CAPTURING LINGUISTIC GENERALIZATIONS WITH METARULES IN AN ANNOTATED PHRASE-STRUCTURE GRAMMAR

Kurt Konolige
SRI International*

1. Introduction

Computational models employed by current natural language understanding systems rely on phrase-structure representations of syntax. Whether implemented as augmented transition nets, BNF grammars, annotated phrase-structure grammars, or similar methods, a phrase-structure representation makes the parsing problem computationally tractable [7]. However, phrase-structure representations have been open to the criticism that they do not capture linguistic generalizations that are easily expressed in transformational grammars.

This paper describes a formalism for specifying syntactic and semantic generalizations across the rules of a phrase-structure grammar (PSG). The formalism consists of two parts:

1. A declarative description of basic syntactic phrase-structures and their associated semantic translation.
2. A set of metarules for deriving additional grammar rules from the basic set.

Since metarules operate on grammar rules rather than phrase markers, the transformational effect of metarules can be pre-computed before the grammar is used to analyze input. The computational efficiency of a phrase-structure grammar is thus preserved.

Metarule formulations for PSGs have recently received increased attention in the linguistics literature, especially in [4], which greatly influenced the formalism presented in this paper. Our formalism differs significantly from [4] in that the metarules work on a phrase-structure grammar annotated with arbitrary feature sets (Annotated Phrase-structure Grammar, or APSG [7]). Grammars for a large subset of English have been written using this formalism [9], and its computational viability has been demonstrated [6]. Because of the increased structural complexity of APSGs over PSGs without annotations, new techniques for applying metarules to these structures are developed in this paper, and the notion of a match between a metarule and a grammar rule is carefully defined. The formalism has been implemented as a computer program and preliminary tests have been made to establish its validity and effectiveness.

2. Metarules

Metarules are used to capture linguistic generalizations that are not readily expressed in the phrase-structure rules. Consider the two sentences:

1. John gave a book to Mary
2. Mary was given a book by John

Although their syntactic structure is different, these two sentences have many elements in common. In particular, the predicate/argument structure they describe is the same: the gift of a book by John to Mary. Transformational grammars capture this correspondence by transforming the phrase marker for (1) into the phrase marker for (2). The underlying predicate/argument structure remains the same, but the surface realization changes. However, the recognition of transformational grammars is a very difficult computational problem.*

By contrast, metarules operate directly on the rules of a PSG to produce more rules for that grammar. As long as the number of derived rules is finite, the resulting set of rules is still a PSG. Unlike transformational grammars, PSGs have efficient algorithms for parsing [3]. In a sense, all of the work of transformations has been pushed off into a pre-processing phase where new grammar rules are derived. We are not greatly concerned with efficiency in pre-processing, because it only has to be done once.

There are still computational limitations on PSGs that must be taken into account by any metarule system. Large numbers of phrase-structure rules can seriously degrade the performance of a parser, both in terms of its running time**, storage for the rules, and the ambiguity of the resulting parses [6]. Moreover, the generation of large numbers of rules seems psychologically implausible. Thus the two criteria we will use to judge the efficacy of metarules will be: can they adequately capture linguistic generalizations, and are they computationally practicable in terms of the number of rules they generate. The formalism of [4] is especially vulnerable to criticism on the latter point, since it generates large numbers of new rules.***

3. Representation

An annotated phrase-structure grammar (APSG) as developed in [7] is the target representation for the metarules. The core component of an APSG is a set of context-free phrase-structure rules. As is customary, these rules are input to a context-free parser to analyze a string, producing a phrase-structure tree as output. In addition, the parse tree so produced may have arbitrary feature sets, called annotations, appended to each node. The annotations are an efficient means of incorporating additional information into the parse tree. Typically, features will exist for syntactic processing (e.g., number agreement), grammatical function of constituents (e.g., subject, direct and indirect objects), and semantic interpretation.

