An Overview of the Project NASEV Parser.

William R. Keller.

July 1987.

Cognitive Studies Research Paper

Serial no: CSRP 86

The University of Sussex
Cognitive Studies Programme
School of Social Sciences
Falmer
Brighton BN1 9QN

An Overview of the Project NASEV Parser.

William R. Keller.

University of Sussex, Cognitive Studies Programme.
Falmer, Brighton BN1 9QN. England.
July 1987.

ABSTRACT

The KIT/NASEV chart-parsing system for GPSG-like grammars is described. The parser is based on a version of Earley's algorithm and accepts rules in IP/LP format with category-labels represented by complex feature structures (complex symbols). Particular attention is paid to the problem of parsing with complex symbols and the need to perform LP checks 'on the fly'.

The parser also incorporates a simple version of the *functional realization principle* of GPSG, augmented with a quantifier storge mechanism. By this means, logical translations of NL input may be built in parallel with syntactic analysis.

An Overview of the Project NASEV Parser.

William R. Keller.

University of Sussex, Cognitive Studies Programme,
Falmer, Brighton BN1 9QN, England.

July **1987.**

1. Introduction.

The KIT/NASEV **[1]** chart parser is an experimental implementation of GPSG under development at the Technische Universitaet Berlin. A primary design goal is the provision of an interpreter for the GPSG formalism as set out in [Gazdar *et al* 1985] (henceforth GKPS). Furthermore the implementation should be efficient, as the parser is to form one component of a practical machine-translation system additionally involving transfer and generation modules.

This report covers work completed on the implementation at the end of September 1986. As described here, the parser accepts rules in *immediate dominance/linear precedence* (ID/LP) format, with category labels represented by complex feature structures. Feature passing is achieved explicitly through the unification of feature values within individual ID rules. In this way something of the complex interactions of the *feature instantiation principles* (FIPs) may be mimicked in order to deal with NL dependency and agreement phenomena.

The parser incorporates a simple implementation of the *functional realisation principle* (FRP) of GKPS permitting the translation of NL input into expressions of Montague's *intensional logic* (IL). In order to cope with quantifier scope ambiguities,

1. Project KIT/NASEV (Kuenstliche Intelligenz und Textverstehen/Neue Analyse- und Syntheseverfahren zur maschinellen Uebersetzung — Artificial Intelligence and Text-Comprehension/New Approaches to Analysis and Generation for Machine Translation) is the Technical University of Berlin component of the Eurotra-D research project sponsored by the Federal Ministry of Research and Technology, West Germany.

Thanks are due to Roger Evans and Gerald Gazdar for their comments on an earlier version of this report.

the FRP has been augmented with a novel version of Robin Cooper's storage mechanism. The complete system also includes output formatting routines and trace facilities which are under the control of the user.

Work in progress includes the implementation of *feature cooccurrence restrictions* (FCRs), the FIPs and the addition of a linguistically-oriented editor to facilitate grammar writing and debugging. A large-scale grammar of German is also currently under development.

2. The Grammar Formalism.

The NASEV parser makes use of a grammar consisting of three basic components:

- ID Rules.
- LP Rules.
- A Lexicon.

Additional information is necessary if the parser is to provide semantic as well as syntactic analyses of its input (see section 4).

2.1. Grammar Format.

ID rules are represented by Prolog terms of the following general form:

```
id(<Number>, <Mother>, <Daughters>).
```

where <Number> is a unique number associated with the ID rule, and <Mother> is the mother symbol. Categories on the right-hand side of the rule are represented by <Daughters> which is itself a Prolog term:

```
r(<Obligatory>, <Optional>, <Star>).
```

Each of <Obligatory>, <Optional> and <Star> is a (possibly empty) list of symbols. As their names suggest, the lists represent those sets of categories which are obligatory, optional and closed under the Kleene-star operation respectively.

(Note that the order in which symbols appear in these lists is of no significance to the parser.)

By way of illustration, the ID rule in (1a) would be represented internally as (1b).

```
(1) a.   16 : vp --> v, (pp_to), vp_inf
    b.   id(16, vp, r([v, vp_inf], [pp_to], [])).
```

It will prove useful subsequently to understand an ID rule such a (1a) as syntactic sugar for the internal form rule shown in (1b).

LP rules are represented internally by Prolog terms having the general form:

lp(<Category1>, <Category2>).

