Technical Report

UCAM-CL-TR-11
ISSN 1476-2986

Number 11

UNIVERSITY OF CAMBRIDGE

Computer Laboratory

Automatic resolution of linguistic ambiguities

Branimir Konstatinov Boguraev

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

http://www.cl.cam.ac.uk/

This technical report is based on a dissertation submitted August 1979 by the author for the degree of Doctor of Philosophy to the University of Cambridge, Trinity College.

Technical reports published by the University of Cambridge Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/techreports/

ISSN 1476-2986

Summary

The thesis describes the design, implementation and testing of a natural language analysis system capable of performing the task of generating paraphrases in a highly ambiguous environment. The emphasis is on incorporating strong semantic judgement in an augmented transition network grammar: the system provides a framework for examining the relationship between syntax and semantics in the process of text analysis, especially while treating the related phenomena of lexical and structural ambiguity. Word-sense selection is based on global analysis of context within a semantically well-formed unit, with primary emphasis on the verb choice. In building structures representing text meaning, the analyser relies not on screening through many alternative structures - intermediate syntactic, or partial semantic - but on dynamically constructing only the valid ones. The two tasks of sense selection and structure building are procedurally linked by the application of semantic routines derived from Y.Wilks' preference semantics, which are invoked at certain well chosen points of the syntactic constituent analysis - this delimits the scope of their action and provides context for a particular disambiguation technique. The hierarchical process of sentence analysis is reflected in the hierarchical organisation of application of these semantic routines - this allows the efficient coordination of various disambiguation techniques, and the reduction of syntactic backtracking, non-determinism in the grammar, and semantic parallelism. The final result of the analysis process is a dependency structure providing a meaning representation of the input text with labelled components centered on the main verb element, each characterised in terms of semantic primitives and expressing both the meaning of a constituent and its function in the overall textual unit. The representation serves as an input to the generator, organised around the same underlying principle as the analyser - the verb is central to the clause. Currently the generator works in paraphrase mode, but is specially designed so that with minimum effort and virtually no change in the program control structure and code it could be switched over to perform translation.

The thesis discusses the rationale for the approach adopted, comparing it with others, describes the system and its machine implementation, and presents experimental results.

Preface

I would like to thank my supervisor, Dr. Karen Sparck Jones, for her invaluable assistance and unfailing support throughout the course of this research and for her remarks, suggestions, criticisms and continuous guidance during the last three and a half years which kept me on the right track. Her endurance and patience in going through draft versions of this manuscript cannot be overemphasised, and her help is greatly appreciated.

The programming would have been impossible without the expert advice of Dr. A.C.Norman who kept the LISP system up and going, and spent a considerable amount of time and effort helping me get the best out of it.

I am grateful to the many people with whom I have had interesting and stimulating discussions. Especially, I would like to thank A.Cater, with whom I shared the joys of research, both for his constructive remarks and suggestions and for his permission to incorporate some of his routines into my system.

I am grateful to Prof. M.V.Wilkes and the Cambridge University Computer Laboratory for providing me with a friendly environment and facilities without which it would be impossible to carry out this work. I am also grateful to Trinity College, Cambridge, for providing financial support while I was a research student.

And finally, I would like to thank all my friends who were always there when I needed them.

* * *

Except where otherwise stated in the text, this dissertation is the result of my own work and is not the outcome of any work done in collaboration.

No part of this dissertation has been submitted for any other degree or diploma at any other university.

Contents.

chapter 0.
General Introduction.

0.1. What This Work Hopes To Achieve.

0.1.1. Overview.

The work reported in this thesis can be viewed as an attempt to achieve several goals which, although corresponding to different levels of generality within the framework of the task of analysing natural language, are closely related.

At the outermost level, the aim is to design and implement a Natural Language (NL) Processing System capable of performing the task of translation/ paraphrase. The original objective was machine translation; "/paraphrase" indicates the way this project evolved. The reasons for the switch are explained in chapter 5; however, for the purposes of analysis, and throughout this work, except where explicitly stated, I shall regard translation/ paraphrase as a single task. Thus in general terms the work has been concerned with the analysis of input text sentences, the construction of a meaning representation of these sentences, and the generation from the representation of output paraphrase sentences. The production of acceptable paraphrases is taken as a test of effective analysis and representation.

The overall objective in particular provides a framework within which it is possible to examine the relation between syntactic and semantic processes of sentence analysis, its implications for automatic text processing, and specifically for the possible ways in which semantics can be integrated into an augmented transition network (ATN) analysis mechanism (see 2.1). It is argued that a purely semantic network is difficult, if not impossible to design. Closer examination of both the reasons for this, and the possible ways of overcoming the problems involved, viewed in the light of the overall objective, leads to the conclusion that even though the primary preoccupation of a natural language analysis system should be the semantic analysis of the incoming text, some conventional syntactic information must be exploited during processing. The analyser therefore seeks to incorporate strong semantic judgement within the framework supplied by syntactically driven parsing. The relation between syntax and semantics is thus made manifest in the use of a transition network processor combining syntactic constituent analysis with the basic ideas of Wilks' theory of preference semantics.

Next, there are the immediate practical problems facing the parser. These are determined both by the overall objective and the strategy adopted, as stated above. One of the most important tasks facing the semantic component of the analyser is the resolution of

word-sense ambiguity, with no particular restriction on the types of items which can be interpreted in more than one way: these can be verbs, or nouns, or adjectives. A mechanism has been developed, based on semantic pattern matching of constituents, which makes it possible for the system to perform in an environment of highly polysemic dictionary entries, without making the entries themselves, or the grammar, unnecessarily complicated. On the other hand, the desire to make use of the augmented transition network mechanism highlighted another problem for the design of an analyser along these lines: the resolution of structural ambiguities. It turned out that these two problems are not entirely unconnected, and a unified approach is needed.

Efficiency in the operation of the analyser is sought as well. To achieve this, it was necessary to modify an existing model of an ATN grammar in order to minimise the unwanted proliferation of alternative syntactic paths (caused mostly by unnecessary embedding). Judicious application of semantic specialist routines, applied when just enough information has been accumulated for them to work on, eliminates (to a large extent) the element of blind search and avoids premature (and possibly wrong) decision-making about the course of the analysis and the semantic structures to be built to represent sentence meaning.

The work of the analyser is thus focussed on the related problems of resolution of lexical and structural ambiguities. The final product of the analysis process is an unambiguous dependency structure equivalent (in some sense) to the meaning of the input text. This is a hierarchically organised verb-centered semantic unit made up of individual components characterised in terms of semantic primitives and linked by case relationships - thus providing a meaning representation of the input text in which both component interpretation and component function (participant role) are made explicit. The hierarchical structure of the representation reflects the manner in which it is built up by the the parser, namely by hierarchical application of the semantic routines, without going through extensive intermediate syntactic representations. It should be emphasised that this structure does not necessarily simply reflect the surface structure of the input sentence.

The nature of the representation structure to be built was a matter of concern in its own right; but since the structure is also taken as input by the paraphrase generation program, the study of its content and format in relation to this use was another aspect of the work. Indeed the production of paraphrases can be regarded as a more useful way of evaluating delivered structures than mere study of their appearance.

The generator is thus designed to interpret the dependency structure back into natural language, and specifically into English. However, it should be emphasised that the choice of paraphrase rather than translation for output is essentially one of convenience, as is argued in chapter 5; for certain classes of paraphrases, the manipulations involved in the generation process are conceptually the same as for translation from English to another language, thus making

it possible to refer to translation/ paraphrase as the overall task of the system; and to design the generator for this overall task.

0.1.2. Cognitive Studies.

The work intended in this thesis is not presented as a model of language, or language comprehension, hence no claims are made as to its relevance to, or implications for, cognitive studies in general. Needless to say, use of language is a typically human activity, and in the course of communicating with each other, people do employ (subconsciously) processes of language interpretation and utterance generation. Although the system implemented is aimed at performing similar tasks, the ultimate aim of the project is to simulate rather than to model certain types of activities essential to the task of automatic translation/ paraphrase.

