A Formalism and Environment for the Development of a Large Grammar of English

Ted Briscoe, Claire Grover

Department of Linguistics, University of Lancaster, Bailrigg, Lancaster LA1 4YT, England

Bran Boguraev, John Carroll

Computer Laboratory, University of Cambridge, Corn Exchange Street, Cambridge CB2 3QG, England

ABSTRACT

Natural language grammars with large coverage are typically the result of many person-years of effort, working with clumsy formalisms and sub-optimal software support for grammar development. This paper describes our approach to the task of writing a substantial grammar, as part of a collaboration to produce a general purpose morphological and syntactic analyser for English. The grammatical formalism we have developed for the task is a metagrammatical notation which is a more expressive and computationally tractable variant of Generalized Phrase Structure Grammar. We have also implemented a software system which provides a highly integrated and very powerful set of tools for developing and managing a large grammar based on this notation. The system provides a grammarian with an environment which we have found to be essential for rapid but successful production of a substantial grammar.

I INTRODUCTION

As part of the Alvey Programme, the UK research initiative in Information Technology, three collaborating projects (at Cambridge, Lancaster and Edinburgh Universities) have been set up to produce a general purpose morphological and syntactic analyser for English. One project is developing an integrated morphological analyser and lexicon package (Russell et al. 1986). The grammar project, which is the focus of this paper, has developed a metagrammatical notation and implemented an associated software system to facilitate the writing of a substantial grammar of English. The third project has implemented a chart parser augmented with unification (Phillips and Thompson 1987), which analyses sentences of English morphosyntactically using the rule systems and lexicon developed by the other projects.

In this paper, we first discuss the metagrammatical notation in which the grammar (Grover et al. forthcoming) is being written and then describe the associated grammar development environment (Carroll et al. 1987) which provides the grammarian with software tools to facilitate the rapid development of a large but consistent and tractable grammar. The metagrammatical notation is a development of Generalized Phrase Structure Grammar (GPSG) (Gazdar et al. 1985) but is designed to be both more *expressively powerful* and more *computationally tractable*. The grammar development environment is modelled on similar systems (e.g. Evans 1985, Karttunen 1986), but goes beyond them in terms of *speed, efficiency*, and the provision of a *genuinely integrated* lexical, morphological and syntactic development environment.

II THE METAGRAMMATICAL NOTATION

The metagrammatical notation we have designed is based on GPSG, but has been modified and extended to be more flexible and expressive and is interpreted somewhat differently. The first motivation for these changes is to define a notation which provides a specialised programming language for specifying grammatical theories and grammars for particular languages, rather than defining a specific theory directly. This approach both ensures that developments in syntactic theory will be less likely to render the grammar development environment obsolete and also provides the grammarian with an expressively flexible and powerful notation for grammar writing, rather than a formalism embodying a restrictive theory.

The second motivation for reinterpreting and modifying the rule types of GPSG is to provide a procedural and strictly ordered interpretation of the various rule types. In GPSG, rules are defined declaratively as applying simultaneously in the projection from Immediate Dominance (ID) rules to local trees. As Shieber (1986), Ristad (1986) and others have pointed out this leads to a computationally complex and conceptually difficult system. The approach taken here is broadly that used by Popowich (1985) for Definite Clause Grammars (Pereira and Warren 1983), and that outlined (although not implemented) by Shieber (1986) for GPSG. Metagrammatical rules are expanded into an 'object' grammar which is a unification grammar. The interpretation of the metagrammatical rules is provided by the procedure for their expansion and the much simpler semantics of the resulting object grammar. Furthermore, because no attempt is made to expand out into a pure context-free formalism, the object grammar remains manageable.

The object grammar consists of a set of phrase structure rules whose categories are feature complexes, and which form the input to the parser. An example of an expanded rule of the object grammar for forming a variety of English clauses is given in Figure 1. Features consist of feature name / feature value pairs. Feature values can be variables (the values beginning with @ in the figure), which can be bound within the rule. Parsing with the object grammar involves matching categories by unifying their feature sets. Unlike GPSG, our metagrammar defines a set of partially instantiated phrase-structure rules and not a set of local trees sufficiently instantiated to define a portion of syntactic structure. However like GPSG, the metagrammar is designed to capture linguistic generalisations; for this reason, and also to simplify and abbreviate the statement of 'object' rules, such as the one in the figure, it contains statements of the following eleven kinds:

```
[N -, V +, BAR 2, SUBJ +,      -->     [N +, V -, BAR 2, POSS -,   [N -, V +, BAR 2, SUBJ -,
 WH @1, INV @2, NEG @3, AGR @4          WH @1, CASE NOM, PRO @8,    H +, INV @2, NEG @3, AGR @4,
 VFORM @5, AUX @6, FIN @7, COMP NORM]   AGR @4]                     VFORM @5, AUX @6, FIN @7]
```

Figure 1. An Object Grammar Rule.