The GPSG LP rule in (2a) is encoded for the parser simply as (2b). As for ID rules, (2a) may be conveniently regarded as syntactic sugar for (2b).

```
(2) a.   np < vp
    b.   lp(np, vp)
```

The lexicon consists of various entries, each of the form:

lex(<Word>, <Category>, <Mexp>).

where <Word> is a Prolog atom representing a word form (e.g. *wichtig*, *verabschiedet*, *auf*, etc), <Category> represents the lexical category for <Word>, and <Mexp> represents an appropriate translation of <Word> into intensional logic (see section 4.1).

2.2. Complex Categories.

In GPSG category labels are not monadic symbols, but sets or bundles of feature specifications. The use of feature sets or structures as category labels, along with the *feature instantiation principles*, accounts for much of the success of GPSG in providing elegant analyses of complex syntactic phenomena. In order to 'talk about' categories in this sense, the GPSG formalism permits rules to be stated in terms of *complex symbols*, or partial representations of feature sets. Similarly, the parser accepts rules and lexical entries which are stated in terms of partially specified *category-descriptors*. A category-descriptor is a Prolog term of the following form:

```
cat(<vl>, <v2>, ..., <vn>).
```

Each of the <vi> is a feature value (possibly unspecified).

From grammar to grammar, the arity of the functor CAT may vary, but for a particular grammar it will be fixed and equal to the number of distinct feature names from which categories may be built. Each argument position encodes a unique feature name. For example, consider the following features and values for a very small GPSG.

(3)	1.	BAR	{0.1.2}
	2.	N	{+.-}
	3.	V	{+.-}
	4.	NUM	{sing.plur}
	5.	PER	{1.2.3}
	6.	CASE	{nom,acc}

By numbering the features as shown in (3) then a complex symbol such as [BAR 1, N-, V+, "CASE] (a description of VP) may be encoded as (4).

(4) cat(l,-,+,X.Y,~).

Note that if a particular feature value is not explicitly specified, then the corresponding argument is left as a Prolog variable (e.g. the values of NUM and PER are represented by the variables X and Y respectively in the above). Undefined, as opposed to unspecified features are represented using a special feature value '~\ so that CASE is undefined in (4).

Category-descriptors, just like the complex symbols of GPSG are *partial representations* of feature sets. Each may be understood as denoting, or *licencing* a whole set of categories. For example, (4) denotes a set of 12 different categories, bearing various combinations of feature values for NUM and PER (note that this includes the possibility of the 'undefined' value '~'). In contrast, the category-descriptor in (5) licences just a single category — it actually uniquely determines one of the 12 categories licensed by (4) (third person singular VP).

(5) cat(1,-,+,sing,3,~)

Feature passing is achieved by unifying feature values across category-descriptors within individual rules. That is, the same Prolog variable is used to stand for the value of separate attributes. Using the feature system of (3), the following ID rule might be used in a toy grammar of English to achieve number and person agreement between subject and predicate.

1 : cat(2,-,+,X,Y,~) --> cat(1,+,-,X,Y,nom), cat(1,-,+,X,Y,~)

3. The Parser.

3.1. Earley's Algorithm and ID/LP Grammar.

It is well known that the problem of context-free recognition is not difficult, indeed practical algorithms exist which perform in time proportional to the cube of the length of the input string [Earley 1970], [Kasami 1965], [Tomita 1985]. Since it is furthermore clear that the weak generative capacity of ID/LP grammars (and indeed GPSGs) is identical to that of standard CFGs, this suggests that with suitable modifications these algorithms may also be used for effective recognition with grammars in ID/LP format [2]. In this respect Earley's algorithm has been particularly attractive on account of its generality and good practical efficiency.

A modification of Earley's algorithm for direct parsing of ID/LP grammar was

2. However, this does not in itself imply that recognition of either ID/LP grammars or GPSGs is of the same order of complexity as recognition of CFGs, *so long as no prior compilation step is permitted*. It has been shown by Barton that the problem of ID/LP recognition is NP-hard, whilst Ristad has located further sources of intractability in the GPSG formalism indicating that the recognition problem is EXP-POLY-hard [Barton *et al* 1987].

first proposed by Shieber [Shieber 1984]. The Shieber predictor, like that of Earley, works top-down and may lead to the introduction of superfluous items. It was suggested by Kilbury [Kilbury 1984a 1984b] that by altering the basic parse-strategy, the introduction of certain items could be avoided and the practical efficiency of ID/LP parsing enhanced.