Clearly, the system can be regarded, on an abstract level, as a performance model. However, any references made to some (intuitive) processes possibly going on in the human mind, concern very general issues such as the relationship and interaction of syntax and semantics throughout the analysis process, and the implications thereof for the organisation and implementation of a computer based system for natural language processing. No psychological justification is offered for the specific devices, mechanisms and principles utilised by this project, though psychological claims have been made for ATN mechanisms [Kaplan72], for semantic primitives as the basis of a representational system [Wilks77], for the "wait-and-see" principle controlling analysis [Marcus75], etc., or could possibly be found for different features of the system.

In the process of developing the system, language, and especially syntax, characterisations taken from linguistics have been exploited. Some description of language features was necessary, and the most natural and convenient way of providing this was to use (relatively) well established categorisations and terms. Thus no claims are made for any specifically linguistic contributions, though the project perhaps pushes the application of some existing ideas further than other projects. Examples are the use of a modified set of recognition rules, an extended system of deep semantic cases, etc. (see below). The reasons for their application can be best understood in the light of a unified approach to the task of automated language analysis, and will be discussed in detail in due course.

0.1.3. The System Implementation.

The general principles of system design and organisation and the proposed mechanisms as discussed below have been utilised for the machine implementation of the proposed NL analysis system. This is a LISP program [McCarthy62], running offline on the Cambridge University Computing Service IBM 370/165. The program occupies about 110 Kbytes in its compiled version (representing about 15000 lines of source

code) with separate analyser and generator components.

The analyser program performs the transformation from the input text to a LISP data structure with a fixed, well-defined format, and equivalent to an unambiguous meaning representation. It combines the work of a front end, low-level syntactic preprocessor, running on top of an ATN interpreter, with semantic specialist routines which handle the constituents recognised by the syntactic component and incorporate their semantic interpretations into larger semantic units. The grammar rules controlling the work of the interpreter are kept in a separate data structure, the ATN grammar. Most of the information needed by the semantic routines is supplied by the dictionary. Between themselves, the grammar and the dictionary contain most of the knowledge base of the analysis component; the rest is distributed procedurally among the functions of the analyser. The hierarchically organised process of recursive application of the specialist routines defines the construction of an intermediate meaning representation, one for each distinct reading of the sentence. The results of the analysis phase are taken as input by the paraphrase generation program, which defines an inverse transformation generating English paraphrases. The generation is essentially a two stage process. Context sensitive synonyms selection underlies the conversion of the input semantic representation into an intermediate data structure called an environment network. This network simultaneously specifies the target language syntactico-semantic relationships and a set of function calls which procedurally define the actual process of paraphrase generation.

The system runs in 500 Kbytes and a typical processing cycle "sentence analysis => paraphrase generation" can take anywhere between 0.5 and 2 seconds, depending on the syntactic complexity of the sentence and the population of alternative readings (see the problem of multiple choice for a blind language processor: 0.3, 1.1, 1.2).

Below is an illustrative sample* of some of the types of sentential constructs and multiple choice problems that the system can handle. The emphasis is clearly on resolving potential ambiguities in the input: selecting the correct word-sense where the dictionary entry contains more than one definition of a lexical item ("ask", "call", "afraid", "club", "crook",...), and/or carrying out a correct structural interpretation. Note that in the case of a genuinely ambiguous input there are more than one paraphrase reflecting the multiple readings+.

The following several pages are an output from a typical run of the program with all the debugging switches off. The timing figures mean 'effective processing + time spent garbage collecting' i.e. only the first figure is actually relevant.

---

* for more examples see Appendix (iv).

+ the analyser works on isolated sentences, and genuine ambiguity has to be accounted for.

```
CAMBRIDGE LISP SYSTEM ENTERED IN ABOUT 500 KBYTES  PARM='V'
CORE IMAGE WAS MADE AT  15.27.13    ON    16 JUN 79
LISP VERSION - VER1 LEV2   IMAGE SIZE = 109468 BYTES
STARTED AT 21.47.59 ON 25 JUL 79 - 59.5% STORE USED

 End of initialising process after 14157+1644 msecs

 Sentence :  JOHN ASKED MARY ABOUT THE BOOK.
**********

 End of analysis after 452+911 msecs
 319+34 msecs generating the environment net
 6 msecs generating the sentence

 Paraphrase:  * John questioned Mary about the book.
**********


 Sentence :  JOHN ASKED MARY FOR THE BOOK.
**********

 End of analysis after 419+0 msecs
 210+0 msecs generating the environment net
 7 msecs generating the sentence

 Paraphrase:  * John requested the book from Mary.
**********


 Sentence :  JOHN ASKED MARY A FAVOUR.
**********

 End of analysis after 429+666 msecs
 230+0 msecs generating the environment net
 3 msecs generating the sentence
 Paraphrase:  * John begged Mary for a favour.
**********


 Sentence :  JOHN ASKED THE POLICEMAN TO INTERROGATE THE CROOK.
**********

 End of analysis after 469+13 msecs
 193+679 msecs generating the environment net
 7 msecs generating the sentence

 Paraphrase:  * John begged the policeman to question the artful
                dodger.
**********


 Sentence :  JOHN IS AFRAID OF THE CROOK.
```

```
**********

 End of analysis after 349+0 msecs
 692+0 msecs generating the environment net
 6 msecs generating the sentence

 Paraphrase:  * John fears the artful dodger.
**********


 Sentence :  JOHN IS AFRAID THAT BILL LOVES MARY.
**********

 End of analysis after 552+662 msecs
 566+0 msecs generating the environment net
 13 msecs generating the sentence

 Paraphrase:  * John suspects that Bill is in love with Mary.
**********


 Sentence :  JOHN IS AFRAID TO TELL THE GIRL THAT HE LOVES THE
             TRUTH.
**********

 End of analysis after 1701+2087 msecs
 622+835 msecs generating the environment net
 6 msecs generating the sentence

 Paraphrase:  * John doesn't want to tell the girl that he
                admires the truth.
**********
 602+775 msecs generating the environment net
 10 msecs generating the sentence

 Paraphrase:  * John doesn't want to tell the truth to the girl
                with whom he is in love.
**********


 Sentence :  JOHN IS AFRAID OF CALLING ON MARY.
**********

 End of analysis after 1135+1378 msecs
 533+6 msecs generating the environment net
 3 msecs generating the sentence

 Paraphrase:  * John doesn't want to visit Mary.
**********


 Sentence :  JOHN CALLED MARY TO FLY TO PARIS.
**********
```

```
 End of analysis after 1028+779 msecs
 243+772 msecs generating the environment net
 6 msecs generating the sentence

 Paraphrase:  * John summoned Mary to fly to Paris.
************


 Sentence :  IT WAS RUMOURED THAT THE PROPOSALS CALLED FORTH
             HOSTILE CRITICISM.
**********

 End of analysis after 1547+1481 msecs
 239+0 msecs generating the environment net
 10 msecs generating the sentence

 Paraphrase:  * someone spread rumours that the proposals
                provoked hostile criticism.
************


 Sentence :  THE CIRCUMSTANCES CALL FOR AN APOLOGY.
**********

 End of analysis after 469+775 msecs
 173+0 msecs generating the environment net
 7 msecs generating the sentence

 Paraphrase:  * the circumstances demand an apology.
************


 Sentence :  THE WORKERS CALLED OFF THE STRIKE AT THE LAST
             MOMENT.
**********

 End of analysis after 1444+756 msecs
 229+16 msecs generating the environment net
 10 msecs generating the sentence

 Paraphrase:  * the workers cancelled the strike at the last
                moment.
************


 Sentence :  JOHN CALLED BILL A FOOL.
**********

 End of analysis after 486+703 msecs
 456+0 msecs generating the environment net
 10 msecs generating the sentence

 Paraphrase:  * John thought that Bill is a fool.
************
```

```
 Sentence :  JOHN CALLED BILL A TAXI.
**********

 End of analysis after 629+732 msecs
 186+33 msecs generating the environment net
 7 msecs generating the sentence

 Paraphrase:  * John ordered a taxi for Bill.
**********


 Sentence :  JOHN STOPPED CALLING AT THE CLUB.
**********


 End of analysis after 573+699 msecs
 153+0 msecs generating the environment net
 3 msecs generating the sentence

 Paraphrase:  * John ceased to visit the meeting place.
**********



 Sentence :  JOHN STOPPED TO CALL AT THE CLUB.
**********

 End of analysis after 669+759 msecs
 139+0 msecs generating the environment net
 7 msecs generating the sentence

 Paraphrase:  * John stopped in order to visit the meeting
                place.
**********



 Sentence :  JOHN KILLED BILL STRIKING HIM WITH THE CLUB.
**********

 End of analysis after 699+712 msecs
 223+699 msecs generating the environment net
 6 msecs generating the sentence

 Paraphrase:  * John hit Bill with the missile weapon and killed
                him.
**********



 Sentence :  KISSING AUNTS CAN BE BORING.
**********

 End of analysis after 909+1474 msecs
 718+742 msecs generating the environment net
 6 msecs generating the sentence
 Paraphrase:  * the aunts who kiss someone can be dull.
**********
```

