University of Reading
Department of Cybernetics

Style in Natural Language Generation

A First Year Report

Karl R. Wilcox

September 1996

1. Introduction

1.1 Background

This report concerns the production of human language by computer, a field known as Natural Language Generation (NLG). In particular it discusses the way that language can be produced in various "styles" and suggests what the author believes may be a wholly new approach to style within NLG.

1.2 Why Study Style?

Natural languages are enormously rich and complex, allowing many subtle nuances and shades of meaning. We are able to express ourselves in many different ways, communicating more than the surface meaning of our utterance would suggest. Is it really necessary to model this in an NLG system?

In the early part of this century Charles Ogden developed Basic English (Ogden 1930), a simplified version of English requiring only 150 adjectives, 600 nouns and 100 "operative" words. This language appears to accomplish everything that is needed for the simple act of communication and NLG systems of this complexity already exist today. Would it not be sufficient for an NLG system to simply implement this language?

Basic English did not gain wide acceptance, and neither, the author believes would a natural language interface that implemented such a restricted subset of language. Such a system may be sufficient for simple communication of facts, requests and so on but it is likely to be insufficient to convey additional, situational information. Typically, this information is conveyed by particular uses of minor language features and can convey the geographical origin of the speaker; their social standing relative to the hearer; the type of activity that they are engaged in and so on. These "minor" language features can be used in two ways; by the hearer to extract additional information about the speaker; and by the speaker to convey information above and beyond the simple sequence of words would suggest. With the restricted language of Basic English it is likely that there is insufficient variability to express this extra linguistic information.

The apparent redundancy of natural language is what makes different styles possible, without redundancy there is only one way to express any given concept. Although this may be desirable in certain highly formal situations (e.g. specification languages) it is not the most effective way to maximise the communication potential between human and machine.

1.3 Potential Applications

Clearly, the ability to tailor the style of computer generated language is useful in almost every type of human computer interaction, but let us consider some specific examples from a domain familiar to the author; the airline industry. Even with today's aircraft, knowledge of weather conditions is still

vital for flying operations. Weather reports are received electronically from around the world but in a highly codified, largely numeric form. Although some pilots choose to use this form directly, others prefer a “natural language decode”, which for example would translate “05350” as “wind speed 5 knots, direction 350°”. If however, it was required to display the same information to passengers a more appropriate form might be “light northerly winds”.

Similarly, typical reservation systems contain a great deal of information about the facilities available in hotels; for example swimming pools, coffee shop, fax machines, laundry and so on, often fifty more features. In an on-line booking environment, both the selection of features to present, and the style of presentation should be different depending on whether the booking being made was for a business trip or a holiday trip.

In short, the vast amount of information inside computer systems is held in a codified manner which must be manipulated before presentation to a human user. Computer interaction is likely to much more effective and useful if that manipulation tailors the presentation precisely to the needs of the user.

1.4 Acknowledgements

The author gratefully acknowledges the generosity of his employer, British Airways Plc., both in allowing him the time to complete this work and in providing financial support.

2. Approach to Research

The approach to the research during this first year has been to develop four parallel areas of investigation, each of which is further discussed in chapters 3 to 6.

- Background reading in Linguistics, especially style in language
- Background reading in Computational Linguistics, concentrating on Natural Language Generation
- Background reading on Artificial Neural Networks, especially connectionist approaches to Natural Language Processing
- An investigation into the development environments and tools available for Computational Linguistics and Artificial Neural Networks

Linguistics is an enormous field, and after reading introductory texts such as Crystal (1985) and Quirk (1968), further reading was undertaken only by reference from work in Computational Linguistics.

Computational Linguistics itself is a large field, although much of the work is of relevance to this research. A subscription was obtained to the main journal, “Computational Linguistics”, published by the Association for Computational Linguistics, and extensive use has been made of the Computational Linguistics pre-print archive maintained at the Los Alamos National Laboratories (http://xxx.lanl.gov/cmp-lg/). Reading in the specialised field of NLG has been extensive and is discussed in detail in section 4.2.

Introductory texts were also used in understanding Artificial Neural Networks, for example Freeman & Skapura (1991), although work involving natural language processing, and especially those few papers concerning language generation were studied in greater detail.

Finally, to determine the appropriate development environment, an overview was obtained from general Artificial Intelligence texts, for example, Cohen & Feigenbaum (1982) and Luger & Stubblefield (1989). Careful note was also taken of the implementation descriptions given in all of the papers read to arrive at a final decision. This was then validated by an in-depth evaluation through a practical example of use, which is covered, along with a discussion of the other options, in chapter 6.

Interim reports were produced throughout the year and a PC database of references has been maintained.

3. Linguistics

3.1 Overview

As noted above, linguistics is an enormous field, being the application of the scientific method to all aspects of "language". The field has a long history although much of the earlier work was descriptive, recording in detail what language was and how it was used. This lead inevitably to comparative linguistics and the grouping of languages into families and ultimately into an evolutionary "tree" of languages. It is only really in the last thirty years that theoretical linguistics has grown in importance; attempting to explain not simply *what* language is like, but *why* it is like it is. Hypotheses have been formulated and attempts made to test them, often by careful study of how languages are learnt (both Noam Chomsky and Steven Pinker, leading researchers in the field are primarily interested in language acquisition). It is fair to say however, that much still remains at issue in linguistics as a whole.

For our purpose, there are two aspects of linguistics that are relevant - the development of grammars and the study of style.

3.2 Grammars

It must first be understood that what a linguist typically means by a grammar is *not* the prescriptive lists of rules taught in schools (e.g. "Do not split infinitives"). A linguistic grammar is simply a framework for describing what constitutes an syntactically correct utterance in a particular language. They are of interest to us because it is precisely this codification of the infinite possibilities of a language into a manageable set of statements and propositions that makes natural language processing possible.

There are a wide variety of formalisms for grammars and some of them are covered in more detail below, especially in reference to how stylistic features can be represented. As will be seen in section 4.3.2, the grammatical knowledge of the majority of NLG systems has been embodied in a grammatical formalism, rather than being embedded directly in the program. This is analogous to the definition of computer languages by a Meta-language. The grammar is then "interpreted" by the system during the generation process. This has the obvious advantage that such knowledge is modularised and hence can be modified without rewriting code.

3.2.1 Generative Grammars

A Generative Grammar describes the ordering and constituents of phrases in a particular language by a series of rules relating to linguistic categories. For example, S = NP + VP denotes that a sentence, S, may be composed of a Noun Phrase, NP, followed by a Verb Phrase, VP. Categories may be broken down further, as in NP = ART + N or NP = N, here providing a choice of noun

phrase with or without ART, an article (“the”, “a”, etc.) Typically, N will be a terminal symbol, which can be matched by suitably marked items from the lexicon.

Clearly such grammars can be used for both parsing and generation. In parsing, words are replaced by their terminal symbols; adjacent symbols that match the right hand side of a rule are replaced by the left hand side and so on until the topmost symbol is all that remains. For generation, the topmost symbol is replaced by an appropriate expansion; these are applied repeatedly until only terminal symbols remain, which can be replaced by words from the lexicon. Such systems can be implemented quite easily by recursive transition networks.

The earliest generation systems used this mechanism, and Klein (1965) was the first to point out that these rules could be used to control the style of the generated text. Klein used a crude mechanism of applying weights to each of the possible options for a given expansions. An expansion was chosen randomly in accordance with the weightings. A different set of weightings would produce a different style of output.

Klein only demonstrated the concept, no further work appears to have been done trying to identify what weightings produced a particular output style. This was probably because generative grammars exhibit major problems with more realistic language use and subsequent systems began using feature based formalisms instead. Klein did however, note that weightings could be obtained by analysing input texts and that by using these weightings in generation, output could be produced that mirrored the style of the input.

The major problem with generative grammars is that they only relate linguistic categories to their position within phrases. To include concepts such as number and gender agreement, new rules and categories must be added, leading to an explosive growth in number and complexity of rules. As an alternative, Augmented Transition Networks (ATNs) could be used to implement these extra features, but at the cost of incorporating grammatical knowledge into the program code. To use such a grammar to control output style also meant that *every* point at which a choice of expansions was possible had to be given a weighting further embedding linguistic information into the program code.

3.2.2 Feature Based Grammars

Building on the earlier work using ATNs a series of new formalisms were developed using the concept of features. These are based around “feature sets” (although the nomenclature varies between formalisms), which associate a particular “string” of a language with a set of feature : value pairs. The value of a feature may itself be another feature set and values may be shared between any level. Typically the only operation permissible on feature sets is Unification, (hence the alternative classification as “unification based grammars”). The unification of two feature set is the minimum set of which both original sets are subsets. Where a feature : value pair is common to both sets then the values must be the same. In addition, there are combinatory rules that show how

strings may be concatenated and how feature structures are related. For example, consider the rule below (after Shieber):

```
X0 → X1 X2
     <X0 category> = S
     <X1 category> = NP
     <X2 category> = VP
     <X0 head> = <X2 head>
     <X0 head subject > = <X1 head>
```

This is analogous to the first of the generative rules given in the previous section, along with the extra requirements that the head of the verb phrase be the same as the head of the sentence and also that the subject of the sentence is the head of the noun phrase[1].

One of the simplest such formalisms is PATR-II in which unification is the only operation allowed on feature sets and concatenation is the only operation allowed on strings. Other formalisms exist, introducing additional concepts or for specific purposes. For example Definite Clause Grammar is functionally almost identical to PATR-II but the notation is such that it can be very easily implemented in the PROLOG language. Functional Unification Grammar (FUG) extends PATR-II by adding ANY values and allowing disjunction (alternatives within values) amongst other changes. A good introduction to several unification grammars can be found in Shieber (1986), although in this work each formalism is treated separately - a comparison table of features would have been useful, along with more discussion of the benefits and disbenefits of each. Implementation details of DCG and PATR-II can be found in Gazdar and Mellish (1989). The use of FUG in generation is evaluated in McKeown and Ellahad (1991).

Note that the features in the set need not be established linguistic fact, they can be anything the designer of the grammar chooses. It is this ability that may allow the concept of style to be incorporated into a grammar. Theoretically, we could extend the feature set to include a feature for each stylistic dimension that we choose; the value of these features being the “setting” of that stylistic dimension. Thus we can use exactly the same mechanism that can ensure, say, agreement in number between nouns and verbs to ensure that the STATUS of the generated text is “Superior”. This is further discussed in section 3.3.1.

