AN EXTENDED LR PARSING ALGORITHM FOR GRAMMARS USING FEATURE-BASED SYNTACTIC CATEGORIES

Tsuneko Nakazawa
Beckman Institute for Advanced Science and Technology
and
Linguistics Department
University of Illinois
4088 FLB, 707 S. Mathews, Urbana, IL 61801, USA
tsuneko@grice.cogsci.uiuc.edu

ABSTRACT

This paper proposes an LR parsing algorithm modified for grammars with feature-based categories. The proposed algorithm does not instantiate categories during preprocessing of a grammar as proposed elsewhere. As a result, it constructs a minimal size of GOTO/ACTION table and eliminates the necessity of search for GOTO table entries during parsing.

1 Introduction

The LR method is known to be a very efficient parsing algorithm that involves no searching or backtracking. However, recent formalisms for syntactic analyses of natural language make maximal use of complex feature-value systems, rather than atomic categories that have been presupposed in the LR method. This paper is an attempt to incorporate feature-based categories into Tomita's extended LR parsing algorithm (Tomita 1986).

A straightforward adaptation of feature-based categories into the algorithm introduces the necessity of partial instantiation of categories during preprocessing of a grammar as well as a nontermination problem. Furthermore, the parser is forced to search through instantiated categories for desired GOTO table entries during parsing. The major innovations of the proposed algorithm include the construction of a minimal size of GOTO table that does not require any preliminary instantiation of categories or a search for them, and a reduce action which performs instantiation during parsing.

Some details of the LR parsing algorithm are assumed from Aho and Ullman (1987) and Aho and Johnson (1974), and more formal definitions and notations of a feature-based grammar formalism from Pollard and Sag (1987) and Shieber (1986).

2 The LR Parsing Algorithm

The LR parser is an efficient shift-reduce parser with optional lookahead. Parse trees for input strings are built bottom-up, while predictions are made top-down prior to parsing. The ACTION/GOTO table is constructed during preprocessing of a grammar and deterministically guides the parser at each step during parsing. The ACTION table determines whether the parser should take a shift or a reduce action next. The GOTO table determines the state the parser should be in after each action.

Henceforth, entries for the ACTION/GOTO table are referred to as the values of functions, ACTION and GOTO. The ACTION function takes a current state and an input string to return a next action, and GOTO takes a previous state and a syntactic category to return a next state.

States of the LR parser are sets of dotted productions called items. The state, i.e. dotted productions, stored on top of the stack is called current state and the dot positions on the right hand side (rhs) of the productions indicate how much of the rhs the parser has found. Previous states are stored in the stack until the entire rhs, or the left hand side (lhs), of a production is found, at which time a reduce action pops previous states and pushes a new state in, i.e. the set of items with a new dot position to the right, reflecting the discovery of the lhs of the production.

If a grammar contains two productions VP→V NP and NP→Det N, for example, then the state *s1* in Fig.1(i) (the state numbers are arbitrary) should contain the items <VP→V . NP> and <NP→.Det N> among others, after shifting an input string "saw" onto the stack. The latter item predicts strings that may follow in a top-down manner.

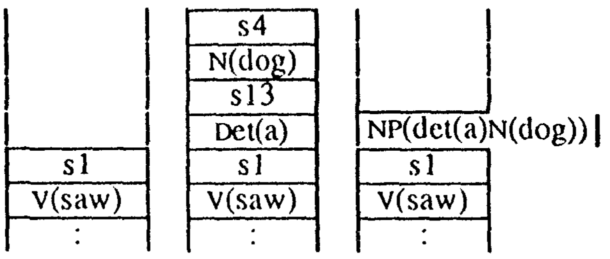

Figure 1. Stacks

After two more strings are shifted, say "a dog", and the parser encounters the end-of-a-sentence symbol "$" (Fig.1(ii)), the next action, ACTION(s4,$), should be "reduce by NP→Det N". The reduce action pops two states off the stack, and builds a constituent whose root is NP (Fig.1(iii)). At this point, GOTO(s1,NP) should be a next state that includes the item <VP→V NP . >.