Kilbury's proposed alteration involves a delayed predictor, now driven bottom-up rather than top-down. A so-called **FIRST** relation is used to guide the predictor by locating just those ID rules which may introduce a given constituent at a particular point in an analysis. **FIRST** is a set of pairs, each of the form **<C,n>**, where **C** is a category and **n** a rule number. For **<A,k>** $\in$ **FIRST**, category **A** may appear as first daughter in some local tree projected from ID rule **k**.

Although Kilbury's technique avoids the need to store certain items, the bottom-up property of the algorithm has its own disadvantages. Other superfluous items may be introduced during prediction, and in some cases the algorithm's performance is actually worse than that of Earley's. A consideration of Kilbury's parse-strategy subsequently lead to a further development of the algorithm by Doerre and Momma [Doerre & Momma 1985].

The Doerre-Momma parse-strategy overcomes the disadvantages of Kilbury's approach without giving up any of the advantages. The algorithm makes use of a delayed predictor, but retains the top-down property of Earley's original strategy and Shieber's adaptation. The predictor is driven by a new relation **FIRST+**, which is basically the transitive closure of Kilbury's **FIRST** relation. **FIRST+** is a set of triples, each of the form **<C1,C2,n>**, where **C1** and **C2** are categories and **n** is a rule number. For **<A,B,k>** $\in$ **FIRST+**, then category **B** may appear as first symbol in a string of symbols dominated by category-label **A**, if **B** is introduced by ID rule **k**.

In terms of practical efficiency, it appears that the Doerre-Momma parse strategy is superior to both Shieber's algorithm and that of Kilbury. For this reason it was chosen as the basis for the NASEV chart parser.

3.2. Parser Implementation.

The NASEV parser is implemented in a core subset of Prolog [Clocksin and Mellish 1981]. and has been run successfully with only minor modifications under Dec 10 Prolog, MProlog, and Waterloo Prolog 1.5.

The implementation makes use of an active chart which is maintained as a set of edges asserted in the Prolog database whilst the program is running. Each edge has the form:

```
item(<Start>,<Finish>,<Mother>,<Daughters>,<Trees>,<Repns>).
```

where <Start> and <Finish> are integers representing the starting and finishing points of the edge in the input string, <Mother> represents the mother category of the corresponding ID rule, and <Daughters> is a record of the daughter categories which remain to be recognised. In addition, <Trees> is a list of parse trees, in order from the left, of the constituents which have already been recognised between <Start> and <Finish>, and <Repns> is a corresponding list of semantic representations, such that the *i*'th member of <Repns> is the semantic representation of the *i*'th parse tree in <Trees>.

New edges are introduced at two separate points in the algorithm: COMPLETE and PREDICT. Once introduced the new edge is passed directly to a predicate CLOSURE which calls COMPLETE and PREDICT recursively. On termination of CLOSURE, the current edge is guaranteed to have been processed as far as possible (i.e. it never needs to be passed to CLOSURE again).

It turns out that not all edges introduced during the COMPLETE and PREDICT phases of the algorithm need to be entered in the chart. During COMPLETE only active edges are ever retrieved from the chart (in an attempt to extend them). At the PREDICT stage, on the other hand, the chart is checked to establish which symbols are expected at the current point in the parse. Again this involves only the inspection of active edges. In consequence, completely inactive edges passed to CLOSURE never need to be recorded [3]. Omitting inactive edges form the chart has two positive effects:

- The parser uses up less space than it otherwise would whilst running.
- The time taken to search the chart for an edge is reduced.

Since the Doerre-Momma algorithm is essentially top-down, some initialisation of the chart is required. In the present implementation explicit initialisation is avoided by adopting a bottom-up strategy at the outset of a parse. Working bottom-up, the first few symbols of an input string effectively provide enough information to quickly converge on a set of possible categories for the string as a whole. Once established, the parser may proceed top-down for the remainder of the input. The parse strategy is thus rather more flexible than a direct implementation of the Doerre-Momma algorithm, since it permits the analysis of strings derived from *any* non-terminal symbol of the grammar (i.e. not just some designated start symbol).