3.2.3 Other Formalisms

One interesting system which does not use a grammar formalism such as those above is GEMS, discussed in Parisi and Giorgi (1988). In this case all grammatical knowledge is contained within the lexicon. A text is generated by choosing the required words from the lexicon. Associated with each word are placement instructions, such as “place any nominative before me”, “place any preposition after me”. Placement instructions also have priorities associated with them and hence by applying the instructions in priority order the correct word order emerges for the final sentence. The authors note that they have chosen these placement instructions so as to produce “the most neutral style”,

hence implying that a different set of placement instructions would lead to a different style of presentation. Although this approach has been successful in the limited domain to which it was applied, this author believes that there are more aspects to style than the ordering of words and that, in common with most other work in NLG, feature based grammars provide the greatest flexibility.

3.3 Attempts to Define Style in Linguistics

There is no single, widely accepted model to describe stylistic variation in language. We consider here some previous work attempting to create frameworks within which style can be addressed.

3.3.1 "Dimensions of Situational Constraint"

Crystal and Davy(1969) in a work described, quite rightly, as "ambitious" by DiMarco & Hirst, set out to define very thorough procedural framework for the analysis of style in the English Language. They provide a starting point by defining eight "Dimensions of Situational Constraint", which are listed with definitions and examples in Table 1.

[1] 'Head' (or 'governor') in the grammatical sense means the principal item of the phrase.

Dimension	Definition	Examples
Individuality	Identifying features of an individual, largely invariable over time.	William Shakespeare Kermit the Frog
Dialect	Distinctive features of geographic regional or, less often, the speaker's class	West Country BBC English
Time	In the sense of diachronic linguistics, i.e. those features indicative of a particular era.	Mediaeval Contemporary
Discourse	Speech or writing, monologue or dialogue	Monologue / Dialogue speech Writing
Province	Occupation or activity being engaged in.	Legal / Medical / Scientific
Status	Relative social standing of participants	Superior / Inferior/ Equal
Modality	The chosen form of expression, within the province or status and different within each.	E.g. for Scientific Province:- Lecture / Report/ Textbook
Singularity	A deliberately introduced (temporary) idiosyncrasy for stylistic effect, e.g. contrast	Any of the above when used "out of context"

• Table 1 - Dimensions of Situational Constraint

Perhaps the main failing of the work is that the analysis framework is laid out, but not backed by worked through examples. The authors suggest that particular linguistic features regarded as stylistically significant be assigned to each of their dimensions, thus language uses in different styles can be described, compared and contrasted in a systematic way. They then examine a whole range of language uses, pointing out what they consider stylistically significant features but fail to complete their analysis by actually assigning features to categories.

3.3.2 Systemic Functional Linguistics

Halliday's work on Systemic Functional Linguistics (Eggins 1994) includes the concept of "Register Theory" (discussed with reference to generation in Bateman & Hovy 1992). This model proposes that language variation can be related to the situation and the audience, expressed by three "registers":

- Field - The subject matter. (akin to PROVINCE in Crystal & Davy)
- Mode - The type of communication (a combination of Crystal & Davy's MODALITY and DISCOURSE)

- Tenor - The social relationships of the participants (akin to Crystal & Davy's STATUS).

Some of the other of Crystal & Davy's dimensions are incorporated into the theory as "Dialectal Variation".

Thus there would seem to be a great deal of commonality between the two models, although there has been much more work subsequently on Register Theory. Indeed, Bateman & Hovy suggest that the theory can be used to compare language generators and theories that would otherwise appear incommensurate, although they do not expand on this.

3.3.3 Other Dimensions

Crystal and Davy do admit that their work does not provide a comprehensive set of dimensions. In Sharples(1985), we see another dimension put forward, that of Maturity. Although not explicitly introduced as a dimension of style (the work is actually a theory of computer assisted creative writing for children) it nevertheless introduces a contrast that is not clearly covered by either of the models discussed above. Indeed, the author gives an explicit list of linguistic features to compare maturity and immaturity in language. These are shown in Table 2.

Mature Features	Immature features
- Word/phrase level - unusual choice of words - abstract nouns - metaphor and simile - multiple modifiers - reflective commentary - inversion - Sentence level - cataphora - ellipsis - apposition - co-ordination by complex relation or adverbial phrase - Section level - changing levels of detail in description - structured forms of text	- Word/phrase level - speech idioms - repetition - ambiguity through lack of pronouns - unspecific words - Sentence level - extensive use of "then" - Section level - arbitrary order - single level of description - exponential growth in pace - chain and tangled forms

• Table 2 - Maturity Dimension

Intentionally or otherwise, for this stylistic dimension, Sharples has completed the analysis suggested by Crystal & Davy, and introduced an aspect of style not apparently covered by either of the models given above.

4. Natural Language Processing

4.1 Overview

Natural Language Processing is the application of information technology to the input, understanding, analysis, transmission, storage and generation of human languages. It is a relatively new field and much work remains to be done in virtually every aspect. Despite early optimism about machine translation, it is only very recently that successful commercial products based on NLP techniques have become a reality; for example “Dragon dictate”, a spoken language input system with a vocabulary of 30,000 words; and a variety of tools to assist (but not *replace)* translators.

A comprehensive and up to date overview of the entire field, its current state and future directions is to be found in Cole (1995). An on-line version is available from the Association of Computational Linguistics homepage.

4.2 Natural Language Generation

At its most trivial almost every computer program written does some “Natural Language Generation”, even if it is nothing more than producing the text: “`SYNTAX ERROR`”. More interesting, and more relevant to this discussion is the somewhat harder task of putting together a “natural” utterance, starting from underlying concepts and expressing these in a grammatically, semantically and pragmatically correct fashion. The language is actually “generated”, rather than simply instantiated. There is an implication of some degree of “understanding” of the language and the concepts it represents, rather than language as an arbitrary string of characters.

Most existing NLG systems are built for a specific knowledge domain. They include linguistic knowledge, usually in the form of a grammar and usually address the question of “what to say” as well as “how to say it”. Typically, in response to a user query, a concept is extracted from the domain knowledge base and some deep syntactic representation is chosen. This may be modified in various ways before being turned into a surface level utterance and displayed or spoken to the user.

An NLG system might thus be defined as:

> A system in which domain knowledge is combined with linguistic knowledge to create a grammatically,semantically and pragmatically acceptable utterance.

There exist several good overviews of NLG systems and history, so rather than reproducing what already exists some pointers to existing work are given. A view of generation systems with specific reference to stylistic issues is given in the next section.

Bateman and Hovy (1992) provide a useful framework into which NLG systems can be placed, offering four, increasingly general categories:-

1) "Canned", pre-defined items of text.

2) "Templates", pre-defined text with slots to be filled by appropriate words.

3) "Cascaded", effectively templates within templates, chosen and filled out as required.

4) "Features", in which individual aspects of the desired text are controlled.

They then go on to discuss various NLG systems with reference to this framework. Clearly, systems sophisticated enough to display variant style are likely to fall into the fourth category. Another useful overview of the field can be found in McKeown and Swartout (1991); this is more detailed but less well organised. Most general Artificial Intelligent reference works contain chapters on Natural Language Processing, often considering generation as well, for example, Luger & Stubblefield (1989) and Cohen & Feigenbaum (1982).

Gazdar and Mellish (1989) provide a good tutorial introduction to the implementation issues involved in NLP, with some, largely incidental, discussion of NLG. Finally, an interesting view of whether the various architectures used in Natural Language Generation can be said to be converging can be found in Reiter (1994). Reiter suggests that the majority of NLG systems have identifiable components covering content determination, sentence planning, surface generation and morphology, but that no system has been able to cleanly implement these components as individual modules; much interconnection is usually required. This reinforces the authors own view that it will not be possible to produce *within* an NLG system a "style" module as style is determined by choices made within all of the components.

4.3 Style In Natural Language Generation

4.3.1 Previous Work in Style

Much of the early work in NLG was actually for the purpose of checking a) the completeness, and b) the validity of generative grammars (Section 3.2.1); the actual language produced was almost a side effect. See for example Friedman (1969) and Friedman (1971). Checking the completeness of a generative grammar is virtually identical to the process of checking a formal specification of a programming language; indeed Friedman used a modification of Backus Naur From (BNF) to specify her grammar. Checks are made for basic "syntax errors" in the specification, unintentional infinite recursion, incomplete specifications and contradictions between production rules. The actual generation part of the systems were simply random, their only aim being to allow the user to see if the generated language was indeed grammatically correct and thus that the underlying generative grammar was a true model of the language. Where such generation was "directed", it was only in the sense that it was directed towards grammatical constructions that were of particular interest.

It was pointed out even before the work above that when faced with choices in a generative grammar, the choices made by an NLG system determine at least to some extent the resulting "style" of the output, see for example Klein (1965). Not much attention appears to have been paid to this however, perhaps because the observation was presented as a linguistic finding rather than something related to computer language generation.

The majority of work in NLG to date has been in the area of "explanation" in Expert Systems. In many cases it is not sufficient for such a system to produce a result, it must also justify that result by explaining the reasoning that lead to it. To be useful to the user this explanation needs to be produced in an easily understandable form, i.e. a natural language, as opposed to some computer representation of reasoning. Although much has been achieved towards these aims the issue of style, particularly variant style never really arose. This may be because most expert systems are used by those who already have some knowledge in the domain in which the system works and hence can overcome any shortcomings in the style of the explanation.

Only in the mid-eighties did research begin to consider variations in the output to take account of the user's own beliefs and requirements, for example the work of Paris (1993) and Hovy (1985) discussed below.

It is only very recently that style has been addressed for its own sake in an NLG system. DiMarco and Hirst (1993) have developed a system called "Stylistique" which can both parse and generate text, identifying or creating a particular style as required. The system at present considers style in three dimensions, Clarity - Obscurity, Concreteness - Abstraction, Staticness - Dynamicness; each of which is categorised into three levels. As with most earlier work, extensive examples are drawn from real texts to back up the linguistic choices that have been made within each of the dimensions.