672+716 msecs generating the environment net
6 msecs generating the sentence

Paraphrase: * (the act of) kissing the aunts can be not particularly exciting.
****

Sentence : SHOOTING ELEPHANTS CAN BE DANGEROUS.
****

End of analysis after 955+1457 msecs
655+746 msecs generating the environment net
10 msecs generating the sentence

Paraphrase: * (the act of) shooting the elephants can be hazardous.
****

Sentence : CALLING ON MARY, JOHN DECIDED TO ASK FOR THE BOOK.
****

End of analysis after 802+1404 msecs
237+0 msecs generating the environment net
7 msecs generating the sentence

Paraphrase: * John made a decision to request the book while John was visiting Mary.
****

Sentence : JOHN IS EAGER.
****

End of analysis after 200+682 msecs
593+17 msecs generating the environment net
3 msecs generating the sentence

Paraphrase: * John is impatient.
****

Sentence : JOHN IS EAGER TO PLEASE.
****

End of analysis after 316+0 msecs
486+769 msecs generating the environment net
3 msecs generating the sentence

Paraphrase: * John wants very much to make someone happy.
****

Sentence : JOHN IS EASY.

```
**********
 End of analysis after 163+656 msecs
 493+0 msecs generating the environment net
 4 msecs generating the sentence

 Paraphrase:  * John feels relaxed.
**********


 Sentence :  JOHN IS EASY TO PLEASE.
**********
 End of analysis after 250+0 msecs
 250+0 msecs generating the environment net
 4 msecs generating the sentence

 Paraphrase:  * it is not difficult for someone to make John
                happy.
**********


 Sentence :  I SAW THE MAN IN THE PARK WITH THE TELESCOPE.
**********
 End of analysis after 499+786 msecs
 163+0 msecs generating the environment net
 7 msecs generating the sentence

 Paraphrase:  * with the telescope , and in the park , I saw
                the man.
**********

 123+679 msecs generating the environment net
 3 msecs generating the sentence
 Paraphrase:  * in the park which had the telescope , I saw the
                man.
**********

 120+0 msecs generating the environment net
 6 msecs generating the sentence

 Paraphrase:  * with the telescope , I saw the man who was in
                the park.
**********

 117+0 msecs generating the environment net
 6 msecs generating the sentence

 Paraphrase:  * I saw the man who had the telescope , and who
                was in the park.
**********

 113+0 msecs generating the environment net
 4 msecs generating the sentence

 Paraphrase:  * I saw the man who was in the park which had the
                telescope.
```

In order to give the reader some idea of the intermediate stages of processing (see above), the dependency structures and environment networks produced by the analyser and generator respectively for two of the example sentences are displayed below.

The exact format of these intermediate data structures, the rules for constructing the representations of the individual constituents based on semantic primitives, the rules for the procedural interpretation of the environment network, and other related issues will be discussed in detail in the following chapters.

For the time being it is sufficient to point out that semantic primitives are the basic elements of the meaning representation. The few basic elements not of this kind are embedded clause headers, and some markers with a quite obvious function: tns, type, etc. It is apparent that traces of the input words are carried over into the dependency structure: "John", "call2", "stop3", "stop4". These are however merely a shorthand for the semantic definitions of these words (see 3.1), the definitions themselves following immediately after, thus "stop3" and "stop4" simply stand for different senses of the (surface) lexical item "STOP" (for detailed discussion see 3.11).

The dependency structure for the first sentence, "John stopped calling at the club", for example, can loosely be interpreted in the following way. A potent subject/ agent (John) does not do an act any more, and the act itself is that of John moving himself to a location (point) in space which (possibly) has other people in it (i.e. the club). All in all, a rather long winded way of saying that "John ceased to visit the club" ("meeting place" denotes one of the readings of "CLUB", as opposed to "missile weapon"). The other structure can be interpreted similarly.

The format and content of the environment network (see chapter 5) is intuitively and generally clear; I shall not discuss it here.

```
 Sentence :  JOHN STOPPED CALLING AT THE CLUB.
**********

Structure1
   (clause
      (type dcl)
      (tns past)
      (v
         (stop3
            ((*pot subj) ((act obje) notdo))
            (@@ agent (John (mal (indiv man))))
            (@@
               act
               (clause
                  (v
                     (call2
                        ((man subj)
                           ((self obje)
                              (((where point) to) move)))
                        (@@
                           agent
                           (John (mal (indiv man))))
                        (@@
                           location
                           (club2
                              ((((this man) obje) wrap)
                                 spread)))))))))))

 End of analysis after 622+802 msecs

 263+23 msecs generating the environment net


 7 msecs generating the sentence


 Map of environment net
*************************

 The top node is: syntax-node1

syntax-node1:   ((agent . syntax-node2)
                   (tense . past)
                   (form)
                   (lexverb . cease)
                   (tocompl . syntax-node5))
syntax-node2:   ((lexnoun . syntax-node3))
syntax-node3:   ((english . syntax-node4))
syntax-node4:   (John)
syntax-node5:   ((dummy-agent . syntax-node6)
                   (tense)
                   (form . to)
                   (lexverb . visit)
                   (object . syntax-node7))
```

```
syntax-node6:   ((lexnoun . syntax-node3))
syntax-node7:   ((lexnoun . syntax-node8))
syntax-node8:   ((english . syntax-node9))
syntax-node9:   (meeting place)

 Paraphrase:  * John ceased to visit the meeting place.
************

 Sentence :  JOHN STOPPED TO CALL AT THE CLUB.
**********

Structure1
   (clause
      (type dcl)
      (tns past)
      (v
         (stop4
            ((*hum subj) ((((act obje) do) goal) notmove))
            (@@ agent (John (mal (indiv man))))
            (@@
               goal
               (clause
                  (tns present)
                  (v
                     (call2
                        ((man subj)
                           ((self obje)
                              (((where point) to) move)))
                        (@@
                           agent
                           (John (mal (indiv man))))
                        (@@
                           location
                           (club2
                              ((((this man) obje) wrap)
                                 spread)))))))))))

 End of analysis after 662+0 msecs
 164+0 msecs generating the environment net


 3 msecs generating the sentence


 Map of environment net
************************

 The top node is: syntax-node1
syntax-node1:   ((agent . syntax-node2)
                 (tense . past)
                 (form)
                 (lexverb . stop)
                 (goal . syntax-node5))
syntax-node2:   ((lexnoun . syntax-node3))
```

```
syntax-node3:   ((english . syntax-node4))
syntax-node4:   (John)
syntax-node5:   ((dummy-agent . syntax-node6)
                   (tense)
                   (form . to)
                   (lexverb . visit)
                   (object . syntax-node7))
syntax-node6:   ((lexnoun . syntax-node3))
syntax-node7:   ((lexnoun . syntax-node8))
syntax-node8:   ((english . syntax-node9))
syntax-node9:   (meeting place)

 Paraphrase:  * John stopped in order to visit the meeting place
************
```

0.2. Some Basic Terminology.

This section introduces some terms which are extensively used in the rest of the thesis. The definitions are informal and the precise meaning of the expressions should become clear from their use.