The ACTION/GOTO table used in the above example can be constructed using the procedures given in Fig.2 (adapted from Aho and Ullman (1987)). The procedure CLOSURE computes all items in each state, and the procedure NEXT-S, given a state and a syntactic category, calculates the next state the parser should be in.

```
procedure CLOSURE(I);
begin
    repeat
    for each item <A→w.Bx> in I, and each
    production B→y such that <B→.y> is not
    in I do
        add <B→.y>to I;
    until no more items can be added to I;
    return I
end;
```

```
procedure NEXT-S(I,B)
;for each category B in grammar
begin
    let J be the set of items <A→wB.x>
    such that <A→w.Bx> is in I;
    return CLOSURE(J)
end;
```

Figure 2. CLOSURE/NEXT-S Procedures for Atomic Categories

It should be clear from the preceding example that upon the completion of all the constituents on the rhs of a production, the GOTO table entry for the lhs is consulted. Whether a category appears on the lhs or the rhs of productions is a trivial question, however, since in a grammar with atomic categories, every category that appears on the lhs also appears on the rhs and vice versa. On the other hand, in a grammar with feature-based categories, as proposed by most recent syntactic theories, it is no longer the case.

3 Construction of the GOTO Table for Feature-Based Categories: A Preliminary Modification

Fig.3 is an example production using feature-based syntactic categories. The notations are adapted from Pollard and Sag (1987) and Shieber (1986). The tags $\boxed{1}$, $\boxed{2}$,... roughly correspond to variables of logic unification with a scope of single productions: if one occurrence of a particular tag is instantiated as a result of unification, so are other occurrences of the same tag within the production.

$$\begin{bmatrix}\text{CAT} & \text{V}\\ \text{SUBCAT} & \boxed{1}\\ \text{TENSE} & \boxed{3}\end{bmatrix} \rightarrow \begin{bmatrix}\text{CAT} & \text{V}\\ \text{SUBCAT} & \begin{bmatrix}\text{FIRST} & \boxed{2}\text{NP}\\ \text{REST} & \boxed{1}\end{bmatrix}\\ \text{TENSE} & \boxed{3}\end{bmatrix} \ \boxed{2}\text{NP}$$

Figure 3. Example Production

Recursive applications of the production assigns the constituent structure to strings "gave boys trees" in Fig.4. The assumed lexical category for "gave" is given in Fig.5.

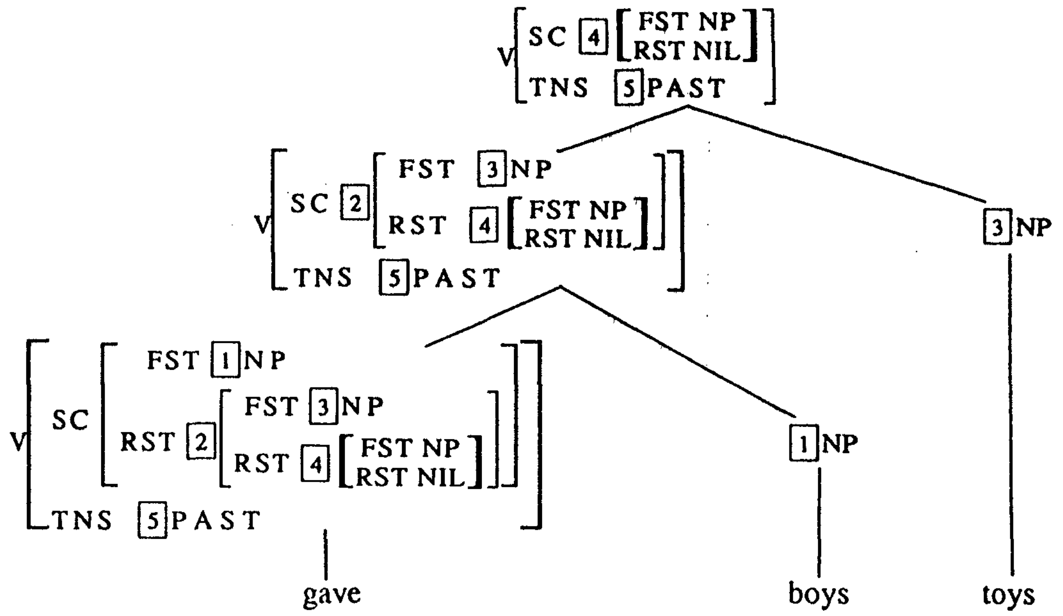
V [SC 4 [FST NP, RST NIL], TNS 5 PAST]
V [SC 2 [FST 3 NP, RST 4 [FST NP, RST NIL]], TNS 5 PAST]
3 NP
V [SC [FST 1 NP, RST 2 [FST 3 NP, RST 4 [FST NP, RST NIL]]], TNS 5 PAST]
1 NP
gave
boys
toys

Figure 4. Example Parse Tree

$$V\begin{bmatrix} \text{SC} & \begin{bmatrix} \text{FST} & \text{NP} \\ \text{RST} & \begin{bmatrix} \text{FST} & \text{NP} \\ \text{RST} & \begin{bmatrix} \text{FST} & \text{NP} \\ \text{RST} & \text{NIL} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{TNS} & \text{PAST} \end{bmatrix}$$