4.3.2 Style in Existing NLG Systems

In this section a number of existing NLG systems are briefly examined. The architecture and environment of each is described and then an analysis of the style of the resulting output and consideration given to whether this approach allows for easy generation of variant styles. Presented first are two generation systems that were created outside of the Expert System explanation world, and then for comparison, three other systems that explicitly address some aspect of style.

4.3.2.1 The *TEXT* System

McKeown (1985). This system works in the domain of a naval database and is capable of answering a limited set of questions about the contents, and importantly, the structure of and differences between entities in a database. The questions comprise define <entity>, what is known about <entity>, what is the difference between <entity1> and <entity2>. The system is based around two techniques; templates, effectively "scripts" of rhetorical techniques, and focus constraints, methods of determining when and how to introduce new subjects.

The generation portion of the system consists of a strategic component that associates one or more templates with a particular query. Templates are represented by augmented transition networks. By considering the type of knowledge available to it in the knowledge base one of these templates is chosen and the information inserted. This template is then passed to a tactical component which uses a functional grammar to generate English text.

The use of templates and focus constraints is backed up by an extensive theoretical framework based on linguistic examination of existing texts.

What is the relevance of these two techniques to developing variant styles? Any multi-sentence utterance needs to be coherent and to “flow” correctly, and focus constraints appear to be a good mechanism to achieve this. This is really a pre-requisite for any generation system and so is not directly involved in producing variant style.

The use of templates or schemata however is potentially much more interesting. As the author herself hints, a different schema may a different impact on the recipient, be more easily understood, or perhaps used to emphasise a different aspect. Clearly though, schemata of this nature are applicable only at the level of which rhetorical techniques to use and in what order (e.g. identification *then* constituency *then* equivalence). Phrase and lexical choice must be addressed elsewhere. In TEXT this was done by the tactical component, in a "fixed" fashion, tailored to knowledge domain in question. In combination with other strategies for lexical and phrasal choice this could be applied to the generation variable style. The TEXT system does however suffer from the same problems as Generative grammars in that linguistic (and hence stylistic) information is embodied in the ATN, rather than in the grammar.

4.3.2.2 Newspaper Report Generation

In Danlos (1987), the author has constructed a generation system for a limited domain on a provable linguistic basis; i.e. the majority of "decisions" made in the production of the text can be justified by reference to demonstrable linguistic facts. The chosen domain is reports of terrorist crimes as they might appear in a newspaper, and short texts can be generated in either French or English.

The underlying linguistic basis is that of a “discourse grammar”. For the domain in question the main goal is to communicate the fact that a terrorist act has occurred and that that act has had certain consequences. Each of these concepts can be expressed in a simple sentence in which the verb is active or passive and may or may not have an agent (someone doing the action). Additionally, it may be possible to conjoin two sentences expressing the action and the result, using pronouns as appropriate. We can also reverse the order of the conjunction giving "<ACTION> causes <RESULT>" or "<RESULT> caused by <ACTION>". This gives 36 combinations of 1 or 2 sentences expressing the act and its consequences, of which 21 can be shown to be unacceptable to a normal English speaker.

In use the user fills in a form with the details of the incident and then one of the fifteen acceptable constructions is chosen. A key part of the hypothesis is that this choice of construction is at least partly constrained by the choice of vocabulary and vice versa. For example, it is not acceptable English to use the verb murder *after the act*, e.g. "John stabbed Mary. He murdered her." Note however, that reversing the order to <RESULT><ACTION> *is* acceptable. "John murdered Mary. He stabbed her." Thus choice of order, discourse structure and vocabulary are all interdependent.

Alternative versions of the same text can also be provided, although this is done simply by choosing another of the acceptable constructions. No attempt is made to choose based on a required style, nor are the various constructions evaluated stylistically.

It is interesting to note that the author has had to include a great deal of linguistic knowledge of all types in the system; syntactic, semantic, morphological and pragmatic. Indeed, it is part of her thesis that it is not possible to regard each of these areas separately, processing each one in a linear fashion. The choices made at one level affect the available options at other levels. This would suggest that to produce text in a different style (say, that reporting the facts to a court) would need rework to all parts of the system and that the output “style” is not contained in any particular part of the system.

4.3.2.3 The *TAILOR* System

Paris (1993). The TAILOR system was constructed with the aim of investigating how output should be tailored dependent on the users level of knowledge and experience. The system maintains a user model which contains the basic concepts understood by the user, and items of specific local expertise. A typical task would be to describe a telephone to someone who already understands electricity and has already been told about the operation of a loud speaker. The knowledge base for the system is a pure hierarchical network of electrical equipment and its component parts, overlaid with a generalisation hierarchy showing for example that a telephone earpiece is an instance of a loudspeaker.

By examining the user model and the hierarchy information the system is able to select what information to provide to the user, i.e. determining “what to say”. More interesting stylistically is the use of the user model to also determine the schema used to organise the information. Two schema are used, Constituency and Process Trace, both of which are closely related to McKeown’s schemata discussed previously. The constituency schema describes equipment in terms of its sub-parts, while the process trace describes the equipment in terms of the various steps in the process it performs. The user model determines both which schema to use initially and whether to switch schema during generation - for example if during generation of the constituency schema a sub part is encountered of which the user does not have local expertise then the process schema is invoked to describe the function of that part.

The knowledge base is implemented as a series of “frames”; the schema is implemented as an augmented transition network and a Functional Unification Grammar (section 3.2.2) is used by the surface generator.

In common with TEXT, the stylistic analysis largely affects the choice and organisation of phrases, there is little or no reference to the user model when choosing words from the lexicon or generating the surface form of the utterance. In order to do this more information would be needed in the user model. The model contains only information about basic concepts understood by the user and their local expertise of particular components, realistically this information only allows tailoring at the phrasal level. To tailor at the word level needs additional information such as the reading age of the user; the amount of detail they require; the purpose for which they want the information and so on. Nevertheless, user modelling appears to be a powerful technique in determining what to say and how to organise it. It does not however, address all the needs of a variant style system, and in common with TEXT, has linguistic knowledge scattered throughout its component parts.

4.3.2.4 The *PAULINE* System

Hovy (1987). PAULINE is a particularly interesting system from a stylistic point of view as Hovy has explicitly attempted to produce a variant output text to achieve particular conversational goals. PAULINE’s user model is much broader than that of TAILOR, and includes consideration of the conversational atmosphere (e.g. time available); the speaker and hearer (e.g. knowledge, interest and opinion of the topic); and their relationship (e.g. depth of acquaintance, relative status).

PAULINE can have various conversational goals such as affecting the hearers knowledge or opinion of the topic, affecting the relationship or interpersonal distance and so on. These conversational goals are mapped on to a series of rhetorical goals, such as simplicity, formality, forcefulness, detail and so on. These rhetorical goals are in turn mapped onto specific strategies for sentence organisation and word choice. Hovy argues for the additional level of rhetorical goals by reference to the fact that a particular linguistic strategy can contribute to more than one conversational goal. Whenever a choice has to be made during the generation process it is always made by reference to the currently active set of rhetorical goals, and hence, unlike most other NLG systems, style is considered at every level of generation.

Despite the apparent success of the system in achieving variant output styles it has been criticised for its arbitrary mappings between goals and strategies and the distribution of linguistic knowledge throughout the system, making it very hard to modify. The former criticism arises perhaps from linguistic snobbery in that Hovy does not back up each of his goal : strategy mappings by reference to a body of linguistic examples; instead he has relied upon the linguistic intuition of a competent language user (himself) to derive the mappings. The second point is perhaps more serious, suggesting as that it is difficult to “modularise” stylistic issues - since we wish to apply them at every level of generation, it may be necessary to distribute stylistic knowledge throughout the system. This criticism of course, has applied to all the systems discussed in this section.

4.3.2.5 The *IMP* System

Jameson (1987). Like TEXT and TAILOR, the IMP system considers “what to say”, more than “how to say it”, but is still interesting stylistically as the resulting output does clearly vary in style, based on the bias that the speaker has towards the subject in question. The domain implemented by IMP is that of a personnel selection interview, with IMP assuming the role of interviewee. The knowledge base consists of facts about the interviewee and his personal circumstances. IMP is actually designed to be domain independent so the actual phrases associated with each fact are supplied “pre-packaged” - IMPs main role is to decide which of the facts to volunteer.

IMP can be given a particular bias in its choice of facts, along with an evaluation of which facts reinforce a positive view and which a negative view. Depending on the question asked and the answer IMP will volunteer more or less information. It does this by calculating a “cost” for revealing each fact, calculated on the basis of the reduction in uncertainty in the mind of the interviewer. For example, assume that IMP believes that the interviewer is ideally looking for someone in a stable family relationship and is asked the question “are you married?” If IMP’s knowledge base contains the fact that he is living with a girlfriend, the calculated “cost” of revealing this will be very high and instead will simply volunteer the answer “No”. The interviewer thus is left with a degree of uncertainty as several possible states are left open. Conversely, if IMP had a different set of facts it may have replied “yes, I am married with two small children”. The additional information resolving any uncertainty in the mind of the interviewer. Thus “bias” is modelled as the degree of certainty in the hearer about positive aspects and the degree of uncertainty about negative aspects.

Although in implementation, IMP is hardly more than a template generation system, the modelling of bias and the concept of the “cost” of an utterance are devices that may have stylistic implications. Again, however, they are unlikely to be all that is sufficient to generate variant style.

5. Connectionist Natural Language Processing

5.1 Overview

According to Hervé Bourlard and Nelson Morgan, writing in Cole (1995), "Generally speaking, connectionist language modelling and NLP has thus far played a relatively small role in large or difficult tasks". The author's own reading would suggest that this is indeed true. Some work has been done showing that ANN's can carry out many of the language tasks that would normally be done by Recursive Transition Networks (RTNs), for example Allen (1992) demonstrated several simple generation and recognition tasks using a network called CLUES. The advantage of ANNs in this case obviously being that the network will learn from its training set and there is no need to determine in advance the set of RTN states and transitions. The work containing Allen (1992), Sharkey (1992) is a useful source of current research into Connectionist NLP, although most work is concentrated on parsing & recognition tasks rather than generation.

ANNs have also been combined with Hidden Markov Models (HMM) for use in speech and handwriting recognition. HMMs are probability chains based on the probability of a particular input state representing a certain output, and the transition probability to the next state. Originally, HMMs used static probabilities but some success has been achieved by Boulard & Morgan (1993) in using ANN to generate the probabilities, to considerably improve performance.