3.3. Linear Precedence and Complex Symbols.

In GPSG LP rules are interpreted as well-formedness conditions on local trees (i.e. trees of depth equal to one). An LP rule, C1 < C2, may be understood as the conditional statement:

> If C1' and C2' occur as sisters in a local tree, then C1' must precede C2' (where C1' and C2' extend C1 and C2 respectively).

Although this is a fairly straightforward idea, the correct implementation of LP checks during parsing is rather less obvious. Consider the problem of deciding *when* to carry out LP checks. In the Doerre-Momma algorithm, LP checking is done 'on

3. An edge is here described as *completely inactive* if it has no constituents (obligatory or otherwise) still to be recognised. This implies that edges with remaining optional (or starred) constituents must always be entered into the chart. It may also be noted in passing that since it is useful (though by no means necessary) to record the final results of a parse in the chart, inactive edges which span the entire input string are asserted in the Prolog database.

the fly' during both COMPLETE and PREDICT. However, the algorithm does not address the problems arising from the use of complex symbols in grammar rules [4].

In general it is not possible to correctly verify LP ordering requirements with respect to a string of *partial* representations of categories, since feature instantiation may subsequently lead to LP violations. The conclusion must be that LP checking cannot be carried out effectively during either COMPLETE or PREDICT, as category-descriptors are typically under-specified at this stage (i.e. they provide only partial information about categories).

Rather surprisingly, it appears that there is *no* point during a parse at which a string of fully instantiated descriptors, representing the daughters of a node, is guaranteed to be available. It follows *a fortiori*, there is no point at which LP ordering requirements may be reliably checked.

Consider the way in which features become instantiated during a parse. In general, category-descriptors are *unified,* causing feature information to flow both 'up' and 'down* the parse structures being built. For a parser, a pathological case can arise when feature information associated with a constituent at one point in a string crucially effects the required ordering of constituents at another. This state of affairs is illustrated by the following simple grammar.

4. Note that the comments in this section also apply to extensions to complex symbols of the algorithms presented by Shieber and Kilbury.

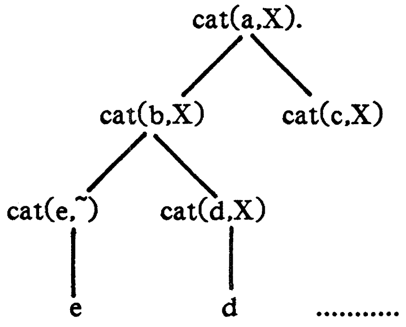
```
1.  CAT      {a,b,c,d,e}
2.  INV      {+,-}

1 : cat(a,X) --> cat(b,X), cat(c,X).
2 : cat(b,X) --> cat(d,X), cat(e,~).

cat(c,+) ->- c_plus.
cat(c,-) ->- c_minus.
cat(d,_) ->- d.
cat(e,_) ->- e.

cat(_,~) < cat(_,+).
cat(_,-) < cat(_,~).
```

Suppose that a left-to-right parser is analysing a string with the prefix *ed* according to the above grammar. After seeing only the prefix, the analysis will have progressed to the stage represented below.

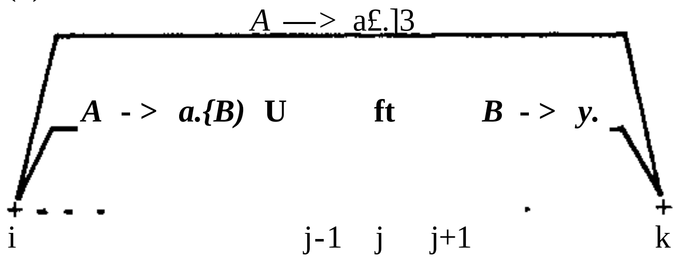


Note that if the string continues with the terminal symbol *c_minus*, then feature INV will take on the value '-' wherever it is currently unspecified (i.e. the variable **X**). As a result, the second LP rule will be violated because **cat(e,~)** and **cat(d,-)** occur as sisters, yet in the wrong order. Until the third input symbol is seen and the variable **X** becomes bound through unification, the LP rules have nothing concrete to say about the linear ordering of **cat(e,~)** and **cat(d,X)**. This is because the latter category-descriptor is not an extension of **cat(_,~)**, **cat(_,+)** or **cat(_,-)** so that the two LP rules simply do not apply.

