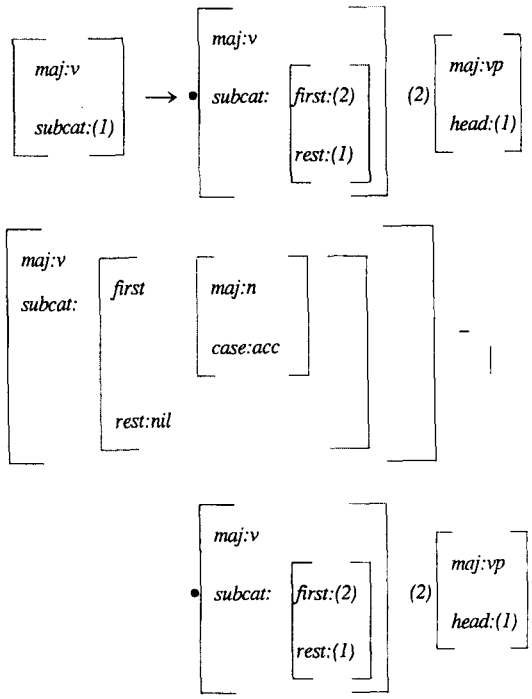
### 2.2 Nontermination Problem

Shieber [13] discusses the nontermination problem that arises when we try to apply standard parsing methods to feature-based grammar formalisms. For example, the straightforward adaptation of Earley's algorithm to those formalisms which allow for infinite nonterminal domains (*e.g.*, list-valued subcategorization in HPSG (Head-driven Phrase Structure Grammar [11, 12] and PATR-style grammars [14]) or slashed functor categories in CUG (Categorial Unification Grammar) [11, 18]) leads to nontermination during parsing.

The same problem arises with LR-parsing algorithms, which make top-down predictions in terms of preprocessing of grammars. They fail to terminate at compile time. Given the grammar production in [Figure 6], for instance, the relevant syntactic category that is unifiable with the LHS of the production will create the first item (or prediction) in [Figure 7]. Since the right-of-the-dot category of this item itself is




[Figure 6] Example Grammar Production




[Figure 7] Items created from the production in figure 7

unifiable with the LHS of the given grammar production, it predicts the second item in [Figure 7], and thus the loop in the CLOSURE function generates an infinite number of new items from the same grammar production.

## 3 Item Set Construction with Feature-based Grammars (II)

### 3.1 Intermediate Algorithm for Constructing Sets of Items

Shieber [13] proposes a technique called "restriction" as a workaround to this problem. Although he demonstrates this technique with the Predictor step of Earley's algorithm, he also suggests that it could be used for LR-parsing algorithms to guarantee the termination of the LR closure function, which can be thought of as a precompiled version of Earley's Predictor.[1] Nakazawa [10] proposed an extended LR algorithm that uses the unification operation while constructing the parsing table of a grammar. Another way of

1) See Aho and Johnson [1] for the similarity between Earley's algorithm and the LR algorithm.

avoiding nontermination during preprocessing of grammars is to predict dotted productions without instantiation of grammar productions through unification. With this method, unification only serves to check if an item to be predicted is legitimate. The revised CLOSURE/GOTO procedures are given in [Figure 8]. It differs only from the algorithm in the previous chapter in adding newly created items without instantiation through unification, thus preventing the CLOSURE function from falling into a left-recursion trap.

**procedure** CLOSURE(*I*)
**begin**
**repeat**
  **for** each item $[A \rightarrow \alpha \bullet B \beta]$, in *I*, each production $C \rightarrow \gamma$ such that *C* is unifiable with *B* and $[C \rightarrow \bullet \gamma]$ is not in *I* do
    add $[C \rightarrow \bullet \gamma]$ to *I*;
**until** no more items can added to *I*;
**return** *I*
**end**