A. Feature Declarations

Feature Declarations define the feature system used by the grammar. They encode the possible values of a given feature. The feature system is very similar to that used in GPSG, but a feature can additionally take a variable value which ranges over the set of actual values as declared.

```
VFORM {BSE, ING, EN, TO}    SLASH CAT
```

B. Feature Set Declarations

Feature Set Declarations define sets of features which propagate in the same manner and which will appear together on particular categories. For example, relevant features to propagate between NPs and head noun daughters (PLU, PER, etc.) may be grouped together in a set called NOMINALHEAD, and the name of this set used later in the rules which perform this type of propagation to refer to the whole collection of features.

```
NOMINALHEAD = {PLU, PER, CASE,...}
```

C. Alias Declarations

Aliases are a convenient abbreviatory device for naming categories and feature complexes in rules. They do not affect the expressive power of the formalism.

```
NP = [N +, V -, BAR 2].    Nom = [CASE NOM].
```

D. Category Declarations

Category Declarations define a particular category as consisting of a given set of features. These declarations are used to expand out the partially specified categories which typically appear in ID rules. They also make the system more explicit by obliging the grammarian to state which categories a feature will appear on. Category Declarations replace part of the function of Feature Cooccurrence Restrictions in GPSG.

```
Nominal: NP => NOMINALHEAD
```

E. Extension Declaration

Some features, such as SLASH, are not part of the 'basic' make up of a category and in this system tend to be added to categories in ID rules by the application of metarules. Features which appear in categories only by virtue of metarule application must be defined as extension features; doing this again makes the system more explicit.

```
Extension = {SLASH,...}
```

F. Immediate Dominance Rules

ID rules encode permissible dominance relations in phrase structure trees. The immediate dominance properties of the 'object' rule in Figure 1 can be expressed by the Clauses rule below; however other properties of the object rule (e.g. the ordering of the categories in it) are determined by other types of rules in the grammar. Transitives below is another example of an ID rule. It states that a transitive VP will contain a lexical head, which must be subcategorised as transitive, and a NP sister.

```
Clauses: S[COMP NORM] --> NP[Nom], VP.

Transitives: VP --> H[SUBCAT NP], NP.
```

G. Linear Precedence Rules

Linear Precedence (LP) rules encode permissible precedence relations in ID rules. The first rule below states that NPs always precede VPs in English PS rules which contain both categories on the right-hand side. The second states that categories bearing the SUBCAT feature (i.e. lexical categories) precede those having no such specification.

```
NP < VP.    [SUBCAT] < [~SUBCAT].
```

H. Phrase-Structure Rules

PS rules encode both permissable dominance and precedence relations in phrase markers. They are included in the formalism so that the expressive power of the system is not restricted by the ECPO property* (Gazdar et al. 1985). Our current grammar makes no direct use of this rule type but the grammarian can at any point choose to abandon the ID/LP format and use a mixed system, entering directly a few 'marked' PS rules (distinguished from ID rules by their lack of commas) which do not conform to the general LP rules for English.

```
Heavy_NP_Shift:
      VP --> H[SUBCAT NP_PP] PP NP[+Heavy].
```

I. Feature Propagation Rules

Feature Propagation rules define how features propagate between mother and daughter categories in ID or PS rules (if either mother or daughter has a variable value for one of these features). The effect of propagation rules is to bind variables or instantiate values of features in rules of the 'object' grammar. Propagation rules can be used to encode particular feature propagation principles, such as the various versions of the Head Feature Convention proposed for GPSG. However, such principles are not 'hard-wired' into the formalism, so that maximum flexibility and expressiveness is maintained. Propagation rules are stated in terms of ID or PS rule patterns which may contain variables over categories (W); for example, the following rule states that NPs inherit features from the head (noun) daughter in any ID rule which is nominal, contains a head and which may optionally contain other daughters of any type.

```
Nominal: [+N, -V] --> [+H], W.
                F(0) = F(1), F in NOMINALHEAD.
```