It is quite possible to devise grammars using the formalism of section 2.1 which involve *unbounded* dependencies of this sort. It follows that even if the parser is capable of looking ahead some fixed number of input symbols then this in the general case LP-checks cannot be reliably performed until *all* of the input has been seen, a state of affairs that is hardly satisfactory. Delaying LP checks until the last moment will mean that the parser wastefully pursues many 'dead-ends' in attempting to analyse its input. Fortunately, it turns out that a minor restriction on the form of the grammars accepted by the parser is sufficient to guarantee that LP checks may be carried out reliably during COMPLETE

First let us note that during COMPLETE an inactive edge, representing a recognised constituent *B,* may be used to extend an active edge. The general situation is pictured in (6), where the active edge from *i* to *j* is extended by an inactive edge from *j* to *k.* Note that a is a (possibly null) string of category descriptors, 0 a (possibly empty) set of category-descriptors, and *y* is either a string of category-descriptors or a terminal symbol.

(6)



Now clearly, it is necessary to ensure that the new string of category-descriptors *ocB* in the edge constructed from *i* to *k* is consistent with the LP rules of the grammar. That is, assuming a to be well-ordered, it must be verified that *B* can indeed follow each category-descriptor in the string a. If this can be ensured each time an edge is extended, then each inactive edge produced by the parser will represent a well-formed (i.e. well-ordered) constituent.

For the reasons outlined above, in order to perform LP checks with respect to a string *otB,* each of the category-descriptors in the string must be fully instantiated. Thus the problem of LP checking during completion effectively reduces to the problem of ensuring that each inactive edge represents a unique constituent.

There are two cases to consider. Firstly, suppose that *B* represents a preterminal category, and *y* a terminal symbol. In this case, to ensure that *B* is a fully instantiated category-descriptor just amounts to putting a restriction on the the form of lexical entries. This is quite simply that all preterminal category-descriptors associated with terminals in the lexicon be fully instantiated. Intuitively, the restriction means that lexical ambiguity should alway be dealt with explicitly, by spelling out all of the various possibilities. Ambiguity should not be 'fudged' by making lexical entries suitably vague.

In the second case *B* represents a non-terminal category, and *y* «= yly2 ... yn a string categories. The inactive edge corresponds to some ID rule of the grammar *B'* —> *{yV*, ..., yn'}, where *B* extends *B'* and each yi extends yi\ 1 < i < n.

Now . we might reasonably expect that given complete information about the components of a phrase we consequently have complete information about the phrase as a whole. Genuine cases of ambiguity are then realised by multiple edges in the chart. Again, any feature specifications which may be carried by a category licensed by *B'* ought to depend only upon *B** or its immediate constituents yl,...,yn. The problem then is to find some fairly natural constraint on ID rules which ensures that *B* is a fully instantiated category-descriptor, whenever each daughter yi is fully instantiated, and the entire string *y* is well-ordered.

There are many more or less natural ways in which such a condition could be imposed, involving feature passing conventions, feature restrictions, feature defaults and so on. In the present implementation however, the condition amounts to the following simple restriction on the form of ED rules, and the use of variables within category-descriptors:

- In an ID rule, any variable which occurs on the mother category-descriptor must also occur on at least one of the daughter category-descriptors.

It should be clear that if each of the daughters is fully instantiated, then the mother will also be fully instantiated (and in the absence of other constraints such as FCRs or FSDs will actually be uniquely determined). No feature value on the mother can be a variable, since this implies either that at least one daughter is under-specified (contrary to initial assumptions) or that the condition itself is violated (i.e. the variable is independent of the daughters).

Given these conditions on the grammars accepted by the parser, LP checking may be carried out correctly during COMPLETE. The conditions turn out to be quite minor in the sense that they do not seem to impose any restrictions on the expressive power of the grammar formalism. In effect, they simply force the grammar-writer to be explicit about possible sources of lexical or phrasal ambiguity.

4. The Semantic Component.

The NASEV parser incorporates a basic version of the *functional realisation principle* of GPSG [GKPS pp.209-211]. By this means, NL input may be translated into expressions of Montague's *intensional logic* [Montague 1974]. Functional realisation basically works by combining as functions and arguments those semantic representations associated with the immediate constituents of a phrase to create a new expression representing the meaning of the phrase as a whole.

Each lexical item may have associated with it a typed semantic representation of the following form:

mexp(<Exp>, <Type>).

where <Exp> is as well-formed expression of IL and <Type> is the semantic, or logical type of <Exp>. A term **mexp**(α, τ) may be read "α is a *meaningful expression* of type τ"

It is additionally necessary to supply type information about non-lexical categories. This is done simply by providing the parser with appropriate type statements of the form:

type(<Category>,<Type>).

where <Category> is a category-descriptor and <Type> is a type expression.