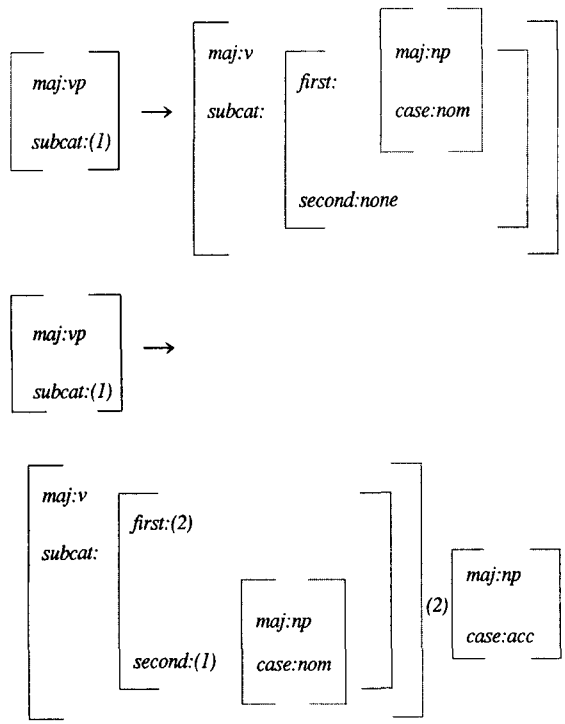
**procedure** GOTO(*I*, *C*)
**begin**
let J be the set of items $[A \rightarrow \alpha X \bullet \beta]$, such that $[A \rightarrow \alpha \bullet X \beta]$ is in *I*, and *X* is unifiable with *C*;
**return** CLOSURE(*J*)
**end**

[Figure 8] LR(0) CLOSURE/GOTO functions (Revised)

Given the first item in [Figure 7], the CLOSURE function will try to predict another item, since the right-of-the-dot category is unifiable with the LHS of the grammar production. However, this new item will not be added because the same dotted production is already in the current set of items. Although the CLOSURE/GOTO functions in [Figure 8] get around the nontermination problem, they reveal another problem of nondeterminism, which we focus on in the next section.

### 3.2 Source of Nondeterminism

The LR(0) CLOSURE functions given in [Figure 8] avoid the nontermination problem by adding dotted productions without instantiating grammar rules through unification. LR-parsing with feature-based grammars, however, faces another problem of nondeterminism that does not arise with grammars using atomic categories. In PATR-style grammars, verbs are often distinguished by subcategori-zation frames, but not by category labels. Suppose that a grammar has the two VP rules in [Figure 9] which roughly correspond to ordinary context-free rules VP ➜ V and VP ➜ V NP.




[Figure 9] Example Grammar Productions

| state | np | v[np] | v[np,np] | $ | vp[np] | s[] |
|---|---|---|---|---|---|---|
| 0 | sh2 | | | | | 1 |
| 1 | | | | acc | | |
| 2 | | sh4 | sh5 | | 3 | |
| 3 | | | | re1 | | |
| 4 | | | | re2 | | |
| 5 | sh6 | | | | | |
| 6 | | | | re3 | | |

[Figure 10] LR Parsing Table

In the construction of the parsing table, these two rules will lead to two separate *shift* actions in the parsing table, both of which start at the same state but result in difference states, as

shown in [Figure 10].[2)]

The problem with feature-based grammars is that we cannot consult the parsing table deterministically. In shifting the verb *loves* in *John loves Mary* onto the stack, for example, the parser sees two possible entries in the parsing table. Unlike parsing tables with atomic categories, we cannot deterministically choose one of these two because the preterminals in the parsing table in [Figure 10] are partially instantiated by the CLOSURE function, while the verb *loves* contains the full-fledged feature structure, part of which might be absent from the corresponding preterminal in the parsing table. Thus, it is only after unification test with these two preterminals that the parser shifts *loves* onto the stack and proceeds to the desired state. This kind of nondeterminism, which goes against the spirit of LR-parsing, has significant effects on the efficiency of the parser because the unification operation is expensive.

## 4 Parsing Table Construction with Feature-based Grammars

We avoid the nondeterminism problem by utilizing partial information of a feature structure, specifically part-of-speech information, when predicting items. This requires modification of the GOTO function. As shown in the GOTO function in [Figure 8], moving the dot is made possible by unification test. Since *sleeps* and *loves* carry different subcategorization frames, they are considered to be different categories, which means that they are to be treated by separate GOTO functions. We modify the GOTO function in [Figure 8] to embed categories of the same kind into a single GOTO action when the parser attempts to move the dot. Taking the grammar rules in [Figure 9] as an example, we make use of the *maj* path which specifies part-of-speech information. Given the revised GOTO function in [Figure 11], we are able to put into the same set of items the two items in which the dot has been moved to the right of the verb, even though the two verbs carry different subcategorization frames.[3)]

**procedure** CLOSURE(*I*)
**begin**
**repeat**
  **for** each item $[A \rightarrow \alpha \bullet B \beta]$, in *I*, each production $C \rightarrow \gamma$
    such that *C* is unifiable with B and $[C \rightarrow \bullet \gamma]$ is not in *I* do
      add $[C \rightarrow \bullet \gamma]$ to *I*;
**until** no more items can added to *I*;
**return** *I*
**end**