Associated with each rule of the grammar are procedures for operating on feature sets of the phrase markers the rule constructs. These procedures may constrain the application of the rule by testing features on candidate constituents, or add information to the structure created by the rule, based on the features of its constituents. Rule procedures are written in the programming language LISP, giving the grammar the power to recognize class 0 languages. The use of arbitrary procedures and feature set annotations makes APSGs an

---

*This research was supported by the Defense Advanced Research Projects Agency under Contract N00039-79-C-0118 with the Naval Electronics Systems Command. The views and conclusions contained in this document are those of the author and should not be interpreted as representative of the official policies, either expressed or implied, of the U.S. Government. The author is grateful to Jane Robinson and Gary Hendrix for comments on an earlier draft of this paper.

*There has been some success in restricting the power of transformational grammars sufficiently to allow a recognizer to be built; see [8].

**Sheil [10] has shown that, for a simple recursive descent parsing algorithm, running time is a linear function of the number of rules. For other parsing schemes, the relationship between the number of rules and parsing time is unclear.

***This is without considering infinite schemas such as the one for conjunction reduction. Basically, the problem is that the formalism of [4] allows complex features [2] to define new categories, generating an exponential number of categories (and hence rules) with respect to the number of features.

extremely powerful and compact formalism for representing a language, similar to the earlier ATN formalisms [1]. An example of how an APSG can encode a large subset of English is the DIAGRAM grammar [9].

It is unfortunately the very power of APSGs (and ATNs) that makes it difficult to capture linguistic generalizations within these formalisms. Metarules for transforming one annotated phrase-structure rule into another must not only transform the phrase-structure, but also the procedures that operate on feature sets, in an appropriate way. Because the transformation of procedures is notoriously difficult,* one of the tasks of this paper will be to illustrate a declarative notation describing operations on feature sets that is powerful enough to encode the manipulations of features necessary for the grammar, but is still simple enough for metarules to transform.

4. Notation

Every rule of the APSG has three parts:

1. A phrase-structure rule;
2. A restriction set (RSET) that restricts the applicability of the rule, and
3. An assignment set (ASET) that assigns values to features.

The RSET and ASET manipulate features of the phrase marker analyzed by the rule; they are discussed below in detail. Phrase-structure rules are written as:

$$CAT \rightarrow C_1\ C_2 \cdots C_n$$

where CAT is the dominating category of the phrase, and C_1 through C_n are its immediate constituent categories. Terminal strings can be included in the rule by enclosing them in double quote marks.

A feature set is associated with each node in the parse tree that is created when a string is analyzed by the grammar. Each feature has a name (a string of uppercase alphanumeric characters) and an associated value. The values a feature can take on (the domain of the feature) are, in general, arbitrary. One of the most useful domains is the set {+,-,NIL}, where NIL is the unmarked case; this domain corresponds to the binary features used in [2]. More complicated domains can be used; for example, a CASE feature might have as its domain the set of tuples {<1 SG>,<2 SG>,<3 SG>,<1 PL>,<2 PL>,<3 PL>}. Most interesting are those features whose domain is a phrase marker. Since phrase markers are just data structures that the parser creates, they can be assigned as the value of a feature. This technique is used to pass phrase markers to various parts of the tree to reflect the grammatical and semantic structure of the input; examples will be given in later sections.

We adopt the following conventions in referring to features and their values:

- Features are one-place functions that range over phrase markers constructed by the phrase-structure part of a grammar rule. The function is named by the feature name.
- These functions are represented in prefix form, e.g., (CASE NP) refers to the CASE feature of the NP constituent of a phrase marker. In cases where there is more than one constituent with the same category name, they will be differentiated by a "#" suffix, for example,

  VP -> V NP#1 NP#2

  has two NP constituents.
- A phrase marker is assumed to have its immediate constituents as features under their category name, e.g., (N NP) refers to the N constituent of the NP.
- Feature functions may be nested, e.g., (CASE (N NP)) refers to the CASE feature of the N constituent of the NP phrase marker. For these nestings, we adopt the simpler notation (CASE N NP), which is assumed to be right-associative.
- The value NIL always implies the unmarked case. At times it will be useful to consider features that are not explicitly attached to a phrase marker as being present with value NIL.
- A constant term will be written with a preceding single quote mark, e.g., 'SG refers to the constant token SG.