Functional realisation is performed during both PREDICT and COMPLETE. A predicate FREAL takes as its 'input' a list of meaningful expressions of IL (i.e. terms of the form given above) and a type expression, and builds a new meaningful expression of the indicated type. Expressions of IL are combined under *intensional functional application* [5], with each expression used only once. Since it is quite possible that expressions can be combined in different ways, FREAL may produce additional results on backtracking.

4.1. Intensional Logic.

Montague's IL is a system of higher-order logic employing a type hierarchy, higher-order quantification, lambda-abstraction and modal operators as well as 'intension' and 'extension' operators [Dowty *et al* 1981, Ch.6]. The semantic component of the NASEV parser utilises the following notational variant of IL (based on Dowty *et al* p.155-156).

Types.

Given the Prolog atoms **t**, **e** and **s**, then the set of types is defined recursively as follows.

1. t is a type.

2. e is a type.

3. if a and b are any types, then [a|b] is a type.

5. That is to say, all arguments to functors are implicitly assumed to be intensional. Introducing intensionality in this fashion permits a simplification of the semantics without any loss of generality.

4. if a is any type, then [sla] is a type.

Note that **s** is not a type.

A, Basic Expressions.

1. Logical constants of all types are represented by Prolog atoms (i.e. strings OJ alphanumerical characters starting with a lower case letter).

2. Variables of all types are represented by Prolog terms of the form:

v(<Var>)

where <Var> is a Prolog variable (i.e. a string of alphanumeric character beginning with a capital letter).

B. Syntactic Rules*

The set of meaningful expressions of type a is defined recursively as follows:

1. If v is a variable of type a, then mexp(v,a)

2. If c is a constant of type a, then mexp(ca).

3. If mexp(a,a) and mexp(v.b), where v is a variable, thei mexp(lambda(v,a),[bla]).

4. If mexp(a,[alb]) and mexp(/3,a) then mexp(app(a,j3),b).

5. If mexp(a,a) and mexp(j3,a) then mexp(eq(a,/3),t).

6.-10. If mexp(#,t) and mexp(^,t) then

6. mexp(not(0),t).

7. **mexp ¢r (φ,ψ0 .t).**

8. **mexp(and(φ,ψ),t).**

9. **mexp(imp(φ,ψ).t).**

10. mexp(iff(0,i/O.t).

11. If mexp(<£,t) and u is a variable of any type, then mexp(forall(u,0),t).

12. If mexp(ϕ,t) and u is a variable of any type, then mexp(exist(u,ϕ),t).

13. If mexp(ϕ,t) then mexp(nec(ϕ),t).

14. If mexp(ϕ,t) then mexp(future(ϕ),t).

15. If mexp(ϕ,t) then mexp(past(ϕ),t).

16. If mexp(α,a) then mexp(int(α),[s|a]).

17. If mexp(α,[s|a]) then mexp(ext(α),a).

4.2. Quantifier Storage.

The functional realization principle of GKPS is strictly compositional, and thus provides no means of treating quantifier ambiguities, which arise independently of any structural ambiguity. It is therefore assumed in GKPS "that quantifier ambiguities should be handled by some variant of 'Cooper storage'" [GKPS p.15], although no details of a suitable mechanism are provided.

Cooper's storage strategy represents a weakening of the strict compositionality assumed by Montague [Cooper 1983]. As a phrase is interpreted, the possibility of *storing* term phrases (i.e. NP denotations) arises as an alternative to combining them directly with the interpretations of other constituents. These stored *binding operators* remain 'on ice' until such time as they are *retrieved*, and combined with the rest of the interpretation. Since the order in which binding operators are retrieved does not depend upon the order of storage, it is possible to represent all scope assignments in this way.

Cooper's strategy naively generates all possible permutations of quantifiers with the result that a sentence containing *n* quantified NPs is predicted to have more than *n!* readings (although not all of these may be truth-conditionally distinct). Under certain circumstances this can give rise to over-generation. More specifically, the storage mechanism proposed by Cooper makes false predictions about the readings available for complex NPs such as those italicised in (7).