**procedure** GOTO(*I*, *C*)
**begin**
let J be the set of items $[A \rightarrow \alpha X \bullet \beta]$, such that $[A \rightarrow \alpha \bullet X \beta]$ is in *I*, and *MAJ*(*X*) = *C*;
such that *MAJ*(*X*) is the path in *X* containing part-of-speech information;
**return** CLOSURE(*J*)
**end**

[Figure 11] The Final CLOSURE/GOTO functions

GOTO($I_2$, [maj v]): $vp[np_1]$ $v[np_1]$
$vp[np_2]$ $v[np_1,np_2]$ $np[np_1]$

[Figure 12] GOTO function on $I_2$

One might argue that the nondeterminism problem remains because the parser has to choose between two possible actions after pushing a verb. If there are no words left to be pushed, the elements in the stack should be reduced to S. Otherwise, the parser has to shift another NP onto the stack. This is still true if we rely on unification to determine the next action (*i.e.*, unifying the feature structure of the current word with the feature structure(s) the state indicates in the table). As can be inferred from [Figure 12], however, what we put into the parsing table is *MAJ* (*i.e.*, [*maj v*] in this particular case). If we put *v*[*np*] and *v*[*np,np*] into the parsing table, the GOTO function in [Figure 11] is vacuous. It will result in the two

2) The category labels in the parsing table abbreviate the corresponding feature structures. The lists of category labels represent subcategorized-for elements to be saturated.

3) The way to encode part-of-speech information varies. The *maj* feature is used in HPSG. In Categorial Grammar, it is expressed in terms of category symbols and directional slashes. Hence, *MAJ* in the revised GOTO function should be regarded as a function representing the part-of-speech information in various grammar formalisms. *MAJ* can be easily fine-tuned depending on the grammar formalism adopted.

separate entries whose current and next states are the same. A look at the parsing table construction algorithm will make this point easy to understand. Unlike the original formulation in Aho and Ullman [2], the right-of-the-dot category $\alpha$ in $[A \rightarrow \alpha \bullet \alpha\ \beta]$ is not identified with $X$ in GOTO($I$, $X$). Rather, we check if the *MAJ* of the right-of-the-dot category is the same as $X$ in GOTO($I$, $X$). Then, we put $X = MAJ(\alpha)$ in the parsing table rather than the whole feature structure of $\alpha$. Therefore, when the parser tries to shift a word, it uniquely matches the current state and the *MAJ* of the word with an entry in the parsing table.

(1) If $[A \rightarrow \alpha \bullet \alpha\ \beta]$ is in $I_i$, GOTO($I_j$, $MAJ(\alpha)$) = $I_j$,
(2) If $[A \rightarrow \alpha \bullet]$ is $I_i$,
then set ACTION[$i$, $MAJ(\alpha)$] to "reduce $A \rightarrow \alpha$" for all $\alpha$ in FOLLOW(A).
(3) If $[S' \rightarrow S \bullet]$ is in $I_i$,
then set ACTION[$i$, \$] to "accept".

(4) GOTO($I_i$, $A$) = , then GOTO[$i$, $A$] = $j$.
(5) All entries not defined by rules (1) through (4) are made "error".
(6) The initial state of the parser is the one constructed from the set of items containing $[S' \rightarrow S \bullet]$.

[Figure 13] Parsing Table Construction Algorithm for Feature-based Grammars

This revised table construction algorithm in which part-of-speech information plays a central role enables us to consult the parsing table deterministically without any unification work. A question that arises is how we can deal with other parts of the feature structure that are crucial to determine the grammaticality of sentences. For example, if we solely depend on the parsing table to parse sentences, there is no way to block sentences such as *John love Mary* in which subject-verb agreement is violated. Unlike standard LR-parsing algorithms for grammars with monadic categories, the structure of a sentence in feature-based grammars is built by way of unification. When the parser tries to reduce a verb phrase and a noun phrase to a sentence, for instance, the parsing table only tells which rule to apply, and nothing more than that. The feature structure of a sentence is built through unification of the NP and VP feature structures with the designated rule. It is through this unification operation that the parser detects the subject-verb agreement violation, if any. This is a significant departure from standard LR-parsing methods in which the structure of a sentence is built in terms of putting elements popped out of the stack under the category symbol they reduce to. Errors are only detected by entries not defined in the parsing table. In our LR-parsing method, however, errors are also detected when building a constituent structure through unification. Thus, a significant amount of work is relegated to the driver routine in our parser. The parsing table still guides the parser efficiently in terms of assigning a correct action as it proceeds.