4.1. Restrictions

The RSET of a rule restricts the applicability of the rule by a predication on the features of its constituents. The phrase markers used as constituents must satisfy the predications in the RSET before they will be analyzed by the rule to create a new phrase marker. The most useful predicate is equality: a feature can take on only one particular value to be acceptable. For example, in the phrase structure rule:

S -> NP VP

number agreement could be enforced by the predication:

(NBR NP) = (NBR VP)

where NBR is a feature whose domain is {SG,PL}.* This would restrict the NBR feature on NP to agree with that on VP before the S phrase was constructed. The economy of the APSG encoding is seen here: only a single phrase-structure rule is required. Also, the linguistic requirement that subjects and their verbs agree in number is enforced by a single statement, rather than being implicit in separate phrase structure rules, one for singular subject-verb combinations, another for plurals.

Besides equality, there are only three additional predications: inequality ($\neq$), set membership ($\in$) and set non-membership ($\notin$). The last two are useful in dealing with non-binary domains. As discussed in the next section, tight restrictions on predications are necessary if metarules are to be successful in transforming grammar rules. Whether these four predicates are adequate in descriptive power for the grammar we contemplate remains an open empirical question; we are currently accumulating evidence for their sufficiency by rewriting DIAGRAM using just those predicates.

Restriction predications for a rule are collected in the RSET of that rule. All restrictions must hold for the rule to be applicable. As an illustration, consider the subcategorization rule for ditransitive verbs with prepositional objects (e.g., "John gave a book to Mary"):

```
VP -> V NP PP
RSET: (TRANS V) = 'DI;
      (PREP V)  = (PREP PP)
```

The first restriction selects only verbs that are marked as ditransitive; the TRANS feature comes from the lexical entry of the verb. Ditransitive verbs with prepositional arguments are always subcategorized by the particular preposition used, e.g., "give" always uses "to" for its prepositional argument.

---

*It is sometimes hard to even understand what it is that a procedure does, since it may involve recursion, side-effects, and other complications.

*How NP and VP categories could "inherit" the NBR feature from their N and V constituents is discussed in the next section.

The second predication restricts the preposition of the PP for a given verb. The PREP feature of the verb comes from its lexical entry, and must match the preposition of the PP phrase*

4.2. Assignments

A rule will normally assign features to the dominating node of the phrase marker it constructs, based on the values of the constituents' features. For example, feature inheritance takes place in this way. Assume there is a feature NBR marking the syntactic number of nouns. Then the ASET of a rule for noun phrases might be:

```
NP -> DET N
ASET:  (NBR NP) := (NBR N)
```

This notation is somewhat non-standard; it says that the value of the NBR function on the NP phrase marker is to be the value of the NBR function of the N phrase marker.

An interesting application of feature assignment is to describe the grammatical functions of noun phrases within a clause. Recall that the domain of features can be constituents themselves. Adding an ASET describing the grammatical function of its constituents to the ditransitive VP rule yields the following:

```
VP -> V NP PP
ASET:  (DIROBJ VP) := (NP VP);
       (INDOBJ VP) := (NP PP).
```

This ASET assigns the DIROBJ (direct object) feature of VP the value of the constituent NP. Similarly, the value of INDOBJ (indirect object) is the NP constituent of the PP phrase.