5.2 Connectionist Natural Language Generation

As noted above, connectionist techniques have so far had little impact on NLP in general and this is also true of NLG. The generation component of Allen (1992) above was simply generation of sequences such as "abbaba" as might be produced by a simple RTN. A more realistic example of generation is to be found in Kukich (1987), who describes some interesting preliminary results from a neural network trained to implement several stages of a process to convert raw stock market information to English commentary such as "closed slightly higher". Input was coded as six types of categorised information, e.g. direction up / down; degree great / moderate / small / unchanged; closing price yes / no. The first stage was to select a series of morphemes[2] based on the input, the second stage was put these morphemes into a syntactically correct order. Each stage was accomplished by a separate application of the neural network. The training set consisted of 113 input : phrase pairs. Most interestingly from the stylistic point of view was the fact that synonyms were included in the training set, for example the phrases "staged a moderate advance" and "posted a modest gain" were both paired with the same input pattern. To cater for this variation two additional inputs were added whose value was random, thus effectively producing a random choice of phrases in the output. Stylistically it might have been more interesting if these additional inputs were used to represent a coding of some stylistic dimension such as bias for or against the result in question.

6. Development Environments

6.1 Overview

If the author's research was ever to get beyond the theoretical then a development environment would be required. Other reading had seemed to indicate that the two most common "AI" languages were LISP and PROLOG, although some work had been done in more general purpose languages such as 'C', and a "best of all worlds" system was available in Sussex POPLOG.

The author obtained LISP and PROLOG interpreters for the PC and experimented with both languages, finding a preference for PROLOG as being clearer to read and having better facilities for the implementation of grammars. Bratko (1990) proved to be an excellent tutorial for AI programming tasks, including the use of grammars such as DCG in PROLOG.

The author also has great familiarity with 'C' and with the various visual programming tools available for the PC, such as Visual Basic. As will be seen below, these did play a part although the author's experience suggests that they are not ideal for AI, being entirely procedural and lacking built in support for grammars.

One option that seemed to offer the simultaneous use of all the above options was Sussex POPLOG. This is an integrated development environment comprising LISP, PROLOG and POP-11 compilers, all of which had the ability to call modules in any of the other languages, or to link to 'C' functions.

6.2 Considerations for Choice

The choice of development environment was constrained by the following requirements:-

- Cost, ideally freely available without restriction for academic use
- Portability, ideally able to run both on the author's PC and on the university Sun machines
- Widely used and supported, so that help would available

Although initially attracted to the POPLOG environment with its enormous flexibility; its lack of a PC version was a serious handicap. In addition, the author believes that such flexibility allows problems to become "hidden". If a particular aspect is difficult to implement in say, PROLOG, then the developer is free to code this portion in 'C'. This may appear the easy option but is likely to lead to problems later when modification or restructuring is needed. The difficulty of implementation may be a sign that the problem formulation is incorrect and effort would better be spent rethinking the problem so that it can be solved in the original language.

[2] "Part of a word", for example 'close' and '-ed' are morphemes of the word 'closed'.

For these reasons, PROLOG was chosen. Section 6.3 describes the actual implementation used.

6.3 A Practical Example

6.3.1 Summary

The majority of current work on Text Generation is carried out using feature based, unification grammars and hence an understanding of them is needed by anyone intending to work in this field. Additionally, much of the practical work is carried out using the AI language PROLOG. Although Shieber's book "An Introduction to Unification Based Approaches to Grammar" is an excellent theoretical discussion; and Gazdar & Mellish (hereafter G&M) in their book "Natural Language Processing in PROLOG" give some code examples; there is no substitute for actually developing and using such a grammar in such a programming language from scratch.

A Grammar has been written for Polish prepositional phrases using the PATR II unification formalism. The grammar has been successfully used both by the recogniser provided by G&M and by a custom written generator. In addition, the formalism has been extended in some non-trivial ways, and the recogniser and generator modified accordingly.

Through this work the author believes that a good basic understanding in the production and use of unification grammars has been demonstrated and sufficient knowledge of PROLOG has been acquired to continue research in this area.

An example of use of the program is shown in the appendix, 9.1.

6.3.2 Unification

Most introductory texts on unification tend to begin with a theoretical discussion of the actual process of unification, however the author feels that it is more useful to first consider how unification is actually used in a practical grammar.

Considering the lexicon first, each word has a set of features that can take particular values; the values may themselves may be sets of feature / value pairs. So for example, the word 'obiad' (lunch) might have a feature 'category' with the value 'noun'; also a feature 'agreement' which has as its value three further features, 'person', 'gender' and 'animate' which have values 'third', 'masculine' and 'no' respectively.

```
obiad ---> category = noun
           agreement = ( person = third     )
                       ( gender = masculine )
                       ( animate = no       )
```

Not every word need have every feature specified, as will be seen below, the value of 'animate' is irrelevant to feminine and neuter nouns and for nouns in these genders it would not be given.

We also have a set of syntactic rules. These show how a syntactic category can be formed out of other categories, and the relationships that must hold between the feature sets of those categories. Relationships can be any of the following types:-

- An explicit value is given ("category = noun")
- Values between two feature sets must unify ("noun case = adjective case")
- Entire feature sets, or feature subsets must unify ("noun agreement = adjective agreement")

It is important to realise that we do not necessarily have to unify all of the feature of the two sets - the syntactic rules specify which parts of the feature sets must unify.

We must now explain exactly what we mean by "unify" - firstly, it does NOT mean "be exactly the same as". Unification is a process which succeeds if the two feature sets can be combined into a new set. Features that appear in only one of the sets are simply "copied" into the new set but features that are specified in both MUST match. The combined feature set can then be used in other, "higher", unifications if required.

The matching process for values, in its pure form is that of an exact match, although, as discussed below, in a practical system, the matching process can be whatever we define it to be. For example we could define that a match occurs if the value of one feature set is the same as any member of a list of values in the other set.

In summary, we have feature sets attached to items, and syntactic rules that specify how items can relate and which parts of their respective feature sets must unify. Unification succeeds if a combined set of features can be constructed that includes all of the specified features of the two sets without a contradiction in their values.

6.3.3 The PATR II Formalism and its Implementation

PATR II is a formalism for writing unification grammars. By using some programming tricks it is written in a "natural" fashion that is also directly useable by PROLOG. Thus, words are specified as follows:-

```
W ord obiad :-
    W :: cat === n,
    W :: agr :: pers === third,
    W :: agr :: gnd === masc,
    W :: agr :: anim === no.
```

W is a PROLOG variable, "ord" is a dummy infix operator, the word itself is a PROLOG atom "obiad" and ":-" is part of the PROLOG syntax to specify clauses. The second line means that the variable W has a feature 'cat' whose value is 'n', the next three lines specify the feature 'agr' which itself three sub-features, 'pers', 'gnd' and 'anim'. This specifies the feature set discussed above.

The operators "::" and "===" are defined such that the variable W becomes instantiated as a list of lists:-

```
[cat :: n, agr :: [pers :: third, gnd :: masc, anim :: no]]
```

And it is this representation that is used internally.

Syntactic rules are specified in a similar fashion using the dummy operator "ule":-

```
R ule AP ---> [SA, SN] :-
    AP::cat === ap,
    SA::cat === sa,
    SN::cat === sn,
    SA::agr === SN::agr,
    AP::case === SN::agr::case.
```

This is a rule for adjective phrases that must consist of a simple adjective followed by a simple noun, as shown in the first line. As in the definition for word above, R is a variable and "ule" a dummy operator. The next three lines specify the categories of each of the feature sets in the first line. The fifth line is the most interesting as it specifies that the 'agr' features of both feature sets must unify - remember that this feature has sub-features and hence the values of each must match for the unification to succeed.

The final line simply sets the 'case' feature of the adjective phrase to be the same as its sub-components. This is for use in other rules that may themselves use AP as a component. If this rule is successfully applied then AP becomes instantiated as the feature set:-

```
AP :: cat === ap,
AP :: case === (whatever the case of the SN is).
```

The combined feature sets of SN::agr and SA::agr have effectively been "discarded" once the unification was proved successful - for the purpose of this grammar their values are not relevant for the way we use AP in other rules. Had we required all of the agreement values to be present in the feature set of AP we could simply amend the last line to be:-

```
AP::agr === SN::agr.
```

6.3.3.1 Extensions to PATR II

Two extensions were made to the PATR II formalism, and both the recogniser and the generator were modified to incorporate them. The first was a new operator, '++' to indicate the relationship between the "basic" form of a word and the suffix, needed to indicate case, gender and so on. Both word and suffix are treated as separate lexical items and only for the purposes of display are they joined together; the '++' operator indicating where this should be done.

Secondly, in an attempt to improve the brevity of the lexicon, another operator 'or' was defined. Using this, a feature could be defined as having one of several values, each separated by 'or'. This allowed near identical word definitions to be combined, for example, neuter noun endings for the locative and accusative cases are the same and thus the value of the 'case' feature can be 'loc or

acc'. Without the new operator both cases would have to be specified separately, each needing six lines of code.

6.3.4 A Grammar for Polish Prepositional Phrases

6.3.4.1 Prepositional Phrases

Prepositions are difficult to define other than by giving a list of examples - 'into', 'above', 'during', 'through', 'opposite' and 'towards' are all prepositions. A prepositional phrase is nothing more than a phrase that starts with a preposition, such as "after lunch", "next to the big park", or "beyond the blue horizon".

6.3.4.2 Polish Prepositional Phrases

Polish was chosen for this exercise as the author has some small familiarity with it and had not seen any prior examples of grammars for Polish. In addition, Polish comes from a different language family than English and would be an interesting test of the expressive power of the formalism. The main differences between English and Polish, especially relevant to prepositional phrases are:-

- Nouns have gender (masculine, feminine or neuter) as well as number and person. Adjectives must agree in gender and number with the noun.
- Nouns also have case, There are seven cases, marked by word endings which differ for each gender and between singular and plural. There are also examples of irregular endings for some words. In addition, the endings of masculine words in some cases depend on whether the noun refers to an object which is typically animate or typically inanimate. Adjectives and nouns must have the same case.
- In most situations the case of the noun is determined by the role it plays in the sentence, subjects are in the nominative, objects in the accusative, negatives in the genitive and so on. An exception to this is prepositional phrases in which the case is determined by which preposition is being used.
- Polish has no definite or indefinite articles. 'Pies' could mean "dog", "a dog" or "the dog" depending on the context.