## 5 LR Parsing and Schematic Rules

### 5.1 Problem with Schematic Rules

In current lexically-based grammars such as HPSG (Head-driven Phrase Structure Grammar) [3, 11, 12] and CUG (Categorial Unification Grammar) [9, 18], grammar productions are highly schematic. In HPSG, for example, the grammar has only a few rule schemata. In this respect, CUG goes to the extreme. In CUG, only a couple of functional operations are required as grammar rules. In either case, there are no construction-specific grammar rules, and almost all the grammatical information is pushed in the lexicon.

This lexicalization of grammar, often called radical lexicalism, makes it very difficult to apply LR parsing to those grammar formalisms, since grammar productions are too underspecified for the LR parsing algorithm to correctly close sets of items. An example from categorial grammar will illustrate this problem. Let us assume CG to have only tow functional operations as grammar productions.

| | |
|---|---|
| $X \rightarrow X/Y\ \ Y$ | np → {John, Mary} |
| $X \rightarrow Y\ \ X\backslash Y$ | s\np → sleeps |
| | (s\np)/np → loves |

[Figure 14] Example Categorial Grammar

Given the grammar in [Figure 14], the partial sets of items will be like those in figure 15. The second item in $I_0$ will result in the set of items $I_2$ by virtue of the GOTO function. The parsing table construction algorithm will eventually put a shift entry (0, X/Y, 1) in the action table from the second item in $I_0$ and GOTO($I_0$, $X/Y$) = $I_2$. With the grammar in [Figure 14], a sentence cannot begin with a lexical item

whose category is a functor. This shows that the straightforward application of the LR parsing method to underspecified rules does not guarantee that the correct sets of items will be constructed.

$I_0$ : [ S' → • S ]
[ X → • X/Y Y ]
[ X → • Y X\Y ]
...
$I_1$ = GOTO($I_0$, S): [ S' → S • NP ]
$I_2$ = GOTO($I_2$, X/Y): [ X → X/Y • Y ]
[ X → • X/Y Y ]
[ X → • Y X/Y ]
...

[Figure 15] Sets of Items

## 5.2 Rule Inference

As seen in the previous section, the LR parsing method does not lend itself to predicting items with underspecified grammar rules as in lexicalized grammars such as HPSG and CUG. To remedy this problem, we propose a rule inference algorithm that supplements the LR parsing table construction algorithm. The main idea is that the rule inference system instantiates underspecified rules into more specified ones containing enough information to correctly close sets of items. This naturally leads to an increase in the number of grammar rules, which often increases the size of the parsing table substantially. It might look a bit ugly from the theoretical standpoint, but it does not cause problems with respect to efficiency because the table size does not affect parsing time.

Rule inference depends on partial feature structures of lexical items (roughly speaking, the set of preterminals). To begin with, the inference algorithm instantiates underspecified rules with the partial preterminal feature structures extracted from the lexicon according to a "restrictor".[4] Then it goes on to infer higher levels of rules (*i.e.*, rules whose daughters can dominate the mother categories of previously inferred rules) from these initially inferred rules until it gets to sentence-level rules. The rule inference algorithm is given in [Figure 16], where ∪ indicates the unification operation.

**procedure** RULE-INFERENCE ($R''$)
**begin**
let $N$ be the set of nonterminals according to a restrictor,
$R$ the set of grammar productions, and
$R$ the set of inferred grammar productions
such that it is initially empty;
**repeat**
**for** each grammar production
$[x_0\ x_1 \ldots x_n]$ in $R$ **do**
**for** each daughter $x_k$ **do**
for each nonterminal $nt$ in $N$ **do**
$\mathbf{R'}_k = [x_0 \ldots x_k \cup nt \ldots]$;
**for** each set of productions
$\{p_1, p_2, \ldots, p_n\}$ in $\Pi\ r$ **do**
$r \in R'$
add $P$ to $R''$ such that
$P$ is the result of $p_1 \cup p_2 \cup \ldots \cup p_n$;
add the restricted LHS of $P$ to $N$;
**until** no more productions can be added to $R''$;
**return** $R''$
**end**;