A rule may also assign feature values to the constituents of the phrase marker it constructs. Such assignments are context sensitive, because the values are based on the context in which the constituent occurs.** Again, the most interesting use of this technique is in assigning functional roles to constituents in particular phrases. Consider a rule for main clauses:

```
S -> NP VP
ASET:  (SUBJ VP) := (NP S).
```

The three features SUBJ, DIROBJ, and INDOBJ of the VP phrase marker will have as value the appropriate NP phrase markers, since the DIROBJ and INDOBJ features will be assigned to the VP phrase marker when it is constructed. Thus the grammatical function of the NPs has been identified by assigning features appropriately.

Finally, note that the grammatical functions were assigned to the VP phrase marker. By assembling all of the arguments at this level, it is possible to account for bounded deletion phenomenon that are lexically controlled. Consider subcategorization for Equi verbs, in which the subject of the main clause has been deleted from the infinitive complement ("John wants to go"):

```
VP -> V INF
ASET:  (SUBJ INF) := (SUBJ VP)
```

Here the subject NP of the main clause has been passed down to the VP (by the S rule), which in turn passes it to the infinitive as its subject. Not all linguistic phenomenon can be formulated so easily with APSGs; in particular, APSGs have trouble describing unbounded deletion and conjunction reduction. Metarule formulations for the latter phenomena have been proposed in [5], and we will not deal with them here.

5. Metarules for APSGs

Metarules consist of two parts: a match template with variables whose purpose is to match existing grammar rules; and an instantiation template that produces a new grammar rule by using the match template's variable bindings after a successful match. Initially, a basic set of grammar rules is input; metarules derive new rules, which then can recursively be used as input to the metarules. When (if) the process halts, the new set of rules, together with the basic rules, comprises the grammar.

We will use the following notation for metarules:

```
MF => IF

CSET:  C1, C2, ... Cn
```

where MF is a matching form, IF is an instantiation form, and CSET is a set of predications. Both the MF and IF have the same form as grammar rules, but in addition, they can contain variables. When an MF is matched against a grammar rule, these variables are bound to different parts of the rule if the match succeeds. The IF is instantiated with these bindings to produce a new rule. To restrict the application of metarules, additional conditions on the variable bindings may be specified (CSET); these have the same form as the RSET of grammar rules, but they can mention the variables matched by the MF.

Metarules may be classified into three types:

1. Introductory metarules, where the MF is empty (=> IF). These metarules introduce a class of grammar rules.
2. Deletion metarules, where the IF is empty (MF =>). These delete any derived grammar rules that they match.
3. Derivation metarules, where both MF and IF are present. These derive new grammar rules from old ones.

There are linguistic generalizations that can be captured most perspicuously by each of the three forms. We will focus on derivation metarules here, since they are the most complicated.

6. Matching

An important part of the derivation process is the definition of a match between a metarule matching form and a grammar rule. The matching problem is complicated by the presence of RSET and ASET predications in the grammar rules. Thus, it is helpful to define a match in terms of the phrase markers that will be admitted by the grammar rule and the MF. We will say that an MF matches a grammar rule just in case it admits at least those phrase markers admitted by the grammar rule. This definition of a match is sufficient to allow the formulation of matching algorithms for grammar rules complicated by annotations.

We divide the matching process into two parts: matching phrase-structures, and matching feature sets. Both parts must succeed in order for the match to succeed.

---

*Note that we are not considering here prepositional phrases that are essentially meta-arguments to the verb, dealing with time, place, and the like. The prepositions used for meta-arguments are much more variable, and usually depend on semantic considerations.

**The assignment of features to constituents presents some computational problems, since a context-free parser will no longer be sufficient to analyze strings. This was recognized in the original version of APSGs [7], and a two-pass parser was constructed that first uses the context-free component of the grammar to produce an initial parse tree, then adds the assignment of features in context.