(7) a. *the agent of every company* arrived.
    b. *a player belonging to every team* was disqualified.
    c. *every attempt to find a unicorn* failed miserably.

In each of the above sentences, the interpretation of the embedded NP is integral to the interpretation of the outer quantifier, and thus the meaning of the subject NP as a whole. Roughly speaking, the problem with Cooper's strategy is that it fails to take this sort of dependency into account, and as a consequence, the semantic role of an embedded NP may 'get lost'. This can lead to anomalous interpretations.

A solution to the difficulties inherent in Cooper's approach is proposed in [Keller 1987]. Essentially, Keller's modification involves the introduction of extra structure into the store making it possible to encode the semantic dependencies which exist between NL quantifiers. Keller's *nested storage* technique forms the basis of the storage strategy adopted for the NASEV parser.

In the implemented version, storage is mandatory [6]. Each syntactic constituent recognised by the parser may have associated with it an appropriate semantic representation:

rep(<Mexp>, <Store>).

where <Mexp> is a typed, meaningful expression of IL (as defined in the previous section) and <Store> is a list of *referenced semantic representations*. Each referenced representation in the store consists of a unique *reference variable* paired with a semantic representation of a term-phrase:

ref(v(<Var>), <Repn>).

6. In contrast, both Cooper's original storage technique and Keller's modification are non-deterministic mechanisms.

In the implementation reference variables are taken to be of the same semantic type as term-phrases, and not of type individual (c.f. Cooper storage and Keller's modification). This move requires some redefinition of the storage and retrieval operations, but should permit both *de dicto* and *de re* readings to be generated from the semantic representations, even though the storage mechanism operates deterministically. Roughly speaking, term-phrases may either be directly substituted for reference variables (leading to narrow-scope interpretations) or else a 'dummy pronoun* is first substituted, followed by an application of Keller's storage retrieval operation. The latter ensures that all of the valid wide-scope readings may be produced. At present the appropriate retrieval operations are not implemented.

5. Parser Environment.

A small environment has been written for the parser in order to make it more convenient to use. The environment supports a number of parse modes which are intended to facilitate grammar-development, and testing of the parser itself. Although by no means a finished product, the main features of the support system are outlined in this section for the sake of completeness.

5.1. Output Display Routines.

To help the user interpret the syntactic and semantic representations built by the parser, the support system includes a package of display routines. With the display package loaded parse trees are printed out in indented list format:

```
cat(2,-,+,sing,3.~)
—  cat(l,+,-,sing,3,nom)  he
—   cat(l,-.+,sing,3,~)
————cat(0,-,+.sing,3,~)  sees
————cat(l,+,-,plur,l,acc)  us
```

The user may additionally specify suitable *aliases* for category-descriptors, which are then used by the formatting routines. An alias definition is of the form:

```
alias(<Category>, <Alias>).
```

where <Category> is a category-descriptor, and <Alias> is an appropriate Prolog atom or term. So for example

```
alias(cat(1,-,+,X,Y,~), vp(X,Y)).
```

sets up a generic VP alias. With further suitable definitions of this kind the display routines might produce the following output instead of that shown above:

```
s
— np(sing,3) he
— vp(sing,3)
——— v(sing,3) sees
——— np(plur,1) us
```

A 'logic paraphrase' mechanism is also included in the package, although this is of fairly limited use. In storage mode it produces output of the following kind:

```
Expression type: t

(v(0)) arrive

        v(0) -> EVERY (agent (of v(1))) ()

                v(1) -> SOME company ()
```

It is important to note that the paraphrase does not constitute a separate level of semantic representation. It is intended to serve simply as a readable approximation to the actual IL translation from which it is derived.

5.2. Parse Modes.

A number of parse modes are available to the user. Each mode may be turned on or off independently of the others using a switch command:

```
switch(<Mode>, <State>).
```

where <State> is either **on** or **off**, and <Mode> is one of the following:

display	Display mode default state is on. In off mode the parser simply prints out a message alerting the user to the success or otherwise of an analysis.
trace	Trace mode default state is off. When on, the parser prints out a trace of all edges as they are produced during analysis.
count	Count mode default state is off. When on, a count of the total number of edges introduced by the parser is printed out at termination of analysis.
semantics	Semantics mode default state is off. When on, the parser performs functional realisation and builds up semantic representations of its input.
storage	Storage mode default state is off. In semantics mode, switching storage on causes term-phrases to be stored up rather than combined with other semantic representations by functional realisation.
paraphrase	Paraphrase mode default state is off. In semantics mode and with display on, switching paraphrase on causes semantic representations to be printed out as simple paraphrases.
propose	Propose mode default state is off. Switching propose on forces the parser to make intelligent guesses about the syntactic category of any unknown lexical item it comes across in the input.