In the traditional theoretical linguistics framework [Lyons68] two basic terms are used to define two complementary aspects of language description: "the form of ... words and the manner of their combination in phrases, clauses, and sentences [are described] by grammar; and the meaning, or content, of the words (or of the units composed of them) by semantics". In this framework syntax is taken to be that part of grammar which accounts for the ways in which words can combine to form sentences.

However, a computer based Natural Language Analysis System is not a descriptive model of language. For the purposes of this work, a natural language processor will be defined as a system which accepts natural language text as input, uses syntactic and semantic processes to transform it into an internal representation, and applies deductive and/or inductive procedures to the representation to drive the performance mechanism. This definition clearly presents syntax and semantics in a different light: they are not to be treated only as descriptive devices they are now collections of active rules and principles, basic to the whole process of analysis. The overall process is carried out by the already mentioned NL processor, which may be referred to as the analyser or parser.

As specified in the theory of formal languages [Aho72], parsing implies at least two things: recognition of well-formed surface strings, conforming to a certain defined surface grammar; and assigning structural interpretations to them. The recognition process, which just answers "yes" or "no" to the question about the well-formedness of a given string, is guided strongly by a set of recognition rules, which represents one type of knowledge that a NL processor should have access to. These recognition rules define what I shall refer to in the context of this presentation as (low level) syntax.

As a consequence of such a pragmatic definition of syntax, the term grammar now denotes the actual embodiment of the recognition rules in a machine understandable form, together with another type of data needed by a NL analyser: instructions as to how to proceed with the process of syntactic recognition. The grammar also specifies (it is irrelevant at this point whether explicitly or implicitly) the mapping from surface text to an equivalent corresponding structural representation of it. In view of the background of this project, 'grammar' will specifically mean (except where explicitly stated otherwise) the machine interpretable form of an augmented transition network grammar (see 2.1) which performs all the functions specified

above.

Once a well-formed syntactic constituent (one of the "phrases, clauses, or sentences" as mentioned above) has been recognised, its meaning has to be represented in a certain manner. Without going into any details as to what meaning is, or might be, in the light of a particular system objective, we note that a semantic theory is needed: first, to specify a set of judgemental criteria, allowing or disallowing the acceptance of a certain piece of text (well-formed constituent) as meaningful, or meaningless; then to define a formalism for expressing the meaning of a word or a constituent from the meanings of their component parts (if such exist, or can be specified); and, finally, to posit a set of principles related to the way in which those criteria should be applied. The term semantics alone will be used below in a restricted sense, applying only to the judgemental criteria and the principles behind their application. Clearly, all this necessitates yet another type of knowledge which a NL analysis system should utilise.

As the definition of a NL processor given above implies, the final result of the analysis process is an internal representation of the meaning of the input text which is used to drive the performance mechanism. It naturally has to be predominantly semantic in nature, since semantics, as defined above, makes the meaning relations part of its domain. Hence, a semantic theory also provides the basis for a semantic representation of a constituent recognised by the syntactic recognition rules. For the rest of this presentation, 'internal representation' and 'semantic representation' will be considered to be synonymous.

0.3. Syntax And Semantics In a Natural Language Processing System.

The previous section defined a NL processor as a system which converts natural language input into an internal representation. What the definition is not at all specific about is the form of this internal representation, as well as any particular details of the syntactic and semantic processes that carry out the conversion.

Early research in computational linguistics was strongly influenced by Chomsky's categorisation of grammars and in particular by his transformational theory of natural language (TG) [Chomsky57], and by formal language theory and compiler writing: elements of language are formally defined objects in a well-defined syntactic structure [Aho72]. Both lines of work emphasised syntactic processing and more and more powerful syntactic analysers were developed, based on different linguistic theories, but nevertheless marking an increased acceptance of the distinction between surface and deep syntactic structures of language: the Scientific String Parser of Sager [Sager73], Kaplan's General Syntactic Processor [Kaplan73], the Chart Parser of Kay [Kay73], Woods' ATN parser [Woods70]. Considerable, but much more unorganised and unstructured effort was invested into gaining insight into what exactly is meant and expected from the 'semantic analysis' component of a natural language system. It is generally accepted that this is the 'understanding' ingredient of such a system: thus it is responsible for extracting and representing the meaning of a piece of text.

One of the problems here is the relationship between the syntactic and semantic processes that take place in the course of text understanding; what is their relative weight and importance; and how can, and should, they be integrated in a complete natural language analysis system? Linguistic theories do not help much in this respect because views differ radically. At the one end of the spectrum there is the view that syntactic and semantic processing are decoupled and can be performed serially (deep interpretive semantics). At the other end, the generative semanticists relate a 'deep semantic representation' to a surface form - without going through an intermediate syntactic structure. And even though research in automatic natural language analysis tends to be more pragmatic, the implications of linguists' views are too important to be dismissed easily. As it is, there are many different approaches and strategies for designing natural language analysis systems (see chapter 2). The problem is further complicated by the fact that each of these systems has very specific objectives, and these objectives differ substantially. Thus the various approaches can only be evaluated in an insufficiently informative, or rather non-uniform, environment. Furthermore, preoccupation with semantic processing resulted in the devaluation of syntactic analysis even when performed by a powerful mechanism. while sufficiently detailed semantic analysis programs suffer from incomplete coverage of English forms. Also, it is not very clear in

general, how to generalise any particular semantic approach to deal with a wider spectrum of language phenomena. Thus, exploring the position of conventional syntax in a generalised semantics based system, as well as the possibility of embodying semantic evaluation within an existing syntactic analysis framework, became one of the incentives for this research.

The other problem with the semantic analysis component of a natural language system is directly related to the mapping 'surface text => formal internal representation'. As Woods notes, "semantics has both a judgemental and a structural aspect" [Woods75]. In other words, apart from constructing a semantic representation of the meaning of an input sentence, semantics should be responsible for rejecting anomalous or ill-formed structures. However, my feeling is that most researchers in the field of natural language processing have made the structural aspect their main study. The very nature of their approach emphasises that: it starts with defining the syntax and semantics of the internal representation, and proceeds to attack the question of how the mapping into it should be performed. On a general level this is all right, but such an approach does not place enough emphasis on the equally important judgemental aspect of semantics. It may be argued that the mere definition of a format for the internal representation, together with a specified procedure for carrying out the conversion, prohibits the building of ill-formed structures. On the other hand, noone argues the fact that natural language is inherently ambiguous. What follows from this is that a natural language analysis program is essentially a 'blind' processor which faces the problem of multiple choice (see 1.1, 1.2) all the time, and must, therefore, have all its information and decision processes complete and explicit. Still, how many systems actually attempt to solve the problem in its generality? (see chapter 2). The only major project which starts with the judgemental aspect of semantics is Wilks' approach to machine translation (see below). He openly acknowledges the polysemy of the words and designs his semantic theory round it. However, in such an approach, the internal representation might suffer if one is not careful enough, as indeed Wilks' does (see 2.2, 3.11). In the attempt to keep the structure simple and uniform, it is not defined rigorously enough, so that it is not clear whether it can provide for a wider variety of surface texts, or how well it can serve as a basis for driving the performance mechanism (the translation generator).

The situation clearly requires achieving the right balance: one's premises have to be carefully defined. In the light of the remarks made above, I decided to concentrate on the problems associated with the judgemental aspect of semantics: resolution of linguistic (lexical and structural) ambiguities; and to take the analysis process to its logical conclusion by constructing and delivering an unambiguous internal representation. In such a framework it will be possible to deal with the problems both of discrimination and representation [Sparck Jones65], which are the cornerstone of a computationally feasible semantic theory.

chapter 1.
The Nature Of The Problem.

1.1. Ambiguity Or The Problem Of Multiple Choice.

The Concise Oxford Dictionary offers a definite view as to what should be considered to be an ambiguous sentence: "an expression capable of more than one meaning". From the point of view of a human trying to extract some meaning from a piece of text this is adequate and intuitively true. He will normally assign more than one interpretation to

John asked Mary where the club is, or

He shot the man with the gun, or

He took her grapes,

while judging that

John shot the man with the club, and

John admitted Bill to the club

have only one meaning, and hence are unambiguous, according to the definition.