6.1. Matching Phrase-structures

For phrase-structures, the definition of a match can be replaced by a direct comparison of the phrase-structures of the MF and grammar rule. Variables in the MF phrase-structure are used to indicate "don't care" parts of the grammar rule phrase-structure, while constants must match exactly. Single lower case letters are used for variables that must match single categories of the grammar rule. A typical MF might be:

```
S ->.a VP
```

which matches

```
S -> NP VP      with a=NP;
S -> SB VP      with a=SB;
S -> "IT" VP    with a="IT";
etc.
```

A variable that appears more than once in an MF must have the same binding for each occurrence for a match to be successful, e.g.,

```
VP -> V a a
```

matches

```
VP -> V NP NP  with a=NP
```

but not

```
VP -> V NP PP
```

Single letter variables must match a single category in a grammar rule. Double letter variables are used to match a number of consecutive categories (including none) in the rule. We have:

```
VP -> V uu
```

matching

```
VP -> V          with uu=();
VP -> V NP       with uu=(NP);
VP -> V NP PP    with uu=(NP PP);
etc.
```

Note that double letter variables are bound to an ordered list of elements from the matched rule. Because of this characteristic, an MF with more than one double letter variable may match a rule in several different ways:

```
VP -> V uu vv
```

matches

```
VP -> V NP PP  with   uu=(), vv=(NP PP);
                      uu=(NP), vv=(PP);
                      uu=(NP VP), vv=().
```

All of these are considered to be valid, independent matches. Double and single letter variables may be intermixed freely in an MF.

While double letter variables match multiple categories in a phrase structure rule, string variables match parts of a category. String variables occur in both double and single letter varieties; as expected, the former match any number of consecutive characters, while the latter match single characters. String variables are assumed when an MF category contains a mixture of upper and lower case characters, e.g.:

```
Vi -> V NP#a NPuu
```

matches

```
VP -> V NP#1 NP       with a=1, uu=();
VP -> V NP#1 NP#2     with a=1, uu=(# 2);
etc.
```

String variables are most useful for matching category names that may use the # convention.

6.2. Feature Matching

So far variables have matched only the phrase-structure part of grammar rules, and not the feature annotations. For feature matching, we must return to the original definition of matching based on the admissibility of phrase markers. The RSET of a grammar rule is a closed formula involving the feature sets of the phrase marker constructed by the rule; let P stand for this formula. If P is true for a given phrase marker, then that phrase marker is accepted by the rule; if not, it is rejected. Similarly, the RSET of a matching form is an <u>open</u> formula on the feature sets of the phrase marker; let $R(x_1,x_2,\cdots x_n)$ stand for this formula, where the x_i are the variables of the RSET. For the MF's restrictions to match those of the grammar rule, we must be able to prove the formula:

$$P \Rightarrow (Ex_1)(Ex_2)\cdots(Ex_n)\ R(x_1,x_2,\cdots x_n)$$

That is, whenever P admits a phrase marker, there exists some binding for R's free variables that also admits the phrase marker.

Now the importance of restricting the form of P and R can be seen. Proving that the above implication holds for general P and R can be a hard problem, requiring, for example, a resolution theorem prover. By restricting P and R to simple conjunctions of equalities, inequalities, and set membership predicates, the match between P and R can be performed by a simple and efficient algorithm.

6.3. Instantiation

When a metarule matches a grammar rule, the CSET of the metarule is evaluated to see if the metarule can indeed be applied. For example, the MF:

```
VP -> "BE" xP
CSET:  x ≠ 'V
```

will match any rule for which <u>x</u> is not bound to V.

When an MF matches a rule, and the CSET is satisfied, the instantiation form of the metarule is used to produce a new rule. The variables of the IF are instantiated with their values from the match, producing a new rule. In addition, restriction and assignment features that do not conflict with the IF's features are carried over from the rule that matched. This latter is a very handy property of the instantiation, since that is usually what the metarule writer desires. Consider a metarule that derives the subject-aux inverted form of a main clause with a finite verb phrase:

```
grammar rule:  S -> NP AUX VP
       RSET:   (NBR NP) = (NBR AUX);
               (FIN VP) = '+;

metarule:      S -> NP AUX VP

       SAI -> AUX NP VP
```

If features were not carried over during an instantiation, the result of matching and instantiating the metarule would be:

```
SAI -> AUX NP VP
```

This does not preserve number agreement, nor does it restrict the VP to being finite. Of course, the metarule could be rewritten to have the correct restrictions in the IF, but this would sharply curb the utility of the metarules, and lead to the proliferation of metarules with slightly different RSETs.