6.3.5 Implementation Notes

Originally, the system was developed in parallel for Sussex Poplog on the Sun workstations and on SWI-Prolog for Windows. Mid-way through the work it was decided that the effort to keep the development in step on both platforms was too great - the area of greatest difference being in string handling which is where the bulk of work was done.

Work was thus concentrated on SWI-Prolog for Windows. This is a Freeware port of the publicly available SWI Prolog which uses none of the windows features except for the larger memory than is available under DOS.

All areas of the code that are believed to be implementation specific have been marked as such. The bulk of the code will run unchanged between the platforms, the main area that needs to be re-written for Sussex POPLOG is the “pretty_print” function which turns a parse tree into a neatly printed string.

Polish makes heavy use of accented characters not available in the normal ASCII character set. Fortunately each character normally only has one accented variant so we have adopted the convention that any character followed by an underscore represents the accented form of the character. For presentation purposes a Visual Basic program was written which would take the final sentence and display the correct characters using a True Type Eastern European font, thus rendering “ksia_ke_” (books) as “&♦♓◷&🡻” for both screen and printer.

6.3.5.1 The Text Generator

The function “expand” has been written which, when given a list of feature sets will seek to expand each of those sets in turn according to the rules permitted in the syntax part of the grammar, using the words from the lexical portion of the grammar. A partial parse tree is returned in which the nodes are marked with the syntactic categories but the leaves (i.e. the words themselves) are unmarked. (Such marking could be done by the addition of an additional line of code, to call “pathval” to retrieve the word category.) Subsequent calls to the generator will produce other valid expansions.

For demonstration purposes a function “generate” was written which would take a syntactic category, construct a feature set with only the ‘cat’ feature instantiated and then call “expand”, pretty printing the result. Thus we could ask the generator to produce “all simple nouns”, “all prepositional phrases” or whatever. It would be quite simple to expand this so that the feature set included more features, so for example we could ask for “all prepositional phrases that are in the genitive case”.

A typical tree returned by the generator looks like:-

```
"[pp,[blisko,ap,[sa,[dobry ++ ego1],sn,[student ++ a0]]]]"
```

Here, syntactic categories are marked by the convention that any atom that immediately precedes a sublist is the syntactic category of that sublist. Thus “pp” is immediately followed by a sublist which is a prepositional phrase, “ap” is immediately followed by a sublist which is an adjectival phrase and so on. The ‘++’ is one of the extensions to the formalism and indicates that the string to the right is a suffix to the word on the left. The trailing digit specifies how many characters should be taken from the end of the word before adding the suffix.

Routines were written to strip the syntactic categories, flatten the list and print it neatly; the above list would be displayed as:-

```
"blisko dobrego studenta"
```

which translates as “close to the good (male) student”.

6.3.6 Findings

6.3.6.1 Expressive Power

The author is impressed both with the expressive power of the PATR II formalism and PROLOG itself. The PATR formalism is capable of handling the complexities of Polish cases, including the animate / inanimate needs of masculine nouns. The extensions were not actually needed to implement the grammar in full, they were added to make the writing of the lexicon a little easier and mark word endings more neatly. The original version of the grammar and code did in fact work without the additions, both for generation and recognition.

PROLOG itself is well suited for this type of work. By using components already provided for the recogniser, a generator was written in just nine lines of code (the ‘expand’ function). The rest of the code is a demonstration routine to call it, and (over half of the total code) is concerned with formatting the resulting tree as a string. This is a similar ratio to that obtained by G&M in their recogniser and the code to neatly print out the resulting feature set. In a more realistic situation this would suggest that string and character manipulation for the very final stage of output would better be done in a procedural language and called externally.

6.3.6.2 Grammar Writing

Writing grammars is very tedious! Particularly the lexicon. Even this small subset of Polish needed on average four feature :: value pairs to be specified for each word. Use of “macros” as suggested by G&M can reduce this slightly, as can the extension to allow use of the ‘or’ operator. It must also be appreciated that this grammar only contained syntactical and morphological feature information; a more realistic grammar is likely to also require at least semantic information and possibly pragmatic and context information as well.

It is thus clear why recent work has begun to address the problem of automating the production of lexicons through examining corpora. Even so, for the purpose of continuing this research, a hand-crafted grammar will be used. The automated construction of grammars would be a major project in its own right.

In addition, as there is no “syntax checking” of the features and values of the grammar, major problems can occur with very simple misspellings. A spelling error in a value usually results in a failed unification that should have succeeded while a spelling error in a feature name usually results in a unification succeeding that should have failed. Spelling errors in both can be disastrous! One possible solution is to create a feature set including *all* the features and each feature having *all*

values, separated by the ‘or’ operator. Before feature sets are unified they could each be unified with this “master set” - failure of this unification would suggest a spelling error of some kind. Tests of this kind that can be switched on or off as required will be built into future versions.

6.3.6.3 The Meaning of Unification

Theoretically, two structures unify only if those features that are specified in both have identical values. In practice, unification is whatever we define it to be through the ‘unify’ clause in the code. Hence we were able to extend the PATR formalism by adding the ‘unify’ clause that allowed any of the values separated by the ‘or’ operator to unify.

Similarly, we could add an explicit ‘ANY’ value by adding the clause:

```
unify( 'ANY', Atom ):-
    atom( Atom ).
```

In fact, if we wanted to we could include as a value the name and arguments of a PROLOG clause that must be true if the values are to be said to unify.

6.3.6.4 Generation vs. Recognition

The same grammar, without modification of any kind can be used both for generation and for recognition. Indeed this can even be done “at the same time” as the recogniser and generator will co-exist in the same PROLOG instance. In a limited fashion the recogniser can be used as a generator - although it cannot generate arbitrary strings it can “fill in the blanks”, for example filling in the correct word endings given a partially complete prepositional phrase. Similarly the generator can be used for recognition, if nothing else by the simple expedient of generating all possible sentences and checking the candidate sentence against each in turn.

Even given this reversibility it still appears better to write separate generators and recognisers. In the first case above there are some serious limitations in what can be generated, in the second case there is inefficiency. (Even in this limited and non-recursive grammar there are 440 possible prepositional phrases that can generated.)

7. Conclusions

7.1 Overview

Given the background reading, and the practical work carried out, the author has arrived at a number of conclusions. Firstly, we can make an attempt at defining *how* language might vary, i.e. what the dimensions or parameters of style might be. This leads us to offer a definition of "style" for the purpose of developing a system that allows us to vary style. Finally, we can make some decisions as to the tools and techniques that we will use to build and test such a system.

7.2 Dimensions of Stylistic Variation

Given the various models of style discussed in section 3.3, in what ways might we expect stylistic variation to be manifested, and what constraints might a variant style system have imposed upon it?

The first constraint would almost certainly be that our variant style system is restricted to a particular language. Style is intimately tied to linguistic features, and these vary from language to language[3]. The definition of "language" however is quite broad, a system for "English" would encompass both "American English" and "British English" even though there are both grammatical and stylistic differences between them. The system would produce output in either form dependent on the setting of the DIALECT dimension.

No constraint exists however, regarding the knowledge domain in which the language is being produced. A style system should perform equally well when applied to any subject in the target language. Particular stylistic features required by the specific knowledge domain would be taken care of by the setting of the PROVINCE dimension.

In addition to setting the values of Dimensions or Registers, it may also be advantageous to tailor the output at a more detailed level; to allow for the adjustment of specific linguistic features, for example the frequency in the use of the passive voice, the degree of formality in personal names and so on. This is tailoring at the "micro" level of style. This is discussed further in section 7.3.1.

With the increasing prevalence of multiple types of output media we could require the system to incorporate "gestures" (screen pointers) and other non-linguistic features. "Style" in this context may be extended to include such things as the "house style" of a journal, the typographical conventions such as layout and the use of fonts and colour. This is discussed further in section 7.3.3.

It might also be expected that the system as a whole might monitor the style of the input from the user and modify the output style in an appropriate manner. Often the most appropriate style would that which is most effective at communicating the required meaning, which will typically be in an output style that mirrors the input, but this will not always be the case. Taking for example the

dimension of MATURITY from section 7.3.2; in a teaching environment the output may be generated to include slightly more mature features than are present in the input. Similarly, if an input shows characteristics of having a "superior" status, the generated output may have an "inferior" status.

So in summary, an ideal Variant Style System might have the following characteristics:-

- Language specific
- Subject independent
- Offering control over micro-level linguistic features
- Offering control over macro-level stylistic categories
- Able to incorporate other output media
- Able to modify itself based on input style

7.3 A Definition of Style?

For the purpose of our research it is proposed to define a particular style as:

> That set of micro-level language features which characterise language use within a particular macro-level stylistic category.

Micro-level features (more properly micro-level stylistic features) are simply instances of language usage over which the language user has some degree of choice. Thus the use of a verb in the passive form is a micro-level feature as the language user has the alternative of using the active form; the use of the possessive *his* in the sentence *The man found his keys* is **not** optional and thus is not within the definition above.

When a particular set of these micro-level features repeatedly appear in a particular language situation, indeed when that set of features primarily *identifies* that language situation then that set forms a macro-level stylistic category.

Each of these levels will now be considered in more detail, along with brief consideration of style in non-linguistic media.

[3] Hence the title of the Crystal and Davy book is "Investigating *English* Style" (my italics).

7.3.1 Micro Level Language Features

As noted above, these are merely particular features of language, specifically those that are stylistically significant; i.e. those that would differ between macro-level categories. What sort of features might these be? Some will simply be particular features of language that are simply "present" or "not present", like subject - verb inversion, "at the door stood a man". Other features may display greater flexibility, having a number of options; for example the type of personal reference used, as shown in Table 3.