However, from the point of view of a NL processor, both cases require making of a decision of some sort. Neither when or how the decision is made, nor its outcome, is relevant at the moment. What is relevant is the fact that the program faces the problem of multiple choice. Assuming that the data file(s) accessible to it contain (at least) two separate entries, reflecting (somehow) the "weapon" and "place" connotations of "club", the analyser will have to perform a rational choice every time it comes across the lexical item "club" - even when no such choice-making, or a similar process, seems to take place in reality. Also, in the last two examples above the program will have to decide on who has the club - John, or the man, as well as on the meaning of "admit".

Throughout this work, I shall use the term 'ambiguity', rather than 'multiple choice', as exemplified above. Hence 'disambiguation' will involve finding the correct reading(s) of a word or/and the correct structural interpretation of a semantic unit (the exact meaning of the term will become clear in due course). In the case of more than one plausible choice, the analyser will deliver them all, as if presenting them to a higher level deductive component which will try to examine, for example, the first of the sentences above in the context of John planning an assault, or John intending to go somewhere. The design of such a component is outside the scope of this project.

What follows is an attempt to analyse the causes for ambiguity, and recognise the situations in which the system under discussion will be faced with the problem of multiple choice.

Very broadly speaking, a computer based NL analysis system faces three main types of ambiguity:

(1) word-sense (lexical) ambiguity,
(2) structural ambiguity,
(3) referential ambiguity.

These are discussed in more detail below. (The discussion does not claim to be an extensive linguistic treatment of ambiguity: it is intended only to convey the reality and form of the problem for a 'blind' automatic analyser - see 0.3.)

1.1.1. Word-sense ambiguity

The roots of this phenomenon (also known as lexical ambiguity) are multiple definitions of items in the lexicon. This presentation will ignore the important theoretical issue of drawing the line between polysemy (roughly "a lexical item having more than one senses") and homonymy ("two or more items having the same pronunciation, and more important, spelling") [Lyons77]. The issue is irrelevant from the point of view of a computer - a choice mechanism application is clearly called for in both cases. What is more important is the appreciation of the situations which necessarily trigger the choice making mechanism.

These can be classified as:

(A.) conceptually similar words* (close in meaning) functioning in different syntactic roles (lexical categories):

Children usually fear the dark.

Children usually experience fear of the dark.

(B.) conceptually different words functioning in different syntactic roles:

He left his father's home to search for the family treasure.

"The Left Hand Of Darkness" is a highly acclaimed SF novel.

(C.) conceptually different words functioning in the same syntactic roles:

John ran to school.

John ran a school.

A flower bed ran along the front of the school.

In the context of the cases and examples thus cited, several general points emerge. Firstly, it appears that all other things being equal, the situation defined by (C.) above is the most difficult

---

* here, and throughout this work, the term 'word' is used informally, and despite discussions of linguists, as equivalent to 'lexical item'.

to deal with. Assuming that the analyser can anchor itself to some low level syntax, then in cases like (A.) and (B.) some advantage can be taken of the fact that apart from exhibiting some meaning (which the system tries to select), the lexical item under question performs certain well known syntactic function. This, however, does not solve all our problems, because, as will be discussed shortly, the possibility of a word functioning in different lexical categories is one of the causes giving rise to the further phenomenon of structural ambiguity. Still, this will (presumably) be dealt with by another mechanism, and as far as the problem of lexical ambiguity remains at issue, it is clear that the bulk of the effort must go into resolving situations like C. above.

No more will be said now about this, except to point out that no matter what the actual disambiguation mechanism is, it will necessarily involve closer examination of the context within a unit of meaning. Which brings us to the second point: without going into details about what should be considered a 'unit of meaning', it is important to point out that 'context' does not necessarily imply a context of a paragraph (with which the system is not extended to deal), or even the context of a sentence: consider the second example in (B.) above, where the relevant context is supplied by the noun group. As a consequence, we should not expect a complete disambiguation to occur immediately within the unit of meaning recognised and analysed, but on the other hand, a hierarchically organised process (see below on the organisation of semantic routines) of lexical disambiguation (which is implicitly and informally suggested by this brief discussion of 'context') might help in delimiting the scope of indeterminacy, and cutting down alternatives before the combinatorial explosion becomes unmanageable. I shall return to this in more detail later, in chapter 3.

The final point I wish to make concerns the decision as to when to introduce a new dictionary definition, i.e. recognise another meaning for a lexical entry. Why should I decide to have a single definition for "independent" and two alternative ones for "short" or "sweet", even though

Is Brazil as independent as the continuum hypothesis?

(example from [McCawley68]) sounds definitely odd, while the implication of "short stick" vs. "short pause"; or "sweet candy" vs. "sweet voice" is that "short" and "sweet" need two distinct definitions. To pursue the point further; if three meanings for "ask", roughly corresponding to "John asked for the book", "John asked about the book", "John asked to see the book", are realised (by the generator at the paraphrase end - see chapter 5) by no less than five different verbs, does not this indicate that there are five, rather than three, separate meanings? These are obviously important decisions to be made even before the detailed design of the NL system is attempted. Here they are simply listed: they will be discussed in more detail later.

1.1.2. Structural ambiguity

The term as I shall use it in this work encompasses the phenomena of both grammatical (i.e. syntactic) ambiguity and transformational ambiguity [Lyons68]. These will be relevant here only so far as they help to establish the causes for, and hence the situations in which, the computer system is again faced with the problem of multiple choice - this time in the process of building the structural representation of a piece of text.

The reason for this multiple choice is the existence of more than one way in which immediate constituents (made available by some recognition process) can be combined into a syntactic structural unit. This clearly indicates that the phenomenon is related to the existence and interpretation of the two kinds of rules normally present in a NL analysis system: recognition rules and structure building rules. In practice these are combined in a single applicative form, because normally analysers are written in such a way as to make sure that whenever constituents are recognised, there are consumers for them: almost all systems (apart from Wilks' - see below) have embodied somewhere in them the fundamental idea of expectation or prediction. The important point about structural ambiguity which follows from this is: its existence and form depend on the particular grammar model that the designer has chosen as the backbone of a particular NL system. So: is it possible to develop a deterministic grammar which altogether avoids the problem of structural ambiguity? Is it possible to develop a grammar in such a way that the nondeterminacy is decreased - i.e. the problem of structural multiple choice is encountered less often.

The view presented in this work is that an affirmative answer can be given to the second question. The properties that a grammar will need to satisfy such a constraint clearly depend on two factors: the particular grammar constituent(s) requiring a certain type of computer response which is externally manifested as facing the problem of multiple choice; and the possible ways of eliminating or minimising the situations in which such a response is needed. These, however, and the general issues raised above, will be discussed in more detail in chapter 3.

Broadly speaking, structural ambiguity can be a function of either the distributional classification of the immediate components of a structure, or of more than one possible way in which these components can be assembled, or of "deeper connections between them" [Lyons68], or of any combination of these. What follows is a closer examination of situations in which multiple choice arises.

(A.) More than one possible way of assembling the immediate components of a structure.

Possibly the most common situation in which this happens is the

problem of establishing the correct structural component being postmodified by a prepositional phrase. Woods refers to the problem as that of 'Selective Modifier Placement' [Woods73] and quotes the classical example of

I saw the man in the park with the telescope.

Even in more subdued contexts, the problem is still there:

Fred returned the book with the pictures.

Fred returned the book with speed.

Prepositional phrases are not the only kind of postmodifiers giving rise to this particular type of structural ambiguity - it is present whenever a certain construct can be attached (no matter how, or when) to more than one potential head. Consider, for example

John admitted to the policeman that he killed Mary by strangling her.

John informed the landlord that he wanted to leave by writing a formal letter.

The difference (in structural terms) between "kill by strangling" in the first sentence and "inform by writing" in the second is obvious.

In situations like these the problem of multiple choice arises when a modifying constituent has been recognised and it can be used either in the same way by quite separate structure-building rules, or in different ways, for modifying different potential heads, by a single rule. The constituents in question do not have to be postmodifying only - for example consider "arithmetic expression evaluation" vs. "efficient expression evaluation".