7. An Example: Dative Movement and Passive

We are now ready to give a short example of two metarules for dative movement and passive transformations. The predicate/argument structure will be described by the feature PA, whose value is a list:

(V NP_1 NP_2 ...)

where V is the predicating verb, and the NPs are its arguments. The order of the arguments is significant, since:

("gave" "John" "a book" "Mary")
<=> gift of a book by John to Mary

("gave" "John" "Mary" "a book")
<=> ?? gift of Mary to a book by John

Adding the PA feature, the rule for ditransitive verbs with prepositional objects becomes:

```
VP -> V NP PP
RSET: (TRANS V) = 'DI;
      (PREP V)  = (PREP PP);
ASET: (PA VP) := '((V VP) (SUBJ VP)(NP VP)(NP PP))
```

The SUBJ feature is the subject NP passed down by the S rule.

7.1. Dative Movement

In dative movement, the prepositional NP becomes a noun phrase next to the verb:

1. John gave a book to Mary =>
2. John gave Mary a book

The first object NP of (2) fills the same argument role as the prepositional NP of (1). Thus the dative movement metarule can be formulated as follows:

metarule DATMOVE

```
      VP -> V uu PP
      ASET: (PA VP) := '(a b c (NP PP))

=>    VP -> V NP#D uu
      RSET: (DATIVE V) = '+;
            (PREP V)   = NIL;
      ASET: (PA VP)    := '(a b c (NP#D VP))
```

DATMOVE accepts VPs with a trailing prepositional argument, and moves the NP from that argument to just after the verb. The verb must be marked as accepting dative arguments, hence the DATIVE feature restriction in the RSET of the instantiation form. Also, since there is no longer a prepositional argument, the PREP feature of the VP doesn't have to match it. As for the predicate/argument structure, the NP#D constituent takes the place of the prepositional NP in the PA feature.

DATMOVE can be applied to the ditransitive VP rule to yield the ditransitive dative construction. The variable bindings are:

```
uu = (NP);
a  = (V VP)
b  = (SUBJ VP);
c  = (NP VP).
```

Instantiating the IF then gives the dative construction:

```
VP -> V NP#D NP
RSET: (DATMOVE V) = '+;
      (TRANS V)   = 'DI;
ASET: (PA VP) :=
      '((V VP) (SUBJ VP) (NP VP) (NP#D VP))
```

There are other grammar rules that dative movement will apply to, for example, verbs with separable particles:

Make up a story for me => Make me up a story.

This is the reason the double-letter variable "uu" was used in DATMOVE. As long as the final constituent of a VP rule is a PP, DATMOVE can apply to yield a dative construction.

7.2. Passive

In the passive transformation, the NP immediately following the verb is moved to subject position; the original subject moves to an agentive BY-phrase:

(1) John gave a book to Mary =>
(2) A book was given to Mary by John.

A metarule for the passive transformation is:

metarule PASSIVE

```
      VP -> V NPuu vv
      ASET: (PA VP) := '(a (SUBJ VP) bb (NPuu VP) cc);

=>    AP -> V PPL vv PP#A
      RSET: (PREP PP#A) = 'BY;
      ASET: (PA VP)  := '(a (NP PP#A) bb (SUBJ VP) cc).
```

PASSIVE deletes the NP immediately following the verb, and adds a BY-prepositional phrase at the end. PPL is a past participle suffix for the verb. In the predicate/argument structure, the BY-phrase NP substitutes for the original subject, while the new subject is used in place of the original object NP. Applying PASSIVE to the ditransitive rule yields:

```
AP -> V PPL PP PP#A
RSET: (TRANS V) = 'DI;
      (PREP V)  = (PREP PP);
ASET: (PA VP) :=
        '((V VP) (NP PP#A) (SUBJ VP) (NP PP));
```

e.g., "A book was given to Mary by John" will be analyzed by this rule to have a PA feature of ("give" "John" "a book" "Mary"), which is the same predicate/argument structure as the corresponding active sentence.