Option	Language Feature	Example
Very Informal	Non-specific noun	"Mate"
Informal	Christian name	"John"
Neutral	Christian / Surname / both	"John Smith"
Formal	Honorific and/or surname	"Sir" / "Mr. Smith"
Very Formal	Honorific + full name	"Mr. John Smith"

• Table 3 - Example of micro level feature

The author believes that in general there will never be more than five realistic options for any particular feature. George Orwell, in the appendix to his novel *Nineteen Eighty Four* (Orwell 1949) describing the language "Newspeak" allows for three levels of adjectives and adverbs - e.g. good / plus-good / double-plus-good. Crystal and Davy also discuss this issue and, albeit in an aside to their main point, consider that there might be up to six categories. Recent work on a stylistic parsing and generation system by DiMarco and Hirst (1993) offered three levels of tuning in their output.

7.3.2 Macro Level Stylistic Categories

We have seen style categorised into three "Registers" in section 3.3.2, eight "Dimensions" in section 3.3.1 and a further category in section 3.3.3 apparently not covered in any of these. Clearly there is as yet no definitive categorisation. It would be unrealistic to expect this research to develop such a paradigm, however what we do aim to achieve is a system that allows variation in more than one Stylistic Category (as discussed in Chapter 4.3, most previous work in Style has concentrated on just one specific register or dimension). Again, it would unrealistic to attempt to model variation in all categories, part of the work will be to identify those categories that are most "useful" in real world situations. For example, although there clearly is a linguistic dimension of TIME (Table 1), it is not generally useful to switch between mediaeval and contemporary modes of speech.

7.3.3 Other Media

Although it is outside the scope of the proposed research a brief discussion of style in non-linguistic media follows.

Computers are becoming more capable, featuring high quality sound output; greater sophistication in the presentation of text; integration with still and moving graphic images, even tactile output through “data gloves”. Some work has already being done in this field, for example, Appelt (1985) developed a planning system that could integrate gestures with language although only recently has worked begun on integrating a wider range of media. Such a wider range of output media has many more degrees of freedom and would thus suggest even more opportunities to demonstrate variation in style. It would be interesting for future work to attempt to define what extra dimensions multimedia offers, and also allow an opportunity to validate the model of micro-level features and macro-level categories proposed for text output.

7.4 Development Environment Conclusions

None of the practical example work done with unification grammars has been ground breaking, although the treatment of suffixes as separate lexical items is unusual. Nevertheless, a good understanding of such grammars and of PROLOG has been gained.

In addition, we now have a useful set of tools with which to investigate style. Recall that features do not have to be merely syntactic, they can describe meanings and stylistic information; for example the micro-level stylistic categories discussed above. Thus we can construct a series of feature sets and, in effect, ask the ‘expand’ routine to generate a sentence with a given meaning, exhibiting a particular stylistic effect. As features can be attached independently to words and rules we will also be able to investigate the stylistic effects produced by choice of word, or by choice of rules.

Note however, that these features work only at the micro-stylistic level - a number of micro-level effects work in combination to produce a macro-level “style”. Something more is needed “on top” of our basic generator to manage what are effectively multiple micro-level stylistic goals. One possible approach would be to assign numeric values to each micro level feature and generating a range of texts. A discriminator such as a neural network could then be used to select the text most like those of the “target” style. This is discussed further in the next section.

8. Research Proposal

Most previous work in style in generated natural language has concentrated on varying style along just one dimension. I propose to investigate the feasibility of a generating text the style of which can vary across multiple dimensions. There are two major components needed for such a system: a grammar of sufficient richness to allow choices in the generation of a text, and a mechanism to determine which choices should be made in order to generate text demonstrating the desired states of stylistic dimensions.

8.1 Construction of the Grammar Component

For the first component It is proposed to construct a new, non-trivial grammar (in English!) with a sufficiently rich lexicon and set of syntactic rules to generate paragraph length texts in a given knowledge domain. To assist the second component it will probably be necessary to mark in some way those features in the grammar that embody some form of choice in the resulting output (i.e. micro level stylistic effects); as distinct from those grammar features needed to ensure agreement, correct syntax and so on.

The above implies that a knowledge base in the chosen domain would also be required. A hopefully fairly straightforward semantic network should suffice for this. This would be consulted during the generation process to ensure that the generated text still “makes sense”. In addition, a simple planner, probably script based may well be needed. Both of these components will follow established practice as they are largely incidental to the proposed area of investigation.

The choice of domain should be irrelevant to the purpose of the investigation, however some criteria can be established:-

a) The domain be must regularly discussed in a variety of different styles; for example in children’s textbooks, student textbooks, training manuals, legal texts or spoken conversation. This will provide training material for the style component.

b) That the knowledge within the domain is largely “self contained” and hence can easily be represented in, for example, a reasonable sized semantic network. This is to ensure that this aspect, peripheral to the investigation does not require an inordinate amount of time and effort.

Given the above we would then have a text generator that, theoretically, would allow us to generate every possible way of expressing a concept from the knowledge domain, within the constraints of the grammar and lexicon.

It is proposed that the knowledge domain to be used be broadly classified as “destinations”, i.e. descriptions of places, their amenities and attractions. This domain meets the criteria above and the author has access to much material of this type.

8.2 Construction of the Style Component

As previous work, especially Hovy (section 4.3.2.4), seems to imply, it is difficult to isolate consideration of style during the generation process; it must be considered at almost every point - choice of words from the lexicon, choice of syntactical construct, order of sentences and so on. Although this may be feasible for generating texts that vary along one particular dimension of style it is likely to become unwieldy, both to construct and maintain, for multiple stylistic dimensions.

Instead, the author proposes to construct a "stylistic classifier" that will choose from amongst a selection of generated texts, the one text which most closely matches the required style. This type of classification would seem well suited to a neural network. As noted in section 6.3, the generator can provide not just the generated text, but also the parse tree, or any other property contained in the grammar, thus providing a rich source of input to the neural network.

We should at this point consider what properties of a given text are stylistically relevant and hence what the inputs to the classifier would be; and also how to encode them. Much of the skill in using ANNs is in determining this encoding. Firstly, the actual words are not relevant *per se*, what is relevant is what *type* of word has been used; whether it is common form, formal, scientific or whatever. Similarly, the actual parse tree is not entirely relevant, but the existence of certain properties of that tree. An example should clarify. It is proposed to encode the input as a vector of properties, '1' indicating presence of the feature, '0' its absence.

A "neutral" sentence, such as `"The cat sat on the mat"` exhibits no special properties; all words are in their basic form and the sentence order is conventional. This might be encoded as `[0,0,0,0,0,...]`.

Conversely, a sentence such as `"Upon the silken hearth rug the feline reclined with regal grace"` exhibits many stylistic features; inversion, adjective phrases, adverb phrases, unusual words and so on. This might be encoded as `[1,0,1,1,0,1,...]`.

This input form only caters for single sentences where we are more interested in longer pieces of text. To cater for this it is suggested that instead of simply marking the presence or absence of a feature we use the number of times that feature appears. To take account of different quantities of text we would also need to include in the input metrics such as number of words and the number of sentences in each text.

The exact list of stylistically significant properties has yet to be determined. In effect we will be carrying out the stylistic analysis suggested by Crystal and Davy, although we will be leaving the higher level classification to the ANN.

To provide a training set for this network it is proposed to locate a series of example texts covering the chosen domain and to classify these texts into different styles. (It may be worth asking a number of people to do this classification to ensure such categories would be widely accepted as such).

Within each text we will identify the presence or absence of our chosen stylistic properties. This then gives us a training set in the same form as the “live” examples that will be produced by the generation component. (It is possible that a program could be written to generate this training set automatically although initial thoughts would suggest that it may be quicker overall to do this manually). As is normal practice, the classifier can be tested against similar texts not in the training set to prove its operation.

8.3 Evaluation

Given the existence of the two components above, they can be used to investigate a number of questions.

- Firstly and most obviously, does such a system actually work in practice to produce texts in a given style?
- Although theoretically all possible texts can be generated and classified this is likely to be computationally expensive. Is it possible to give “hints” to the generator that reduces the number of texts that must then be classified?
- To extend this idea further, is it possible to introduce feedback from the classifier into the generator, possibly by weighting the various stylistic choice points and allowing the classifier to modify those weights?
- Is it possible to extend this further and actually allow a suitably trained neural network to control the generation process such that only one text is produced - the text that most closely matches the required style?

In short, we are proposing a hybrid approach to generation and exploring the range of possible combinations of conventional, grammar based generation with neural network classification.

It is the authors belief that this combination of conventional text generation with neural network classification to achieve variant styles is a new and promising area of research.