Situation similar to the modifier placement problem can sometimes occur in connection with conjunctions. For example, a conjunction can be followed by a constituent which could either be attached to more than one preceding potential head, or which could accept as a premodifier the premodifier of a constituent preceding the conjunction: situations some of which are expressed by the following schemata: NP1 prep NP2 "and" NP3; S "that" S "because" S; adj noun "and" NP:

John ordered spaghetti with meat sauce and wine, [Winograd72]

...dangling constituents and prepositional phrases,

John bought the car which Bill was selling because he needed the money,

John bought the car which Bill was selling because he needed a means of transport.

This system does not deal with the problem of conjunctions at all; they were felt to be too large a problem to be tackled along with all the others.

(B.) Distributional classification of the terminal (and/or non-terminal) elements of a structure.

The situation arises as a result of lexical items being classified as belonging to more than one distributional class (lexical category). The occurrence of such lexical items on the borderline of potential well-formed constituents sometimes results in them being accepted as a valid part of either constituent, giving rise to nondeterminacy. Whether we feel it, as in

They can fish, [Lyons68]

John took her grapes,

Bill admitted to the girl that he loves the truth,

or not, as in

Bill admitted to the policeman that he killed Mary,

the problem of multiple choice exists.

(C.) Transformational ambiguity.

As already mentioned, this is a function of the 'deeper connections' between structure components. Without going into transformational grammar theory, I want to point out that a computer based NL processing system faces the problems of recovering these deep connections, and inferring certain 'deleted' constituents or ones absent from the surface text. constituents.

This project has been mostly concerned with the ambiguity of "-ing" participle phrases. For a more detailed account of the phenomenon, and the causes for it, the reader is referred to [Lyons68] and [Quirk72]. Briefly, the situation arises due to the fact that sometimes more than one different proposition generates identical surface strings under nominalisation and/or adjectivalisation. This is the origin of such examples as

Kissing aunts can be boring,

The shooting of the hunters was terrible.

Clearly, the mechanism for dealing with these and similar constructions will be facing the problem of multiple choice - both in the situations quoted above, and in the case of

Shooting elephants can be dangerous,

The stinging of the bee hurt.

Consider also (numbers in brackets indicate possible readings; some of the examples are from [Quirk72]):

I watched Bob teaching Mary (1)

I watched the man teaching Mary (2)

I saw the man teaching Mary (2)

I talked to the man teaching Mary (1)

I listened to the man teaching Mary (1)

I dislike Brown painting his daughter (1)

I dislike the man painting his daughter (2) .

Some other examples of the necessity of multiple choice due to transformations are: ambiguity through ellipsis:

He loves playing tennis more than his wife,

or in the situation of

John is easy to please,

John is eager to please.

To make the problem even more complicated, structural ambiguity can result from the combination of some (or perhaps all) of the factors discussed above:

I heard an earthquake singing in the shower, [Wilks75b]

I heard Mary singing in the shower,

I saw the man sitting in the corner,

...some more convincing evidence [Lyons68].

What is worse, there is nothing in the discussion so far to forbid the manifestation of lexical and structural ambiguity within the same sentence. This gives rise to examples like

Time flies like an arrow, [Kuno62]

I heard her cry. [Lyons77]

Max left Sue to wash the dishes.
[T.Moore - personal communication]

As already emphasised, I am using the term 'ambiguity' to refer to the problem of multiple choice, not to ambiguity as linguistically defined. What this implies is that a computer based NL analysis system must incorporate a mechanism for recognising and dealing with the multiple choice situation. It also implies that linguistic definitions may not be particularly relevant or helpful. The examples given above make it clear that most often than not the decision will have to be made on semantic grounds (for certain exceptions see chapter 3, section 3.6) The exact strategy of when and how much semantic judgement to invoke, as well as the methodology, will be discussed later.

1.1.3. Referential ambiguity.

This project has not been involved with this type of ambiguity at all; the reasons being both lack of time, and the belief that apart from being a research topic on its own right, the problem requires a different approach involving the design and implementation of a higher level deductive component operating on the partial or complete semantic representation delivered by the analyser and performing more general and goal-oriented manipulations than the comparatively limited application of semantic routines. Even though a single sentence environment can be specified, the task will still require inference making processes and mechanisms:

> The soldiers fired at the women and I saw several of them fall. [Wilks73]
>
> The city council refused the women a permit because they feared violence.
>
> The city council refused the women a permit because they advocated violence. [Winograd72]

Actually, the implemented system at its present state does not attempt even any pronoun resolution (not to speak of cases where ambiguity of reference of noun phrases in extended discourse has to be resolved), and as a result phrases as "his monkey" or "their project" invariably get 'paraphrased' into "someone's monkey", or "someone's project".

The fact that referential ambiguity is generally considered to be non-linguistic in nature does not imply that since my system does nothing about it, it is a basically linguistic system. As will be shown later, it makes extensive use of common world knowledge, and so could be extended to attack the problem of this third general type of ambiguity; at present referential ambiguity is ignored because it was not of primary interest at the beginning of the project, and there was no time left at the end to pursue it.

1.2. Prepositions And Prepositional Phrases.

Perhaps the most often encountered cause of structural ambiguity is trailing prepositional phrases acting as postmodifiers. The phenomenon is a function of two factors. First, there is the much discussed ambiguity of the prepositions themselves. Ambiguity here is a particularly complex matter. In my view this is because almost all English prepositions, (if not all of them) are devoid of meaning on their own; basically because there is a great number of varied constructions a preposition can participate in, and it is these that define its 'meaning' in every particular case. To make things more complicated, this 'meaning' as just defined, is expressed in the specific relationship between the main element of a constituent being modified (i.e. its head) and the prepositional phrase itself. This is where the second factor comes: a prepositional phrase can equally well modify the main verb of a clause, and/or any of the heads appearing between the verb and itself. As a result, the number of ways in which a linear sequence of constituents can be interpreted, increases uncomfortably.

Let us suppose that the program has somehow identified the following constituents of a sentence:

V NP1 prep NP2 prep NP3

Generally, there are 4!=24 permutations between the main elements; the assumption that the verb is always the leading one cuts down this number to 3!=6 possible ways of interdependent postmodification. Exploring the English syntactic constraint that forbids a 'crossover' modification (the equivalent of a situation where NP2 modifies V, and NP3 modifies NP1) reduces the number of interpretations to 5. Pursuing the same line of thought in an attempt to generalise the points above, it is possible to show that with 5 constituents under consideration, the number of possible structural interpretations goes up to f(5)=14; and in general this number grows exponentially with the syntactic complexity of the piece of text being analysed: 1, 1, 2, 5, 14, 42, 132,... Clearly, it is not desirable to generate all syntactically valid structures and then attempt to account for them on semantic grounds. This is especially the case where, for example, NP1 cannot be modified by NP2; and consequently all (V (NP1 (NP2 NP3))), (V (NP1 (NP2 (NP3)))) and (V (NP1 (NP2)) NP3) need not be considered at all. On the other hand, any one, or more than one, or indeed all, structural interpretations could also be semantically plausible, and obviously a computer-based NL system must be able to account for them. (My program, for instance, is capable of producing all five (expected) readings of the well known "I saw the man in the park with the telescope": see 0.1)

The discussion above makes it clear that some intelligent and efficient method is required for dealing with prepositional phrases,

as compared with the undesirable blind syntactic nondeterminism. For this, it will be necessary to

a.) adopt some specific parsing strategy;
b.) design an algorithm for efficiently determining the correct level of modification at which a prepositional phrase functions, and attaching it with the relevant relational tie; and possibly
c.) define heuristics for choosing between more than one alternative interpretation (or at least choosing the most probable one). All of this will be discussed in chapter 3.

The problem of proper treatment of prepositions and prepositional phrases is further complicated by the fact that in normal (real world) text, prepositions are sometimes used (semi)-idiomatically, and thus introduce the additional side effect of "shift of meaning": when used with a particular verb different prepositions impose different meanings on the verb, or, alternatively, express a finer distinction of meaning. Consider "aim at" vs. "aim for"; "immune against" vs. "immune to" vs. "immune from". (This is not to be confused with particled/ phrasal verb constructions like: "call off", "give in", "drink up", "catch on"...) And while this situation clearly has to be dealt with by a natural language analysis system, it leads us to the most important fact about prepositions from a computational point of view.