Figure 5. Lexical Category for "gave"

In grammars that use feature-based syntactic categories, categories in productions are taken to be underspecified: that is, they are further instantiated through the unification operation during parsing as constituent structures are built. The preterminal category for "gave" in Fig.4 is the result of unification between the lexical category for "gave" in Fig.5 and the first category on the rhs of the production in Fig.3. This unification also results in the instantiation of the lhs through the tags. The category for the constituent "gave boys" is obtained by unifying the instantiated lhs and the first category of the rhs of the same production in Fig.3. In order to accommodate the instantiation of underspecified categories, the CLOSURE and NEXT-S procedures in Fig.2 can be modified as in Fig.6, where ∧ is the unification operator.

```
procedure CLOSURE(I);
begin
    repeat
    for each item <A→w.Bx> in I, and each
    production C→y such that C is unifiable
    with B and <C∧B→.y'> is not in I do
        add <C∧B→.y'> to I;
    until no more items can be added to I;
    return I
end;

procedure NEXT-S(I,C)
; for each category C that appears to the right
; of the dot in items
begin
    let J be the set of items <A→wB.x> such
    that <A→w.Bx> is in I and B is unifiable
    with C;
    return CLOSURE(J)
end;
```

Figure 6. Preliminary CLOSURE/NEXT-S Procedures

The preliminary CLOSURE procedure unifies the lhs of a predicted production, i.e.

C→y, and the category the prediction is made from, i.e. B. This approach is essentially top-down propagation of instantiated features and well documented by Shieber (1985) in the context of Earley's algorithm. A new item added to the state, <C∧B→. y'>, is not the production C→y, but its (partial) instantiation. y is also instantiated to be y' as a result of the unification C∧B if C and some members of y share tags. Thus, given the production in Fig.3 and a syntactic category V[SC NIL] to make predictions from, for example, the preliminary CLOSURE procedure creates new items in Fig.7 among others. The items in Fig.7 are all different instantiations of the same production in Fig.3.

$$<\mathrm{V}\left[\begin{array}{ll}\mathrm{SC} & \boxed{1}\mathrm{NIL}\\ \mathrm{TNS} & \boxed{3}\end{array}\right]\rightarrow \;.\;\mathrm{V}\left[\begin{array}{ll}\mathrm{SC} & \left[\begin{array}{ll}\mathrm{FST} & \boxed{2}\mathrm{NP}\\ \mathrm{RST} & \boxed{1}\mathrm{NIL}\end{array}\right]\\ \mathrm{TNS} & \boxed{3}\end{array}\right]\;\boxed{2}\mathrm{NP}>$$

$$<\mathrm{V}\left[\begin{array}{ll}\mathrm{SC} & \boxed{1}\left[\begin{array}{ll}\mathrm{FST} & \mathrm{NP}\\ \mathrm{RST} & \mathrm{NIL}\end{array}\right]\\ \mathrm{TNS} & \boxed{3}\end{array}\right]\rightarrow \;.\;\mathrm{V}\left[\begin{array}{ll}\mathrm{SC} & \left[\begin{array}{ll}\mathrm{FST} & \boxed{2}\mathrm{NP}\\ \mathrm{RST} & \boxed{1}\left[\begin{array}{ll}\mathrm{FST} & \mathrm{NP}\\ \mathrm{RST} & \mathrm{NIL}\end{array}\right]\end{array}\right]\\ \mathrm{TNS} & \boxed{3}\end{array}\right]\;\boxed{2}\mathrm{NP}>$$

$$<\mathrm{V}\left[\begin{array}{ll}\mathrm{SC} & \boxed{1}\left[\begin{array}{ll}\mathrm{FST} & \mathrm{NP}\\ \mathrm{RST} & \left[\begin{array}{ll}\mathrm{FST} & \mathrm{NP}\\ \mathrm{RST} & \mathrm{NIL}\end{array}\right]\end{array}\right]\\ \mathrm{TNS} & \boxed{3}\end{array}\right]\rightarrow$$

$$.\;\mathrm{V}\left[\begin{array}{ll}\mathrm{SC} & \left[\begin{array}{ll}\mathrm{FST} & \boxed{2}\mathrm{NP}\\ \mathrm{RST} & \boxed{1}\left[\begin{array}{ll}\mathrm{FST} & \mathrm{NP}\\ \mathrm{RST} & \left[\begin{array}{ll}\mathrm{FST} & \mathrm{NP}\\ \mathrm{RST} & \mathrm{NIL}\end{array}\right]\end{array}\right]\end{array}\right]\\ \mathrm{TNS} & \boxed{3}\end{array}\right]\;\boxed{2}\mathrm{NP}>$$