9. Appendices

9.1 Program Code

```
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
%
%   Example code from the book "Natural Language Processing in Prolog"
%
%                      published by Addison Wesley
%
%        Copyright (c) 1989, Gerald Gazdar & Christopher Mellish.
%
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
%
%
% pro_patr.pl [Chapter  7] Prolog PATR interpreter
% - originally written by Bob Carpenter (except where noted)

?- consult('dagunify.pl').

?- op(600,xfx,===).
?- op(500,xfx,--->).
?- op(400,xfx,ule).
?- op(600,xfx,ord).

/***********************************************************************
 * ALL CODE ADDED BY KRW IS COMMENTED LIKE THIS                        *
 * We define here a new operator for a special form of string concat-  *
 * enation for word suffixes (see the conjoin function below)          *
 * We also turn off style checking for unused variables (the 'R' in    *
 * the rules) and consult the lexicon and syntax files for Polish.     *
 ***********************************************************************/

/*************** New Code Start ***************************************/

?- op(600,xfx,++).          % string concatenation
?- style_check(-singleton).
% ?- style_check(-discontiguous).
% ?- consult('pol_lex.ptr').
% ?- consult('pol_syn.ptr').

/****************** New code End ************************************/

X1 === Y1  :-                   % following 2 predicate definitions
slightly
  denotes(X1,X2),               % changed from those shown in the book
  denotes(Y1,Y2),
  unify(X2,Y2).

denotes(Var,Var) :- var(Var), !.
denotes(Dag::Path,Value) :- !,
  pathval(Dag,Path,Value,_).
denotes(Constant,Constant).      % allows atomic or complex DAGs

% Left-corner recognizer due Pereira and Shieber, taken from
% their book _Prolog and Natural Language Analysis_ p. 180.

/********************* New Code Start *******************************/
% clause added to deal with concatenation operator
leaf(Dag1 ++ Dag2) --> [Word1 ++ Word2], {Dag1 ord Word1, Dag2 ord
Word2}.
/********************* New Code Ends ********************************/
```

```
leaf(Dag) --> [Word], {Dag ord Word}.
leaf(Dag) --> {_ ule Dag ---> []}.

recognize(Dag1) -->
  leaf(Dag0),
  left_corner(Dag0,Dag1).

left_corner(Dag1,Dag2) --> [], {unify(Dag1,Dag2)}.
left_corner(Dag1, Dag2) -->
  {_ ule Dag0 ---> [Dag1|Dags]},
  recognize_rest(Dags),
  left_corner(Dag0,Dag2).

recognize_rest([]) --> [].
recognize_rest([Dag|Dags]) -->
  recognize(Dag),
  recognize_rest(Dags).

% The 'test' predicate takes a list of words, and will print out
% the 'extensional' part of the dag that it gets recognized as (i.e.
% the part without path equivalences).

test(String) :-
  write(String), nl,
    recognize(Category,String,[]),
  pp_dag(Category), nl, nl.

% pretty printing routine to print out legible dags without
% specifying path equations.

pp_dag(Dag) :-
  pp_dag(Dag,0,yes).

pp_dag(VarTail,Column,yes) :-
  var(VarTail),!.
pp_dag(VarTail,Column,no) :-
  var(VarTail), write(VarTail), !.
pp_dag(_,Column,yes) :-
  nl, tab(Column), fail.
pp_dag([F :: Dag1|RestDag],Column,_) :-
  !, write(F), write(' :: '),
  atom_length(F,N), NewColumn is Column + N + 3,
  pp_dag(Dag1,NewColumn,no),
  pp_dag(RestDag,Column,yes).
pp_dag(Atom,_,_)  :-
  !, write(Atom).

atom_length(Atom,N) :-
  name(Atom,List),
  length(List,N).

/*************** New Code Start ************************************/

/*****************************************************************
 * The generate function here takes as an example a syntax category   *
 * and builds a DAG with that category. In principle any feature ::   *
 * value pair could be created by repeated application of 'pathval'   *
 * 'expand' is called to expand the given DAG into a parse tree. Note *
 * that expand actually expects a list of DAGs so we give it one.     *
 * For demonstration purposes we do not need the parse tree so we call*
 * 'strip' to remove the category markers and flatten to produce a    *
 * single list. 'pretty_print' prints the list, taking special care   *
 * of the ++ concatenation operator defined earlier.                  *
 *****************************************************************/
```

```
generate( Category, ParseTree ):- % generate possible strings matching
category
   pathval( Dag, cat, Category, _ ), % create dag with feature cat
having value
                        %Category
   do_expand( [Dag | [] ], ParseTree ).% build parse tree

do_expand( List, Tree ):-
   expand( List, Tree ),
   print_out( Tree ).       % and display it as a tidy string

print_out( Tree ):-
   strip( Tree, Tree2 ),       % remove syntax categories from the tree
   flatten( Tree2, List ),        % turn it into a simple list
   tell('pol_out.txt'),
   pretty_print( List ),
   !.                 % & print it. The cut ensures that on
                      % backtracking we redo only the 'expand'
                      % clause NOT 'flatten' & 'strip' etc.

/*********************************************************************
 * 'expand' is the routine that actually does the generation. The 1st *
 * goal simply ensures termination. The second is needed to cope with *
 * the special concatenation operator. It can be ommitted if the ++   *
 * operator is not used. The third clause will expand a DAG if there  *
 * is a rule in the syntax that expands it. RHS is returned as a      *
 * list of DAGs for further expansion. The final clause will expand a *
 * DAG into a word if there is one matching in the lexicon. The last  *
 * two clauses may be reversed, leading to a different order of the   *
 * various phrases that can be generated.                             *
 *********************************************************************/

expand( [], [] ).        % terminate at the end of the list

expand([Dag1 ++ Dag2 | Dags], [ Word1 ++ Word2 | Rest] ):-
   Dag1 ord Word1,    % look for two words matching the DAGs either side
   Dag2 ord Word2,    % of the ++ and put them either side of a ++ in
the
   expand(Dags, Rest).   % output list of words

expand([Dag | Dags] , [Cat, Words | Rest]):-   % A dag can be expanded
into a phrase
   _ ule Dag ---> RHS,            % if there is a rule that expands it,
   pathval(Dag, cat, Cat, _),        % get the category for the parse
tree
   expand(RHS, Words),            % expand the resulting list
   expand(Dags, Rest).            % and then the rest of the DAGs

expand([Dag | Dags], [Word | Rest]):-    % or it can be expanded into a
word if
   Dag ord Word,           % it can unify with a word in the lexicon.
   expand( Dags, Rest ).       % then expand the rest of the DAGs

/*********************************************************************
 * 'conjoin' is a function to handle word suffixes. It takes a root   *
 * word and the suffix in a special form - a digit is appended to the *
 * suffix. This is the numbers of characters that should be taken off *
 * the end of the root word before the suffix is added. It can vary   *
 * from zero to the length of the root word, in the latter case the   *
 * whole word is replaced, thus allowing for completely irregular     *
 * endings such as the Polish "pies" or dog. (see the lexicon)        *
 * Note that the present version only allows removal of up to 9 chars *
 * this could be extended if required but would be even more complex. *
 * Prolog is NOT good at this sort of stuff! If this was for real it  *
 * better be written in 'C' and called externally.                    *
 * NOTE: ALL CODE IN CONJOIN IS HEAVILY IMPLEMENTATION DEPENDANT!!!!  *
```

```
 ************************************************************************/

conjoin( R, x0 ):-       % we adopt the convention that x0 means
   write( R ).           % "no suffix". (x never occurs in Polish)

conjoin( R, S ):-
   string_to_atom( Root, R ),  % convert the root and suffix to strings
   string_to_atom( Suff, S ),
   string_length( Suff, SuffLen ),% determine their lengths
   string_length( Root, RootLen ),
   substring( Suff, SuffLen, 1, SuffLast ), % extract last char. of
suffix
   string_to_atom( SuffLast, SuffDigit ), % which is always a digit, so
convert to
   atom_to_term( SuffDigit, SuffBack, _ ), % integer so can do
arithmetic with it
   RootSub is RootLen - SuffBack, % calculate how much of root word we
need
   substring( Root, 1, RootSub, RootOut ),  % and extract if from the
root
   write( RootOut ),
   SuffSub is SuffLen - 1,     % lop off the last character from the
suffix
   substring( Suff, 1, SuffSub, SuffOut ), % (we don't print the digit).
   write( SuffOut ).

/************************************************************************
 * Turn a list of lists into a single list. Taken from Bratko, Prolog *
 * Programming for Artificial Intelligence, 2nd Edition.              *
 ************************************************************************/

flatten( [Head | Tail], Flatlist ):-
   flatten( Head, FlatHead ),
   flatten( Tail, FlatTail ),
   append( FlatHead, FlatTail, Flatlist ).

flatten( [], [] ).

flatten( X, [X] ).

/************************************************************************
 * Remove syntax categories from a parse tree. Actually it simply     *
 * removes any atom that immediately precedes a sublist, as this is   *
 * how our parse trees are constructed.                               *
 * NOTE 'IS_LIST' IS IMPLEMENTATION DEPENDANT!!!!                     *
 ************************************************************************/

strip( [], [] ).            % Ensure termination at end of list

strip( [_, List | Rest], [List1 | List2]):-
   is_list(List), !,        % if the 2nd item is itself a list
   strip(List, List1),      % then strip the sublist and the rest of the
   strip(Rest, List2).      % list. Note that '_' is ignored

strip( [Item | Rest], [Item | List1] ):-
   strip(Rest, List1).      % if 2nd item isn't a list then add first
item
                  % to list and strip the rest.

/************************************************************************
 * pretty print a list as space separated strings, taking special     *
 * notice of the ++ concatenation operator. We could wrap lines at a  *
 * given length but for demonstration purposes assume that all lists  *
 * are quite short.                                                   *
 ************************************************************************/
```

```
pretty_print( [] ):-        % if we are at the end of the list, print
newline
   nl.

pretty_print( [ Root ++ Suffix | Rest ] ):-
   conjoin( Root, Suffix ), % print suffixed words specially.
   write( ' ' ),
   pretty_print( Rest ).

pretty_print( [ Word | Rest ] ):-
   write( Word ),        % just print the rest of them as usual.
   write( ' ' ),
   pretty_print( Rest ).
```

9.2 Example Outputs

```
Welcome to SWI-Prolog (Version 1.9.0 June 1994)
Copyright © 1993,1994 University of Amsterdam.  All rights
reserved.

1 ?- consult('pro_patr.pl').
dagunify.pl compiled, -nan sec, 1,076 bytes.
pol_lex2.ptr compiled, -nan sec, 14,508 bytes.
pol_syn.ptr compiled, -nan sec, 1,504 bytes.
pro_patr.pl compiled, -nan sec, 22,924 bytes.

Yes
2 ?- generate(sn,S).

Adam
S = [sn,['Adam' ++ x0]] ;

Adama
S = [sn,['Adam' ++ a0]] ;

Adamowi
S = [sn,['Adam' ++ owi0]] ;

Adamem
S = [sn,['Adam' ++ em0]] ;

Adacie
S = [sn,['Adam' ++ cie1]] ;

ksia_ka
S = [sn,[ksia_ka ++ x0]] ;

ksia_ke_
S = [sn,[ksia_ka ++ e_1]]

Yes
3 ?- generate(sa,S).

dobry
S = [sa,[dobry ++ x0]] ;

dobrego
S = [sa,[dobry ++ ego1]] ;

dobremu
S = [sa,[dobry ++ emu1]] ;
```

In this first example we are asking for all "simple nouns". The system first prints the returned phrase (by a call to 'pretty print') and then shows what S became instantiated to, in this case the partial parse tree discussed above.