PASSIVE can also apply to the rule generated by DATMOVE to yield the passive form of VP's with dative objects:

```
AP -> V PPL NP PP#A
RSET: (DATMOVE V) = '+;
      (TRANS V)   = 'DI;
ASET: (PA VP) :=
        '((V VP) (NP PP#A) (NP VP) (SUBJ VP));
```

e.g., "Mary was given a book by John".

8. Implementation

A system has been designed and implemented to test the validity of this approach. It consists of a matcher/instantiator for metarules, along with an iteration loop that applies all the metarules on each cycle until no more new rules are generated. Metarules for verb subcategorization and finite and non-finite clause structures have been written and input to the system. We were especially concerned:

- To check the perspicuity of metarules for describing significant fragments of English using the above representation for grammar rules.
- To check that a reasonably small number of new grammar rules were generated by the metarules for these fragments.

Both of these considerations are critical for the performance of natural language processing systems. Preliminary tests

indicate that the system satisfies both these concerns; indeed, the metarules worked so well that they exposed gaps in a phrase-structure grammar that was painstakingly developed over a five year period and was thought to be reasonably complete for a large subset of English [9]. The number of derived rules generated was encouragingly small:

Subcategorization:
1 grammar rule
7 metarules => 20 derived rules

Clauses:
8 grammar rules
5 metarules => 25 derived rules

9. Conclusions

Metarules, when adapted to work on an APSG representation, are a very powerful tool for specifying generalizations in the grammar. A great deal of care must be exercised in writing metarules, because it is easy to state generalizations that do not actually hold. Also, the output of metarules can be used again as input to the metarules, and this often produces surprising results. Of course, language is complex, and it is to be expected that describing its generalizations will also be a difficult task.

The success of the metarule formulation in deriving a small number of new rules comes in part from the increased definitional power of APSGs over ordinary PSGs. For example, number agreement and feature inheritance can be expressed simply by appropriate annotations in an APSG, but require metarules on PSGs. The definitional compactness of APSGs means that fewer metarules are needed, and hence fewer derived rules are generated.

REFERENCES

1. W. Woods, "An Experimental Parsing System for Transition Network Grammars," R. Rustin (ed.), Natural Language Processing, Prentice-Hall, Englewood Cliffs, New Jersey, 1973.

2. N. Chomsky, Aspects of the Theory of Syntax, MIT Press, Cambridge, Mass., 1965.

3. J. Early, "An Efficient Context Free Parsing Algorithm," CACM, Vol. 13 (1970) 94-102.

4. Gerald Gazdar, "English as a Context-Free Language" University of Sussex, (unpublished paper, April, 1979).

5. Gerald Gazdar, "Unbounded Dependencies and Coordinate Structure" University of Sussex, (submitted to Linguistic Inquiry, October, 1979).

6. Kurt Konolige, "A Framework for a Portable NL Interface to Large Data Bases," Technical Note 197, Artificial Intelligence Center, SRI International, Menlo Park, California (October 1979).

7. William H. Paxton, "A Framework for Speech Understanding," Technical Note 142, Artificial Intelligence Center, SRI International, Menlo Park, California (June 1977).

8. S. R. Petrick, "Automatic Syntactic and Semantic Analysis," Proceedings of the Interdisciplinary Conference on Automated Text Processing, (November 1976).

9. Jane Robinson, "DIAGRAM: A Grammar for Dialogues," Technical Note 205, Artificial Intelligence Center, SRI International, Menlo Park, California (February 1980).

10. B. A. Sheil, "Observations on Context-Free Parsing," Statistical Methods in Linguistics, (1976).