There exist two modes in which prepositions can be used. This however does not imply two disjoint sets of prepositions - just two different modes, to which I shall refer as obligatory or, for reasons explained below, verb dependent, and optional. These perform different functions with respect to the overall understanding of a sentence.

The obligatory use of a particular preposition is imposed by the idiosyncratic requirements of the verb sense intended in the context of the clause. In other words, the preposition, together with the nominal it introduces, makes an important contribution to the verb meaning. In terms of Fillmore's case grammar framework [Fillmore68], the preposition introduces the verb related role that the nominal (i.e. the verb's argument) plays. And the preposition is obligatory, because the deep semantic role performed by the nominal argument is essential for conveying the verb's message. Generally speaking, one cannot just "look" - he is either "looking for", or "looking at", or "looking after" something; if the "defeat" sense of "beat" is intended, usually "beat at" is used; and so on. This, however, is not to be confused with the situation where the preposition introduces an expected, but not an obligatory, slot filler for a specific case frame. "With" traditionally introduces the instrumental case for a verb like "strike", but even if it is missing, "strike" in "John struck Mary" still means much the same as "strike" in "John struck Mary with a ball/ hammer/ feather" as much as essentially the same meaning is conveyed in both cases; the only difference being that in the former the instrument is not explicitly specified, although the person to whom the information is communicated knows (or infers) that some such instrument must have been used. On the other hand, in "John struck Mary as a fool" a specific, and different meaning of "strike"

is intended, and in order to get it across, the use of "as" is crucial, and in that sense, obligatory.

It is the optional use of prepositions which could loosely be associated with the vague notion of the meaning of a preposition. According to circumstances, some prepositional phrases specify certain things about an event - time, place, duration, instrument,... - that are not as central as the essential deep cases; and, further, could be specified similarly for many other events. These are not integral part of the meaning of the verb (or adjective, or noun, for that matter) being modified; but they extend its meaning: thus "in the garden" tells us more than just "the garden". Because of this, even though the preposition still remains one of the most ambiguous parts of speech, certain ranges of its applications can be delimited with some degree of preciseness (i.e. "it can be used in a situation A, but not in situation B"), and generality ("situation", as used above ,is definable in terms of classes and properties of entities that can function as the coordinate arguments of the relation implied in each particular case by the use of the preposition). For example, to quote but a few instances of the varied and general optional uses of "with" [Wood67]:

1. being accompanied by, in the presence of,
2. association or identification on the level of ideas, beliefs, etc.,
3. having, as a possession or as a characteristic,
4. denote an instrument,
5. denote cause or reason,
6. suggest coincidence of time and circumstances; etc.

Thus the use of "with" in

I would like to go to the theatre with you,

I believe with Emerson that a foolish consistency is the hobgoblin of little minds,

...a person with a large fortune,

I managed to beat the dog off with a stick,

The small child trembled with fear,

We rose with the sun,

is optional, because it does not satisfy the intuitive criteria for the obligatory verb-dependent use of a preposition.

These criteria for obligatory use of a preposition can be phrased in a relatively formal way as follows [Ritchie77]:

1. The prepositional phrase will fill an obligatory role in the verb/ adjective meaning,
2. The syntactic properties of the particular verb sense predict that a particular (verb dependent) preposition will be used to indicate the filler for that role,
3. The prepositional phrase occurs after the main verb that creates the prediction.

Ritchie also poses another constraint, which we are not ready to formulate precisely yet: the preposition should cause no change in the semantic structure of the noun-phrase involved.

Once the distinction between the two modes of prepositional use has been made, it becomes clear that a computer-based NL analysis system must be able to distinguish between them and react accordingly. (Unfortunately, as will be discussed later, existing systems account for the first, obligatory, mode only - Riesbeck, Simmons, Ritchie; or the second, optional, mode only - Wilks, Woods).

Two types of distinct mechanisms are required, and it is obvious what and why. At this point I shall only try to establish appropriate frames of reference for these and so define the terms

A. First order (obligatory) prepositional phrases analysis,
B. Default (optional) prepositional phrases analysis.

The final point I want to make here, again to establish a frame of reference, is about the strategy adopted for dealing with the different cases. Two basic approaches are open: I shall refer to them as active and passive. The difference between these can procedurally be exemplified as follows. An analyser, upon encountering and identifying the main verb of a clause, has two courses open. It can consult the dictionary entry for the verb, find out whether it requires any obligatory prepositions, and then actively seek them (and the noun group following) in the rest of the text. Or, alternatively, the analyser can scan the text further, recognise a prepositional phrase, and then globally analyse all relevant factors in order to determine the compatibility and the relation between it and the main verb (or some other preceding constituent).

Pros and cons regarding the two methods; the way this system has decided to treat the problem; as well as its attitude towards the two mechanisms for dealing with the different types of prepositional phrases, will be discussed in more detail later.

chapter 2.
The Basis Of This Work.

2.1. Augmented Transition Networks.

2.1.1. The Basic Model (Woods).

The name of Woods is connected with several quite different issues relating to the computer analysis of written text. These are all embodied in the design and implementation of the LSNLIS (LUNAR) project [Woods72]. In particular, Woods developed the augmented transition network (ATN) grammar as a device for the analysis of natural language, and defined a formalism (language) for representing any ATN grammar in a form interpretable by a separate language processor (interpreter, parser). It is this aspect of Woods' work that I shall discuss most fully here, since the ATN model is in the basis of the work described in this thesis. However, it is important to realise that Woods has separately developed an extensive transition network grammar of English (written in the ATN formalism), a generalised ATN interpreter, the transition network parser, and an integrated NL processing system as a front end to a geological data base (DB).

The augmented transition network model is a development of the basic transition network grammar: a finite state transition diagram with named states connected by arcs, the arcs themselves labelled with terminal symbols, or state names - i.e. a recursion (push-down) mechanism is part of the model. A successful parse is associated with a complete path through the network, starting at the initial state, and terminating at one of the final states of the grammar. The "augmentation" comes from adding to the arcs an open-ended set of structure-building and register-setting actions, as well as test conditions. Thus, named cells (registers) can be assigned arbitrary structures - the results of partial computations - and can, at any later stage of the analysis, be interrogated, thereby determining the further course of analysis, and can also be modified. In the grammar developed by Woods, the structures built were (parts of) transformational grammar (TG) tree structures [Chomsky65], and the tests were, in most cases, conditions on input words. There is nothing in the model, however, to place any restrictions on the nature of tests and operations to be performed during the process of sentence analysis, or on the structures delivered at the end of it. This alone is sufficient to make the model extremely attractive. Together with the efficiency of representation, clarity of expression and simplicity of operation, the ATN model can be a powerful tool for text analysis. These features result partly from the fact that the model is equivalent in power to a Turing machine, and partly from the convenient language defined in which an ATN can be formally represented [Woods70].

The grammar is represented as a set of states, together with the associated names and sets of outgoing arcs. There are several distinct types of arcs, the most important of which are CAT, WRD, PUSH and POP. The first two match terminal symbols in the grammar (by testing for membership of a certain (syntactic) category, or for textual/ literal equivalence); PUSH activates the recursion mechanism and initiates a new (lower) level computation, and POP both specifies a final state in the network, and terminates a computation at a given level - i.e. it must be bracketed with a PUSH: each subnetwork must terminate with at least one POP arc. Clearly, it is the PUSH arcs in the grammar which handle embedded constituents, and if one has been recognised, its structural representation is (usually) returned - by the POP arc - as the result of the embedded computation. All arcs have the same general format:

(<arcname> <arclabel> <test> <action>*),

where 'arcname' is one of the above set, 'arclabel' depends on the type of arc, and is self-explanatory:

(CAT <cat. name> ...) => (CAT NOUN ...)
(PUSH <st. name> ...) => (PUSH NP/ ...), etc.

The actions ('*' means that an arbitrary number of actions can be specified on an arc) mainly manipulate registers; setting them (by SETR), or initialising them at levels different from the current one (SENDR for example, initialises registers for an embedded, lower level computation, before control is actually transferred to the appropriate subnetwork.)