J. Feature Default Rules

Feature Default rules default specified values onto features with no value in categories in a particular environment in an ID or PS rule. These default rules replace Feature Specification Defaults in GPSG; because the rules assign values to features in the context of an ID or PS rule, their application can be accurately controlled, and thus the need for Feature Cooccurrence Restrictions to prevent the construction of 'illegal' categories is diminished. Both Feature Default rules and Feature Propagation rules have a similar syntax to Kilbury's (1986) independently motivated Category Cooccurrence Restrictions, although their function is somewhat different. The Accusative default below states that an NP which is a sister to a verbal or prepositional lexical head will be accusative (if unspecified for any other CASE value).

```
Accusative: [-N] --> H, NP. CASE(2) = ACC.
```

* A grammar has the Exhaustive Constant Partial Ordering property if the set of expansions (defined by ID rules) of any one category observes a partial ordering that is also observed by the expansions of all the other categories.

K. Metarules

Metarules encode systematic relationships between sets of ID or PS rules and automatically add further rules to the basic set produced by the grammar writer and the application of other metarules. Metarules can be written to apply to ID rules or PS rules (including those which result from linearisation of ID rules). Metarules can also be restricted to apply to ID or PS rules containing a lexical head through use of the `W` category variable (as opposed to the `U` 'unrestricted' category variable) in the input pattern. For example, `Passive` adds a new `VP` rule without the `NP` direct object (and with an optional 'by' phrase) for every basic `VP` rule with a lexical head and a non-predicative `NP` daughter. `Slash` creates new ID rules with a slash feature appearing on one of the non-head daughters with a variable value which is bound to the slash feature of the mother.

```
Passive: VP --> NP[-Prd], W. ==>
           VP[Pas] --> W, (PP[by]).

Slash: VP --> H, XP, U. ==>
         VP[SLASH @x] --> H, XP[SLASH @x] U.
```

III METAGRAMMAR INTERPRETATION

In GPSG, the set of legal local trees is defined declaratively by simultaneous application of the various rule types during the 'projection' from ID rules to fully instantiated local trees. In this system, although the metagrammatical notation shares many similarities with GPSG, there are crucial differences in the interpretation of the notation. The rules of the metagrammar jointly specify a set of partially instantiated phrase structure rules, rather than fully instantiated local trees. That is, rules are allowed to contain variable values for features and these variables are instantiated at parse time by unification. Propagation rules specify restrictions on the instantiation of these variables; for example, the rule

```
PLU_agreement:
     NP[PLU @1] --> Det[PLU @1], N1[PLU @1].
```

will force agreement between the determiner and nominal phrase in the ID rule for `NP`s which introduces determiners and nominal phrases, by binding the `PLU` variables on all the categories. If we assume that whereas 'a' is specified as `[PLU -]`, 'the' is unspecified for a value of `PLU`, then the result of matching 'the' to the `Det` category in the expanded ID rule will not instantiate `@1`. However, matching 'boys' to `N1` will instantiate all the instances of `@1` to +. Therefore, 'a boys' will not be accepted by the expanded ID rule because `@1` cannot both be instantiated as + and −.

The object grammar is produced by applying category, propagation and default rules to ID (and PS) rules in that order and then applying the non-linear metarules one-by-one in order to the set of fleshed out ID rules*. After each metarule has applied, propagation and default rules are applied to any new ID rules and these are added to the original set before the application of the next metarule. Next, the resulting expanded set of rules is linearised according to the LP rules and any linear metarules are applied to the complete pool of PS rules. This procedure is summarised in Figure 2.

In essence, this grammatical formalism combines Prolog-style unification, as used in Definite Clause Grammars (Pereira and Warren 1983) and PATR-II (Shieber 1984), with a metagrammatical notation designed to capture linguistic generalisations and so simplify the statement of the grammar. The formalism defines a 'space' of potential grammatical theories which can be explored by the theoretical linguist. It also allows full use of any mixture of these theories for the flexible specification of a wide coverage grammar of a particular natural language.