Figure 7. Items Created from the Same Production in Figure 3

As can be seen in Fig.7, the procedure will add an infinite number of different instantiations of the same production to the state. The list of items in Fig.7 is not complete: each execution of the repeat-loop adds a new item from which a new prediction is made during the next execution. That is, instantiation of productions introduces the nontermination problem of left-recursive productions to the procedure, as well as to the Predictor Step of Earley's algorithm. To overcome this problem, Shieber (1985) proposes "restrictor", which specifies a maximum depth of feature-based categories. When the depth of a category in a predicted item exceeds the limit imposed by a restrictor, further instantiation of the category in new items is prohibited. The Predictor Step eventually halts when it starts creating a new item whose feature specification within the depth allowed by the restrictor is identical to, or subsumed by, a previous one.

In addition to the halting problem, the incorporation of feature-based syntactic categories to grammars poses a new problem unique to the LR parser. After the parser assigns a constituent structure in Fig.4 during parsing, it would consult the GOTO table for the next state with the root category of the constituent, i.e. V[SC [FST NP, RST NIL], TNS PAST]. There is no entry in the table under the root category, however, since the category is distinct from any categories that appear in the items partially intstantiated by the CLOSURE procedure.

The problem stems from the fact that the categories which are partially instantiated by the preliminary CLOSURE procedure and consequently constitute the domain of the GOTO function may be still underspecified as compared with those that arise during parsing. The feature specification [TNS PAST] in the constituent structure in Fig.4, for example, originates from the lexical specification of "gave" in Fig.5, and not from productions, and therefore does not appear in any items in Fig.7. Note that it is possible to create an item with the particular feature instantiated, but there are a potentially infinite number of instantiations for each underspecified category.

Given the preliminary CLOSURE/ NEXT-S procedures, the parser would have to search in the domain of the GOTO function for a category that is unifiable with the root of a constituent in order to obtain the next state, while a search operation is never required by the original LR parsing algorithm. Furthermore, there may be more than one such category in the domain, giving rise to nondeterminism to the algorithm.

4 Construction of the GOTO Table for Feature-Based Categories: A Final Modification

The final version of CLOSURE/NEXT-S procedures in Fig.8 circumvents the described problems. While the CLOSURE procedure makes top-down predictions in the same way as before, new items are added without instantiation. Since only original productions in a grammar appear as items, productions are added as new items only once and the nontermination problem does not occur, as is the case of the LR parsing algorithm with atomic categories. The NEXT-S procedure constructs next states for the lhs category of each production, rather than the categories to the right of a dot. Consequently, from the lhs category of the production used for a reduce action, the parser can uniquely determine the GOTO table entry for a next state, while constructing a constituent structure by instantiating it. No search for unifiable categories is involved during parsing.

```
procedure CLOSURE(I);
begin
    repeat
    for each item <A→w.Bx> in I, and each
    production C→y such that C is unifiable
    with B and <C→.y> is not in I do
        add <C→.y> to I;
    until no more items can be added to I;
    return I
end;

procedure NEXT-S(I,C)
;for each category C on the lhs of productions
begin
    let J be the set of items <A→wB.x> such
    that <A→w.Bx> is in I and B is unifiable
    with C;
    return CLOSURE(J)
end;
```

Figure 8. Final CLOSURE/NEXT-S Procedures

Note, furthermore, the size of GOTO table produced by the final CLOSURE/NEXT-S procedures is usually smaller than the table produced by the preliminary procedures for the same grammar. It is because the preliminary CLOSURE procedure creates one or more instantiations out of a single category, each of which the preliminary NEXT-S procedure applies to, creating separate GOTO table entries. Although a smaller GOTO table does not necessarily imply less parsing time, since there are entry retrieval algorithms that do not depend on a table size, it does mean fewer operations to construct such tables during preprocessing.

5 Further Comparisons and Conclusion

The LR parsing algorithm for grammars with atomic categories involves no category matching during parsing. In Fig.1, categories are pushed onto the stack only for the purpose of constructing a parse tree, and reduce actions are completely independent of categories in the stack. In parsing with feature-based categories, on the other hand, the parser must perform unification operations between the roots of constituents and categories on the rhs of productions during a reduce action. In addition to error entries in the ACTION table, unification failure should result in an error also. Since categories cannot be completely instantiated in every possible way during preprocessing, unification operations during parsing cannot be eliminated.