[Figure 16] Rule Inference Algorithm

$N$ initially denotes the set of partial preterminal feature structure according to a restrictor, and is augmented with the mother categories of inferred rules as the loop goes on. Informally stated, the inference algorithm generates the cross product of sets, each of which contains the results of instantiating a daughter in a rule with available nonterminal feature structures. If each element in each set in the product is unifiable with others, then the result of unification is added to the inventory of inferred rules. For example, the PATR-style [14] rule schema in [Figure 17] in which *x2* subcategorizes for *x1* is instantiated as productions responsible for generating noun phrase, sentence, etc. in the sense of phrase structure grammars. Assuming that a noun subcategorizes for a determiner, the rule schemata is first expanded as NP by way of unifying its daughters with Det and Noun. (More precisely, the step involves unifying each

4) Shieber [13] originally proposed restriction to ensure a terminating algorithm for feature-based grammars. The restrictor (*i.e.*, a finite set of paths) in rule inference serves the same purpose. If we make use of the complete set of lexical entries as preterminals, the inference algorithm will not terminate when it tries to instantiate underspecified rules with infinite nonterminal domains.

daughter of the rule schemata with preterminal feature structures. The inferred rule is the result of unifying each partially instantiated rule with others) Then, the mother category of this inferred rule, *i.e.*, NP is added to the set of nonterminal feature structures *N*. This NP is also used to infer the S rule with VP to be inferred from V and NP. The two inferred rules are given in [Figure 18].[5] Note that NP and S are realized as feature structures whose *maj* value respectively is *n* and *v*, but which do not subcategorize for anything, *i.e.*, which are saturated.

```
x0  x1  x2
<x0 maj> = <x2 maj>
<x0 subcat> = <x2 subcat rest>
<x2 subcat first> = <x1>
<x2 subcat rest> = nil
```

[Figure 17] Sample Rule Schemata

```
x0 → x1  x2
<x0 maj> = <x2 maj>
<x0 subcat> = <x2 subcat rest>
<x2 maj> = n
<x2 subcat first> = <x1>
<x2 subcat first maj> = det
<x2 subcat rest> = nil

x0 → x1  x2
<x0 maj> = <x2 maj>
<x0 subcat> = <x2 subcat rest>
<x2 maj> = v
<x2 subcat first> = <x1>
<x2 subcat first maj> = n
<x2 subcat rest> = nil
```

[Figure 18] Inferred Rules

The rule inference algorithm works bottom-up, initially unifying the daughters of each schematic rule with possible preterminal feature structures. If these initial instantiations succeed, the set of possible mother category (*i.e.*, the feature structure for the LHSs of schematic rules (again, according to a restrictor)) will be obtained which are used for inferring higher levels of rules. As described above, after unifying the feature structures for Det and N with the daughters of the schematic rule in [Figure 17], the LHS of the rule has *n* as its value for *maj*, because the *maj* value of the mother (*i.e.*, x0) is reentrant with that of the second daughter (*i.e.*, x2). However, the mother category is distinguished from the daughter by the degree of saturation in subcategorization. The way of instantiating schematic rules assumes that a rule must be defined in such a way that after instantiations of daughter categories, the inference algorithm will get the mother category specific enough to instantiate other possible rules. Let us consider the schematic rule in [Figure 19], which is identical with the rule in [Figure 17] except that the information on the *maj* feature is missing.

```
x0 → x1  x2
<x0 subcat> = <x2 subcat rest>
<x2 subcat first> = <x1>
<x2 subcat rest> = nil
```

[Figure 19] Example Rule Schemata

Although the algorithm is still able to unify Det and N with the daughters, the feature structure of the LHS still remains quite underspecified because the result of unification does not lead to the LHS with the part-of-speech information specified. Underspecified LHS categories of this kind causes problem with the table construction algorithm which relies on the part-of-speech information to collect sets of items. From the practical point of view, it is not likely that we run into this kind of problem. It seems unlikely that we would have a grammar rule such that we cannot decide what the mother category of the rule is, after fully expanding its daughters with relevant information. Chances are that the part-of-speech information of each category in a rule is independently specified or some information on the relations between categories is specified (*e.g.*, via reentrancy). Nevertheless, the rule schema in [Figure 19] is a legitimate rule according to the PATR formalism, and the inference algorithm cannot handle rule schemata of the kind properly. Thus, the application of the rule inference algorithm is limited to the class of grammars where categories involved in a rule contain the information defined by a restrictor independently or indirectly (*i.e.*, by reentrancy). This ensures that after

5) It should be noted that with a categorial grammar including general type-raising rules such as X/(X\Y) Y, the inference algorithm will not terminate. However, it is common practice in CG to use more restricted types of raising rules such as S/(S\NP) NP instead of general raising rules.

instantiating the RHS categories of a rule, the LHS category will also have necessary information used for instantiating other rules.