IV THE GRAMMAR DEVELOPMENT ENVIRONMENT

The grammar development environment (GDE) is a software system which has been developed to facilitate the writing of a grammar for a significant fragment of English in our grammatical formalism. By analogy with present-day programming environments (Barstow et al. 1984) where software development is supported by powerful interactive programming tools, the GDE provides grammar writers with a set of tools for grammar development, allowing them to input, edit and generally manage the necessarily large number of metagrammatical rules and monitor the interactions between them. The GDE is also able automatically to expand out such a set of rules into an object grammar, and from the collaborating projects incorporates versions of the parser and of the morphological analyser with a lexicon of approximately 5000 entries.

The GDE is a 'second generation' system, drawing on several ideas first implemented in systems such as ProGram (Evans 1985), GPSGP (Phillips and Thompson 1985), and D-PATR (Karttunen 1986), but differs from them in that it was designed to satisfy all three of the following criteria:

- to be easy to use
- to enable a grammarian to achieve high productivity
- to be portable and machine independent.

A. Ease of Use

The first objective, simplicity of use and robustness, is particularly important to a linguist with little or no experience of the system. Thus, since the metagrammatical notation is based on GPSG, the syntax accepted for its rules follows that of Gazdar et al. (1985) as closely as possible. The user of the system is therefore likely to be familiar with the GDE's rule notation and would not have to learn a new notation, as would be necessary when starting to use the (also GPSG-based) ProGram system (Evans, 1985). For example, Figure 3 gives an example of a metarule as it has to be presented, firstly to ProGram, and secondly to the GDE.

* This approach to metarule expansion is more restrictive than Thompson's (1982) principle of finite closure.

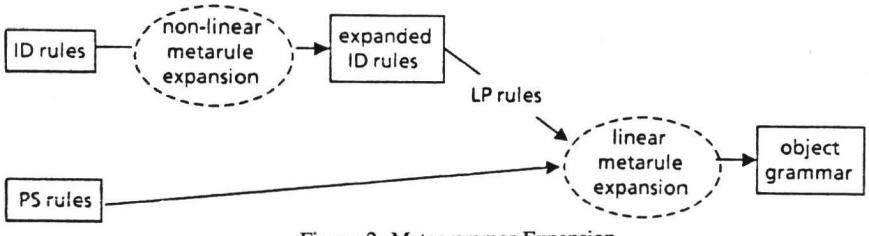


Figure 2. Metagrammar Expansion.

```
Passive:
  (VP1 --> np(-prd) , ...
     where VP1 is vp)
==>
  (VP2 --> ... , opt(pp(by))
     where VP2 is vp(pas),
           VP1 matches VP2).


Passive:
  VP --> NP[-Prd], W. ==>
    VP[Pas] --> W, (PP[by]).
```

Figure 3. The Contrast Between ProGram and GDE Declarations.

ProGram requires the user to input the grammar both in a syntax that will be acceptable to the underlying PROLOG system lexical analyser, and one which corresponds closely to the internal data-structures used by ProGram. Apart from being idiosyncratic, this notation is difficult to use and a syntax error in a declaration is likely to provoke a PROLOG error message. The GDE, on the other hand, has its own grammar declaration parser and gives informative diagnostics for invalid input, based on the type of grammar item (e.g. feature name) it was expecting.

The GDE is an environment with many powerful capabilities; however we disagree with Teitelman and Masinter (1984) (but in the context of the development of grammars rather than programs) that a powerful environment need necessarily be only for the expert user. All interaction with the GDE is carried out through a logically structured set of commands which are able to prompt at any stage as to the type of input expected next. There is also automatic command completion. GDE commands perform high level operations; for example in a single command the user can instruct the GDE to display all the rules resulting from applying a given subset of metarules to a specified set of ID rules:

```
view id VP*(PASS,SAI)
;; all VP rules under PASS and SAI metarules
```

Performing a similar task using ProGram involves the user in a lengthy interaction to select the ID rules and metarules concerned, followed by explicit commands to 'normalise' both sets of rules and perform the metarule expansion. Thus in general, the major difference between the two systems is that with the GDE the user specifies a goal for the system to achieve, whereas to achieve the same goal with ProGram the user has to devise a sequence of low level operations and lead the system through them by hand.