Lest this go on all day we finally accept a generated phrase and then repeat the query asking for simple adjectives. Note that the system first offers all cases of a word, then moves on to the next word. This is a consequence of the rule expansion appearing before the word expansion in 'expand'. If the clauses are reversed so is the order of presentation.

```
dobrym
S = [sa,[dobry ++ m0]] ;

dobra
S = [sa,[dobry ++ a1]] ;

dobra_
S = [sa,[dobry ++ a_1]] ;

dobrej
S = [sa,[dobry ++ ej1]] ;

dobrej
S = [sa,[dobry ++ ej1]] ;

dobre
S = [sa,[dobry ++ e1]] ;

dobrego
S = [sa,[dobry ++ ego1]] ;

dobremu
S = [sa,[dobry ++ emu1]]

Yes
4 ?- generate(ap,S).

dobry Adam
S = [ap,[sa,[dobry ++ x0],sn,['Adam' ++ x0]]] ;

dobry obiad
S = [ap,[sa,[dobry ++ x0],sn,[obiad ++ x0]]] ;

dobry pies
S = [ap,[sa,[dobry ++ x0],sn,[pies ++ x0]]] ;

dobry student
S = [ap,[sa,[dobry ++ x0],sn,[student ++ x0]]] ;

dobry park
S = [ap,[sa,[dobry ++ x0],sn,[park ++ x0]]] ;

dobry obiad
S = [ap,[sa,[dobry ++ x0],sn,[obiad ++ x0]]] ;

dobry psa
S = [ap,[sa,[dobry ++ x0],sn,[pies ++ psa4]]] ;

dobry park
S = [ap,[sa,[dobry ++ x0],sn,[park ++ x0]]] ;

dobrego Adama
S = [ap,[sa,[dobry ++ ego1],sn,['Adam' ++ a0]]] ;

dobrego obiadu
S = [ap,[sa,[dobry ++ ego1],sn,[obiad ++ u0]]] ;

dobrego psa
S = [ap,[sa,[dobry ++ ego1],sn,[pies ++ psa4]]]
Yes
5 ?- generate(pp,S).
blisko Adama
S = [pp,[blisko,sn,['Adam' ++ a0]]] ;

blisko ksia_ki
```

Finally we ask for all the prepositional phrases that the system knows about. There are two rules for such phrases, the first without an adjective, the second with. Because of this ordering phrases of the form 'preposition noun' are returned first.

We are now asking the system to generate all the adjective phrases it knows of. The order of presentation is again a consequence of the ordering of the clauses in 'expand'.

```
S = [pp,[blisko,sn,[ksia_ka ++ i1]]] ;

blisko obiadu
S = [pp,[blisko,sn,[obiad ++ u0]]] ;

blisko psa
S = [pp,[blisko,sn,[pies ++ psa4]]] ;

blisko studenta
S = [pp,[blisko,sn,[student ++ a0]]] ;

blisko studentki
S = [pp,[blisko,sn,[studentka ++ i1]]] ;

blisko sniadania
S = [pp,[blisko,sn,[sniadanie ++ a1]]]

<items omitted.............>
blisko dobrego obiadu
S = [pp,[blisko,ap,[sa,[dobry ++ ego1],sn,[obiad ++ u0]]]]
;

blisko dobrego psa
S = [pp,[blisko,ap,[sa,[dobry ++ ego1],sn,[pies ++ psa4]]]]
;

blisko dobrego studenta
S = [pp,[blisko,ap,[sa,[dobry ++ ego1],sn,[student ++
a0]]]] ;

blisko dobrego parku
S = [pp,[blisko,ap,[sa,[dobry ++ ego1],sn,[park ++ u0]]]] ;

blisko dobrej ksia_ki
S = [pp,[blisko,ap,[sa,[dobry ++ ej1],sn,[ksia_ka ++ i1]]]]
;

blisko dobrej studentki
S = [pp,[blisko,ap,[sa,[dobry ++ ej1],sn,[studentka ++
i1]]]]

Yes
6 ?- statistics.
nan seconds cpu time for 113,954 inferences
1,118 atoms, 732 functors, 1,052 predicates, 18 modules
21,422 byte codes; 3,527 external references
                     Limit    Allocated       In use
Heap         :                                204,996 Bytes
Local  stack :    2,048,000       32,768          404 Bytes
Global stack :    4,096,000      204,800          752 Bytes
Trail  stack :    4,096,000       32,768          312 Bytes

Yes
7 ?- halt.
```

After exhausting all possibilities of the first expansion rule the system then displays all possible expansions of 'preposition - adjective - noun'.

9.3 Glossary and Abbreviations

In this report a number of terms from linguistics have inevitably been used; some of these terms have technical meanings that differ from their everyday usage. In the glossary below these are identified by "(ling.)" in the definitions.

ANN Artificial Neural Network

Apposition (ling.) The placing of a phrase (esp. a noun phrase) after another to modify it's meaning (e.g. "Bill Clinton, *the President of the United States*, said that…").

ATN Augmented Transition Network. An RTN in which "processing" is allowed during transitions and states may have "memories" of previous events.

Cataphora (ling.) Use of a pronoun to refer to something that appears later in the sentence (e.g. "*He's* very tall, that policeman").

Computational Linguistics The application of computers to the field of Linguistics

Connectionism The use of artificial neural networks

Corpora (ling.) A collection of examples of text, either from a single author or more usually in linguistics, covering a range of uses of a particular language. Various "reference" corpora are available against which the performance of NLP systems can be evaluated.

DCG Definite Clause Grammar

Ellipsis (ling.) Referring, by implication, to something in an earlier phrase or sentence (i.e. words omitted from the sentence).

FUG Functional Unification Grammar

Grammar (ling.) A formal specification of the rules of syntax.

Inversion (ling.) Reversing the normal order of a phrase by for example putting the object at the front of the sentence. ("The *ball*, the man kicked").

Morphology (ling.) The rules by which words are put together out of smaller units ("morphemes") (e.g. anti-dis-establish-ment).

NLG Natural Language Generation

NLP Natural Language Processing (encompasses both understanding and generation)

Pragmatics (ling.) Those aspects of language use that interact with the "real world" The classic example in pragmatics is Chomsky's phrase "Colourless green ideas sleep furiously", which is morphologically and grammatically correct but does not make sense in a "real world" context.

RTN Recursive Transition Network. A network of states and transitions. A transistion between states is made based on the state of the input, and possibly generating an output. A state may itself contain a transition network.

Semantics (ling.) The meaning of words and phrases, separate from their grammatical construction.

Syntax (ling.) The arrangement of words and phrases within a particular language.

Unification Formally, the minimal set of which two or more other sets are all subsets.

9.4 References

Allen, R.B., (1992), Connectionist Language Users, in Sharkey (1992) pp163-195.

Appelt D. E., (1985), KAMP, Planning English Sentences, 1st ed., CUP, Cambridge.

Boulard H, & Morgan N. (1993), Connectionist Speech Recognition - A Hybrid Approach, Kluwer Academic, London.

Bratko I, (1990), PROLOG Programming for Artificial Intelligence, 2nd ed., Addison Wesley, Wokingham.

Cohen P.R., Feigenbaum E.A., (1982), The Handbook of Artificial Intelligence, Pitman, London.

Cole R.A. (Editor in chief), (1995), Survey of the State of the Art in Human Language Technology, National Science Foundation, USA.

Crystal D., (1969), Investigating English Style, 1st ed., Longman, London.

Crystal D., (1985) Linguistics, 2nd ed., Penguin, London.

Danlos L, (1987), The Linguistic Basis of Text Generation, CUP, Cambridge.

DiMarco C, Hirst G, (1993), A Computational Theory of Goal Directed Style in Syntax, Journal of Computational Linguistics, 19, suppl. 3, pp451-499.

Eggins S, (1994), An Introduction to Systemic Functional Linguistics, Pinter, London.

Elhadad M., McKeown K.R., (1991), A Contrastive Evaluation of Functional Unification Grammar, Natural Language Generation in Artificial Intelligence and Computational Linguistics (Eds. Paris, Swartout, Mann,), Kluwer Academic, Dordrecht, pp352-395.

Freeman J.A., Skapura D.M., (1991), Neural Networks, Algorithms, Applications and Programming Techniques, Addison Wesley, Reading Mass.

Friedman J, (1969), Directed Random Generation of Sentences, Communications of the ACM, 12, pp40-46..

Friedman, J, (1971), A Computer Model of Transformational Grammar, Elsevier, New York.

Gazdar G., Mellish C. (1989), Natural Language Processing in POP-11/LISP/PROLOG: an introduction to computational linguistics, Addison Wesley

Hovy E. H., (1987), Some Pragmatic Decision Criteria in Generation, Natural Language Generation (Ed. Kempen), Martinus Nijhoff, Dordrecht, pp3-17.

Jameson A., (1987), How to Appear to be Conforming to the Maxims even if you Prefer to Violate Them, Natural Language Generation (Ed. Kempen), Martinus Nijhoff, Dordrecht, pp19-41.

Klein S, (1965), Control of Style with a Generative Grammar, Language vol. 41, pp619-631.

Kukich K., (1987), Where do phrases come from: Some Preliminary Experiments in Connectionist Phrase Generation, Natural Language Generation (Ed. Kempen), Martinus Nijhoff, Dordrecht, pp405-419.

Luger, Stubblefield, (1989), Artificial Intelligence and the Design of Expert Systems, Benjamin / Cummings, London.

McKeown K.R., Swartout W.R., (1988), Language Generation and Explanation, Advances in Natural Language Generation (Eds. Zock & Sabah), Pinter, London, pp1-49.

McKeown, K. R, (1985), Text Generation: Using Discourse Strategies and Focus Constraints to Generate Natural Language Text, 1 ed., CUP, Cambridge.

Ogden C.K., (1930), Basic English, Routledge, London.

Orwell G, (1949), Nineteen Eighty Four, Penguin 20th C. Classics, London.

Paris C.L, (1993), User Modelling in Text Generation, 1 ed., Pinter, London.

Parisi D., Giorgi A., (1991), A Lexically Distributed Word Ordering Component, Natural Language Generation in Artificial Intelligence and Computational Linguistics (Eds. Paris, Swartout, Mann), Kluwer Academic, Dordrecht, pp51-57.

Quirk R,. (1968), The Use of English, 2nd ed., Longman, London

Reiter E, (1994), Has a Consensus NL Generation Architecture Appeared, and is it Psycholinguistically Plausible?, 1994 International NLG workshop.

Sharkey N, (1992) Connectionist Natural Language Processing, Kluwer Academic, London.

Sharples M, (1985), Cognition Computers and Creative Writing, Ellis Horwood, London.

Shieber S.M., (1986), An Introduction to Unification Based Approaches to Grammar, 2 ed., CSLI, Stanford.