## 6 Summary and Conclusion

We have discussed two problems with LR parsing with regard to constructing parsing tables with feature-based grammars. First, we have shown that the blind application of LR methods to feature-based grammars causes the problems of nontermination and nondeterminism. To get around these problems, we have proposed a new LR method that does not bring about nontermination and nondetermisim. Second, we have described the problem of adapting LR parsing to feature-based grammars with schematic rules. To remedy this problem, we have proposed a rule inference algorithm which instantiates underspecified rules into more specified ones containing enough information to correctly close sets of items.

The LR method for feature-based grammars that we have proposed is important in that it enables us to maintain the efficiency of LR parsing without worrying about the problems of nontermination and non-determinism. The rule inference algorithm is also important in the viewpoint of grammar writing. It frees the grammar writer from unnecessary computational considerations that are not directly relevant to the grammar formalism adopted. The grammar writer can be faithful to the grammar formalism he/she adopts (he/she can make a grammar as much general as he/she wants.), and the inference engine will take care of everything else so that the LR algorithm correctly close sets of items.

## References


[1] Aho, A. V. and S. C. Johnson 1974. "LR parsing". *Computing Surveys*, 6(2):99-124.

[2] Aho, A. V. and J. D. Ullman 1977. *Principles of Compiler Design*. Addison Wesley, Reading, MA.

[3] Bouma, Gosse, Frank Van Eynde & Dan Flickinger 2000. "Constraint-based Lexica", in Frank Van Eynde and Dafydd Gibbon, *Lexicon Development for Speech and Language Processing*, Kluwer Academic Publishers, Dordrecht-Boston-London, pp. 43-75.

[4] Briscoe, T. and J. Carroll 1993 . "Generalized Probabilistic LR Parsing of Natural Language (Corpora) with Unification-based Grammars". *Computational Linguistics*, 19(1): 25-57.

[5] Earley, J. 1970 "An Efficient Context-Free Parsing Algorithm". *Communications of the ACM*, 13(2):94-102.

[6] Karttunen, L. 1986. "Radical Lexicalism". *Paper presented at the Conference on Alternative Conception of Phrase Structure.*

[7] Kindermann and Meier 1988. "An Extension of LR-Parsing for Lexical Functional Grammar". In U. Reyle and C. Rohrer, editors, *Natural Language Parsing and Linguistic Theories*, 131-148. Reidel, Dordrecht.

[8] Kipps, J. R. 1991. "GLR Parsing in Time $O(n^3)$". In M. Tomita, editor, *Generalized LR Parsing*, 43-59. Kluwer, Dordrecht.

[9] Milward, D. 1995. "Incremental Interpretation of Categorial Grammar". *Proceedings of EACL '95*, 119-216.

[10] Nakazawa, Tsuneko. 1991. "An Extended LR Parsing Algorithm for Grammars Using Feature-based Syntactic Categories". *Proceedings of the ACL European Chapter*, 69-74.

[11] Pollard, C and I. Sag 1987. *Information-based Syntax and Semantics. Volume 1: Fundamentals*. CSLI, Stanford, CA.

[12] Pollard, C. and I. Sag 1994. *Head-driven Phrase Structure Grammar*. CSLI, Stanford, CA.

[13] Shieber, S. 1985. "Using Restriction to Extend Parsing Algorithms for Complex-feature-based Formalisms". *Proceedings of ACL*, 145-152.

[14] Shieber, S. 1986. *An Introduction to Unification-Based Approaches to Grammar*. CSLI, Stanford, CA.

[15] Shieber, S. and F. Pereira 1987. *Prolog and Natural Language Analysis*. CSLI, Stanford, CA.

[16] Tomita, M. 1986. *Efficient Parsing for Natural Language: A Fast Algorithm for Practical Systems*. Kluwer Academic Press, Boston, MA.

[17] Tomita, M. 1987. "An Efficient Augmented Context-Free Parsing Algorithm". *Computational Linguistics*, 13(1-2): 1-46.

[18] Uzkoreit, H. 1986. "Categorial Unification Grammar". *Proceedings of Coling '86*, 187-194.