B. Encouraging High Productivity

The second design criterion was to help a more experienced user of the system achieve as high a productivity as possible. The GDE encourages an interactive and incremental style of grammar development by providing metagrammatical rule editing facilities, and by enabling the grammarian to quickly identify incorrect rules and assess the consequences of changing them. Identifying the incorrect rules is usually the harder task. With other grammar development environments, D-PATR (Karttunen 1986) being a good example, the grammarian has to study syntax trees produced by the parser, and sometimes only from incorrect feature values in them try to deduce which rules are incorrect. In the GDE the name of an incorrect PS rule in the object grammar can be read directly from the portion of a parse tree that is faulty. PS rule names are unique and each is generated from the name of the ID rule in the metagrammar and the names of any metarules that were involved in its formation; multiple linearisations of the same rule are also distinguished. Thus, the faulty ID rule, metarule or LP rule in the metagrammar is easily identified. As well as a parser the GDE also contains a sentence and sentence fragment generator which is particularly useful for detecting rules which cause overgeneration and so degrade the performance of the grammar.

When an incorrect rule is found, it may be edited within the GDE. The high level rule display commands (mentioned above) are often sufficient for assessing the consequences of the rule change, and using them is usually a more reliable strategy than just another attempt to parse several more sentences. The GDE performs extensive 'bookkeeping' on behalf of the grammarian, including keeping track of which files contain rules that have been changed and so should eventually be saved, and regenerating the object grammar when required after a change to the metagrammar. As rules are added to the system, the grammarian can constantly monitor the coverage and performance of the grammar by parsing a corpus of sentences which are maintained by the system and which are intended to represent the coverage of the grammar at any one time. The grammarian is freed by the bookkeeping from having to worry about details of management and is thus able to concentrate attention on grammar development.

Of course, the powerful facilities provided by the GDE would be of limited use if the actual system were not quick enough in execution to keep up with the grammarian; with metagrammars of the size currently being developed it has proven necessary to make the GDE cache several types of intermediate result, such as the rules resulting from metarule application. Expanding a large metagrammar (in order to be able to parse a sentence) from scratch currently takes of the order of three minutes on a Motorola 32016-based workstation, but after this has been done once, re-expansion when the GDE detects a change to the metagrammar rarely takes more than a few seconds. Parsing a sentence seldom takes longer than five seconds (using the highly optimised chart parser from the collaborating parsing project). In contrast, since ProGram and D-PATR were not developed with efficiency in mind, their processing times for single sentences (even using smaller grammars) of a minute or more significantly limit the productivity of the experienced user.

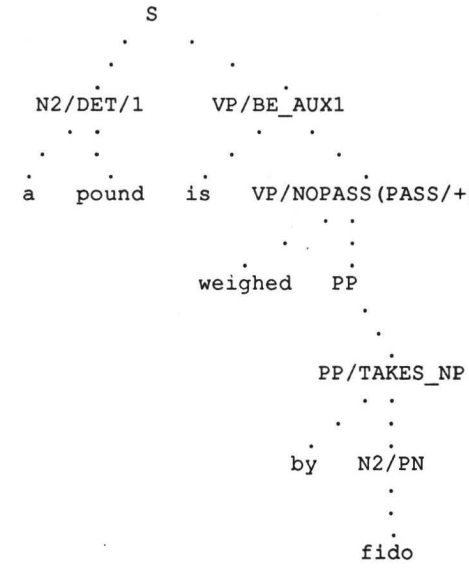
C. Portability

Although the GDE was implemented as a software tool for a specific project, the flexibility of our metagrammatical formalism (section III) makes the GDE of potential use to the more general community of linguists and grammarians. It is intended that the software should be freely available, and so firstly it is *portable*, being originally written in Cambridge Lisp (a derivative of Portable Standard Lisp) and since translated into Common Lisp, and secondly *machine independent*, since it does not count on being able to use a mouse or any windowing capability. Indeed the same version of the GDE runs on an IBM mainframe, a GEC minicomputer and an Acorn single-user workstation. In contrast, any serious work using ProGram requires the Poplog (Hardy 1982) editor running on a specific type of terminal, and D-PATR needs to be able to call the Interlisp-D windowing functions and the special system text editor (TEDIT).