6. Summary and Conclusion.

What has been described in this report is a chart parsing system for a GPSG-like grammar formalism. The parser accepts rules in ID/LP format with category-labels represented by complex feature structures. A simple implementation of the functional realization principle, augmented with a quantifier storage mechanism enables semantic representations of NL input to be built up in parallel with syntactic analysis.

The implementation has demonstrated the practicability of Earley-based strategies for parsing with ID/LP grammars. A consideration of the problems arising from the use of partial representations of categories (i.e. complex symbols) in conjunction with linear precedence rules has additionally lead to insights into direct parsing with GPSG-like formalisms. In particular it has been shown that certain restrictions on the form of grammars may be motivated on *computational* grounds.

This point is important and has ramifications for subsequent work on the NASEV parser. It is noted that GPSG was initially chosen for the NASEV project because it provides a formally precise and linguistically well-motivated theory of linguistic knowledge. However, the formalism *lacks* a useful computational interpretation. Indeed, a fundamental mis-match exists between the extensional, and essentially static conception of the formalism as presented in GKPS, and the way in which partial information may be understood in computational terms.

This mis-match shows up whenever it becomes necessary to seek a sensible *computational* interpretation for a particular component of the GPSG formalism. LP rules, for example, properly apply to categories, and not to their representations, complex symbols. The former are complete, fully specified linguistic objects, whilst the latter are typically under-specified, or partial. Yet a parser operates on objects of the latter kind and not the former, and so a more suitable interpretation of LP rules must be found (in this case with consequences for the rest of the formalism).

Likewise, the FCRs, FSDs, and FIPs of GPSG are correctly understood as simultaneous well-formedness conditions on fully specified linguistic objects (categories or local trees). Finding suitable computational interpretations of these components of GPSG as constraints and operations on partial objects will be the focus of future research.

References.

Barton, G.E., R.C. Berwick and E.S. Ristad (1987). *Computational Complexity and Natural Language*. Cambridge, Mass.: MIT Press.

Clocksin, W.F. and C.S. Mellish (1971). *Programming in Prolog*. Berlin: Springer-Verlag.

Cooper, R. (1983). *Quantification and Syntactic Theory*. Dordrecht: D. Reidel.

Doerre, J. and S. Momma (1985). Modifikationen des Earley-Algorithmus und ihre Verwendung fuer ID/LP Grammatiken. Stuttgart: Institute for Linguistik/Romanistik, University of Stuttgart.

Dowty, D.R., R.E. Wall and S. Peters (1981). *Introduction to Montague Semantics*. Dordrecht: D. Reidel.

Earley, J. (1970). An Efficient Context-Free Parsing Algorithm. *Communications of the ACM 13* pp.94-102

Gazdar, G., E. Klein, G.K. Pullum & I.A. Sag (1985). *Generalized Phrase Structure Grammar*. Oxford: Blackwell.

Kasami, T. (1965). An efficient recognition and syntax algorithm for context-free languages, *Scientific Report AFCRL-65-758*, Air Force Cambridge Research Lab, Bedford, Mass.

Keller, W.R. (1987) Nested Cooper Storage: The Proper Treatment of Quantification in Ordinary Noun Phrases. *Cognitive Science Research Paper, 78 (CSRP 078)*, Brighton: University of Sussex.

Kilbury, J. (1984a) A Modification of the Earley-Shieber Algorithm for Direct Parsing of ID/LP Grammar. Ms, Berlin: Technical University Berlin.

Kilbury, J. (1984b) Earley-basierte Algorithmen fuer direktes Parsen mit ID/LP Grammatiken. *KIT-16*, Berlin: Technical University Berlin.

Montague, R. (1974). *Formal Philosophy*. New Haven: Yale University Press.

Shieber, S.M. (1984) Direct Parsing of ID/LP Grammar. *Linguistics and Philosophy 7*, pp.135-154.

Tomita, M (1985). An efficient Context-Free Parsing Algorithm for Natural Languages, in *Proceedings of the 9th International Joint Conferenc:e on Artificial Intelligence*.