What motivates partial instantiation of productions during preprocessing as is done by the preliminary CLOSURE procedure, then? It can sometimes prevent wrong items from being predicted and consequently incorrect reduce actions from entering into an ACTION table. Given a grammar that consists of four productions in Fig.9, the final CLOSURE procedure with an item <S→. T[F a]> in an input state will add items <T[F $\boxed{1}$]→. T[F[F $\boxed{1}$]] T[F[F b]]>, <T[F[F a]]→. a> and <T[F[F b]]→. b> to the state. After shift and reduce actions are repeated twice, each to construct the constituent in Fig.10(i), the ACTION table will direct the parser to "reduce by *p2*" to construct T[F $\boxed{1}$b] (Fig.10(ii)), and then to "reduce by *p1*", at which time a unification failure occurs, detecting an error only after all these operations.

p1: S→T[F a]
p2: T[F $\boxed{1}$]→T[F[F $\boxed{1}$]] T[F [F b]]
p3: T[F[F a]]→a
p4: T[F[F b]]→b

Figure 9. Toy Grammar

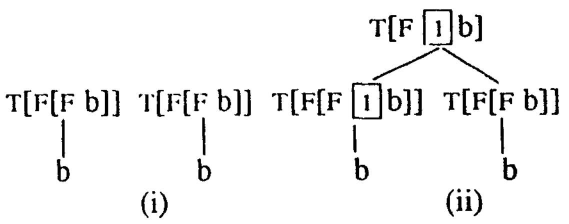

Figure 10. Partial Parse Trees

On the other hand, the preliminary CLOSURE procedure with some restrictor will add partially instantiated items <T[F $\boxed{1}$a]→. T[F[F $\boxed{1}$a]] T[F[F b]]> and <T[F[F a]]→. a>, but not <T[F[F b]]→. b>. From an error entry of the ACTION table, the parser would detect an error as soon as the first input string *b* is shifted.

Given the grammar in Fig.9, the preliminary CLOSURE/NEXT-S procedures outperform the final version. All grammars that solicit this performance difference in error detection have one property in common. That is, in those grammars, some feature specifications in productions which assign upper structures of a parse tree prohibit particular feature instantiations in lower structures. In the case of the above example, the [F a] feature specification in *p1* prohibits the first category on the rhs of *p2* from being instantiated as T[F[F b]]. If the grammar were modified to replace *p1* with p1': S→T, for example, then the preliminary CLOSURE/NEXT-S procedures will have nothing to contribute for early detection of errors, but rather create a larger GOTO/ACTION table through which otherwise unmotivated search must be conducted for unifiable categories to find GOTO table entries after every reduce action. (With a restrictor [CAT][F[F]], the size of ACTION/GOTO table produced by the preliminary procedures is 11(states)x9(categories) with a total of 52 items, while that by the final procedures is 8x7 with 38 items.)

The final output of the parser, whether constructed by the preliminary or the final procedures, is identical and correct. The choice between two approaches depends upon particular grammars and is an empirical question. In general, however, a clear tendency among grammars written in recent linguistic theories is that productions tend to be more general and permissive and lexical specifications more specific and restrictive. That is, information that regulates possible configurations of parse trees for particular input strings comes from the bottom of trees, and not from the top, making top-down instantiation useless.

With the recent linguistic trend of lexicon-oriented grammars, partial instantiation of categories while making predictions top-down gives little to gain for added costs. Given that run-time instantiation of productions is unavoidable to build constituents and to detect errors, the advantages of eliminating an intermediate instantiation step should be evident.

REFERENCES

Aho, Alfred V. and Jeffrey D. Ullman 1987. *Principles of Compiler Design.* Addison-Wesley Publishing Company.

Aho, Alfred V. and S. C. Johnson 1974. "LR Parsing" *Computing Surveys* Vol.6 No.2.

Pollard, Carl and Ivan A. Sag 1987. *Information-Based Syntax and Semantics* Vol.1. CSLI Lecture Notes 13. Stanford: CSLI.

Shieber, S. 1985. "Using Restriction to Extend Parsing Algorithms for Complex-Feature-Based Formalisms" *23rd ACL Proceedings.*

Shieber, S. 1986. *An Introduction to Unification-Based Approaches to Grammar.* CSLI Lecture Notes 4. Stanford: CSLI.

Tomita, Masaru 1986. Efficient Parsing for Natural Language: A Fast Algorithm for Practical Systems. Boston: Kluwer Academic Publishers.