V AN EXAMPLE GRAMMAR DEVELOPMENT SESSION

The GDE interaction that follows demonstrates several of the points made in the previous section. The major point is that the output of a single GDE command is often sufficient to pinpoint a bug in the grammar. Other features highlighted are the integrated nature of the tools (e.g. parser, generator) provided, and the bookkeeping that is performed to allow the grammarian to focus fully on grammar development. The user first reads in a file containing an existing (small) grammar, and tries to parse a couple of sentences. (User input in bold).

```
Gde> read gram.rules
File read

Gde> parse
Compiling grammar...
17 ID rules, 1 metarules,
          6 propagation rules, 3 default rules
*** Warning, multiple
           match between VP/TAKES_2NP and PASS
31 expanded ID rules, 32 linearised rules

Parse>> fido weighs a pound

80 msec CPU, 1000 msec elapsed, 367 conses
15 edges generated
1 parse

((fido) (weighs (a pound)))

Parse>> a pound is weighed by fido

140 msec CPU, 1000 msec elapsed, 756 conses
32 edges generated
1 parse

((a pound) (is (weighed ((by (fido))))))

Parse>> view rules
```



The second sentence should not have received a parse, so there must be a bug in the grammar. Viewing the parse tree immediately suggests where the problem lies: the expansion of the metagrammar has produced a rule VP/NOPASS(PASS/+), which both appears at the place where the parse should have failed, and which also from its name looks suspect. The user goes on to exit the parser, examine the ID rule and metarule involved, and restrict the applicability of the latter.

```
Parse>> q

Gde > view id *NOPASS

VP/NOPASS :
   VP --> H[SUBCAT NOPASS], N2[+PRD].

Gde> view metarule PASS

PASS : VP --> W, N2. ==>
         VP[Pas] --> W, (P2[PFORM BY]).

Gde> input
Construct type? meta
Metarule declaration? PASS :
> VP --> W, N2[-PRD]. ==>
>   VP[Pas] --> W, (P2[PFORM BY]).

Replace existing definition (y/n)? y

Gde> names id *(PASS)

VP/TAKES_NP(PASS/-)   VP/TAKES_NP(PASS/+)
VP/DITR(PASS/-)       VP/DITR(PASS/+)
VP/TAKES_2NP(PASS/-)  VP/TAKES_2NP(PASS/+)
VP/OR(PASS/-)         VP/OR(PASS/+)
```

The VP/NOPASS ID rule no longer appears in the list of rules resulting from the updated PASS metarule; the rules that do appear are the expected ones. Carrying on, an attempt to parse the last sentence indeed fails as it should. The relevant parts of the metagrammar are automatically reexpanded since the GDE remembers that PASS has been changed.

```
Gde> p
Compiling grammar...
17 ID rules, 1 metarules,
          6 propagation rules, 3 default rules
25 expanded ID rules, 26 linearised rules

Parse>> previous
(a pound is weighed by fido)

80 msec CPU, 1000 msec elapsed, 478 conses
21 edges generated
No parses

Parse>> q

Gde> generator N2

Gen>> s s 2
a dog
dog a
kim

Gen>> q

Gde> write gram.rules
Backing up file gram.rules
Writing file gram.rules
```

An exhaustive generation of all noun phrase structures licensed by the grammar indicates that the rule introducing determiners may be overgenerating, but the user decides to ignore this for the time being, and write the changed grammar back to disc. The GDE first saves the existing version of the file in case the user later wants to refer to it.

This example interaction is representative of real ones in the process of developing a large grammar. Our experience is that a

system such as the GDE is essential for rapid but successful production of a substantial grammar in a metagrammatical notation.

VI CONCLUSIONS

The usefulness of our formalism and associated grammar development environment can only be assessed ultimately on the basis of its success in allowing rapid development of grammars (and associated theories and analyses); work on expanding the coverage of the grammar is still in progress, but currently, on the basis of only 10 person-months of development, the English grammar contains detailed analyses of:

- most VP complements, including the various control constructions
- the auxiliary system
- declaratives, imperatives and passives
- y/n and constituent questions, topicalisation
- NP complements, VP and PP modifiers
- ordinary, zero and free relatives
- the NP specifier system
- AP complements.

A recent metagrammar contained 127 ID rules, 34 Metarules, 41 Propagation rules, 11 Default rules, and 16 LP rules, which produced an expanded object grammar of 478 PS rules. This makes the current grammar roughly equivalent in size (though not coverage) to Diagram* and the Critique project grammar**. These latter grammars are widely recognised to be two of the largest developed for machine; however, both involved considerably greater human effort.

ACKNOWLEDGEMENTS

The system described in this paper evolved as a team effort and was greatly influenced by the comments of members of the two collaborating projects—Alan Black, John Phillips, Steve Pulman, Graeme Ritchie, Graham Russell and Henry Thompson—as well as from discussion with Bob Moore and Stu Shieber. We are grateful for comments on earlier versions of this paper from Roger Evans and Graham Russell. The work was supported by research grant GR/D/05554 from the UK Science and Engineering Research Council under the Alvey Programme.

REFERENCES

Barstow, D., E. Shrobe and E. Sandewall, *Interactive Programming Environments*. New York: McGraw Hill, 1984.

Carroll, J., B. Boguraev, C. Grover and E. Briscoe, "A Development Environment for Large Natural Language Grammars", Technical Report (to appear), Computer Laboratory, University of Cambridge, 1987.

Evans, R., "ProGram – a Development Tool for GPSG Grammars". *Linguistics*, 23:2 (1985) 213–243.

Gazdar, G., E. Klein, G. Pullum and I. Sag, *Generalized Phrase Structure Grammar*. Oxford: Blackwell and Cambridge, Mass.: Harvard University Press, 1985.

Grover, C., E. Briscoe, J. Carroll and B. Boguraev, "The Alvey Natural Language Tools Project Grammar – a Large Computational Grammar of English", Lancaster Papers in Linguistics, Department of Linguistics, University of Lancaster, forthcoming.

Hardy, S., "The POPLOG Programming System", Cognitive Studies Research Paper No. CSRP 003, University of Sussex, 1982.

Jensen, K., G. Heidorn, S. Richardson and N. Haas, "PLNLP, PEG and CRITIQUE: Three Contributions to Computing in the Humanities", Research Report RC 11841, Computer Sciences Department, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1986.

Karttunen, L., "D-PATR: A Development Environment for Unification-Based Grammars". *Proc. 11th International Conference on Computational Linguistics*, Bonn, Germany, 1986, 74–80.

Kilbury, J., "Category Cooccurrence Restrictions and the Elimination of Metarules". *Proc. 11th International Conference on Computational Linguistics*, Bonn, Germany, 1986, 50–55.

Pereira, F. and D. Warren, "Parsing as Deduction". *Proc. 21st Meeting of the Association for Computational Linguistics*, MIT, Cambridge, Mass., 1983, 137–144.

Phillips, J. and H. Thompson, "GPSGP – a Parser for Generalized Phrase Structure Grammars". *Linguistics*, 23:2 (1985) 245–261.

Phillips, J. and H. Thompson, "A Parser and an Appropriate Computational Representation for GPSG" in E. Klein and N. Haddock (Eds.), *Cognitive Science Working Papers 1*, Centre for Cognitive Science, University of Edinburgh, 1987.

Popowich, F., "SAUMER: Sentence Analysis using Metarules". *Proc. 2nd Conference of the European Chapter of the Association for Computational Linguistics*, Geneva, Switzerland, 1985, 48–56.

Robinson, J., "DIAGRAM: a Grammar for Dialogues". *Communications of the ACM*, 25:1 (1982) 27–47.

Russell, G., S. Pulman, G. Ritchie and A. Black, "A Dictionary and Morphological Analyser for English". *Proc. 11th International Conference on Computational Linguistics*, Bonn, Germany, 1986, 277–279.

Shieber, S., "The Design of a Computer Language for Linguistic Information". *Proc. 22nd Annual Meeting of the Association of Computational Linguistics*, Stanford, California, 1984, 362–366.

Shieber, S., "A Simple Reconstruction of GPSG". *Proc. 11th International Conference on Computational Linguistics*, Bonn, Germany, 1986, 211–215.

Teitelman, W. and L. Masinter, "The Interlisp Programming Environment" in D. Barstow, H. Shrobe and E. Sandewall (Eds.), *Interactive Programming Environments*, New York: McGraw Hill, 1984, 83–96.

Thompson, H., "Handling Metarules in a Parser for GPSG" in M. Barlow, D. Flickinger and I. Sag (Eds.), *Developments in Generalized Phrase Structure Grammar: Stanford Working Papers in Grammatical Theory, Volume 2*, Bloomington, Indiana: Indiana University Linguistics Club, 1982.

* Robinson (1982) suggests that Diagram's core then contained about 100 constituent structure rules.
** Jensen et al. (1986) give a figure of 235 decoding rules representing the "central ... grammatical structures of English".