Thus, it is possible to imagine the following directive (arc) being a part of a sentence level network - it expresses an obvious grammar rule:

```
(S/V
   ........
   (PUSH NP/
      (AND (FEATURE TRANS v) (GETR subj))
      (SETR obj *)
      (TO S/OBJ))
   ........)
```

('*' here stands for a special register which always holds the current item - terminal symbol, or result of an embedded processing - which has permitted the arc transition.)

On this basis, Woods has developed the LUNAR system as a front end to a database, enabling a geologist user to ask questions, compute averages and ratios, make listings of selected subsets of data, compare concentrations of elements in different samples, etc., all in natural English. For this purpose an extensive syntactic recognition grammar has been developed, which delivers, as already noted, deep structures based on the TG theory.

This is where questions start to arise, concerning both Woods' approach in general, and the ATN formalism in particular. The ATN

model is highly non-deterministic in nature, and especially where an extensive grammar is concerned, the problem of structural multiple choice becomes the norm, rather than an exception. Woods goes to great lengths in designing a complex and flexible parser, in which various switches allow for various strategies in trying the arcs leaving a state, thus "attempting to find the computation of the abstract machine that corresponds to the 'most likely' parsing in some context" [Woods73]. There are additional features for selecting "most likely" analyses: a mechanism for computing and assigning "likelihood weight" to an analysis path; and a selective modifier placement facility which is intended to solve the problem of parsing postmodifiers. What is not entirely clear, however, is the precise basis on which decisions about the "likelihood" of a parse are taken and how they work in situations corresponding to various types of structural ambiguity (see 1.1). Also, this is the extent to which the problem of ambiguity in general is considered: "a sentence is ambiguous if there is more than one possible accepting path for that sentence". The problem of lexical ambiguity does not exist in LUNAR: "give" always means "get from the data base and print out"; "contain" always triggers a check of some relationship between a sample and its components.

The representation of meaning following parsing is in terms of specialised categories and procedures appropriate to the geological data base with queries interpreted as expressions in an extended predicate calculus. It is not obvious how such data-specific semantic interpretive rules could solve the multiple choice problem in general. On the other hand, it is clear that the representation semantics cannot be extended beyond the tasks associated with the LUNAR system.

Another characteristic of Woods' approach to NL analysis is the serial organisation of his system. LUNAR is a typical example of a system where semantics works in a separate module, called after the completion of syntactic analysis. Woods claims that "it looks as if it takes longer to do the parsing and semantic interpretation ... if the interpretation is done during parsing" [Woods73], and I have strong objections to this (see chapter 3). Apart from the discussion as to which takes longer, the very nature of the problems we are trying to solve here, for the general language case, naturally suggests a heterarchical organisation of the syntactic recognition and semantic interpretation routines (this is also Winograd's approach [Winograd72]). Woods' strategy is to find all syntactically valid parsings, pass the first one on to the semantic interpreter, and the syntactic alternatives are left untried unless the interpreter does not like the first one. However, this relies implicitly, and to a greater extent than is justifiable, on the order of the arcs leaving a state*, and much of the time spent in syntactic recognition is devoted to constructing parse trees that will never be used.

Furthermore, the ATN model is essentially an expectation/ prediction device: it expects a certain type of constituent at a certain

---

* if it was possible to specify a certain ordering of the arcs, it would be possible to write a grammar for which an interpreter with a deterministic control structure would suffice - which is not the case - see below (chapter 3).

position of the input string. It is only natural to attempt to extend this idea, and on the basis of some generalised analysis of context, to predict not only a certain type of syntactic constituent, but a constituent with a certain semantic content, or, at least, certain semantic properties. Clearly, this again suggests that semantics should work in closer coordination with the syntactic recognition. Unfortunately, the original ATN model offers no explicit provision for this; it has been designed primarily as a syntactic recognition and structure building device. Part of my research, therefore, has been concerned with the question of how semantic judgement could be incorporated within the traditional ATN framework.

2.1.2. Semantic ATNs (Simmons).

An explicit attempt to combine the ATN model with some semantic processing has been made by Simmons [Simmons73]. A specially developed ATN grammar is used to build directly a "semantic network" during the analyser's scan through the sentence. Verbs have associated with them "paradigmatic ordering rules" and these are consulted by the structure-building actions on the arcs to determine the "deep semantic cases" of surface nominal structures. However, there are certain problems with Simmons' approach. The networks are somewhat unorganised, in that all sorts of heterogenous information is stored at a node, in the hope that sometime later it might be useful; and they are nevertheless not really "semantic" in content. The only semantic operation that Simmons' parser performs is done by the ARGEVAL function which decides which of the possible arguments (surface nominal structures) to fit into a particular slot. It is not made very clear how ARGEVAL selects the relevant paradigmatic ordering rule, especially in an ambiguous environment (the rule must be attached to a verb sense, rather than to the surface verb - see below: 3.9, 4.2) - in fact, possibly this is the reason for Simmons' extreme view on lexical ambiguity: "varied sense meanings of a verb can be accounted for as varied implications of a given word class". I shall discuss this in detail later (3.6), but such a view is clearly inadequate for our purposes. No explicit treatment is offered for structural ambiguity as well: ARGEVAL expects all possible prepositional phrases (or other obligatory postmodifiers) to be mentioned explicitly in the paradigmatic rules, thus requiring them to specify in advance the correct order, syntactic environment and semantic details of the expected "deep semantic classes". Default treatment of optional modifiers is not considered, neither are other cases of structural ambiguity (see 1.1). Possibly part of the problem with Simmons' approach comes from the fact that his semantic structure building process is not set in a sound framework: the structures are being gradually built on the arcs of the grammar, and this (as will be discussed later in more detail: 3.3, 4.2, 4.3; see also [Ritchie78]) is quite difficult. In any case, Simmons' attempt to construct a semantic ATN just scratches the surface of the problem.

Before going on to discuss my approach to a semantically based ATN processor, I shall outline two other natural language analysis projects, predominantly semantic in nature, which have influenced this work to a large extent: the semantic parsers of Wilks and Riesbeck.

2.2. Semantically Guided Parsing.

This idea lies behind Winograd's SHRDLU system [Winograd72] which is significantly more intelligent when it comes to dealing with (some kinds of) structural ambiguity than the systems considered so far. When faced with the problem of multiple choice, a semantic specialist is called, with control eventually handed over to a deductive component which applies semantic and environmental knowledge relating to the system's world model (a closed world of multicoloured blocks, boxes, and pyramids) and thus chooses the correct structural interpretation. There is nothing wrong with this in principle, but to a large extent the system is only able to work in this way, because SHRDLU is essentially a theorem proving program which works on top of fixed and rigid axiomatic knowledge base. There are certain problems of understanding that SHRDLU never attempts to tackle; worse - it offers no clues how they could be solved within the proposed framework: lexical ambiguity is one of them. I shall not discuss Winograd's system any further. Instead, I shall concentrate on some attempts to build general text analysers, where the parsing process is guided to a large extent by some sort of semantic judgement.

2.2.1. Semantic Pattern Matching (Wilks).

Wilks' complete system, as implemented at Stanford, and described in [Wilks73b], [Wilks75b] is aimed at machine translation, though I shall not discuss the generation component here. The whole approach is quite distinctive and unique: Wilks makes an explicit attempt to deal with the problem of multiple choice in general as it faces a 'blind' natural language processor; the analysis process is entirely semantics based, in that no (explicit) syntactic recognition is performed at all; the foundation of the system is the concept of 'preference semantics'. This is an underlying principle, rather than a single applicative rule embodied somewhere in the implementation. Apart from being the driving force behind the various disambiguation techniques, the notion of 'preference', as opposed to 'strict requirement', allows Wilks to extend the range of constructs handled by his system to include metaphors and other instances of extended use of language.

Semantic knowledge in Wilks' system is distributed among several types of structures. At the lowest level there are some 80 to 100 atomic semantic primitives; these are the basic building units for semantic formulas, one formula for each word-sense, which are constructed and interpreted subject to a rigorously defined syntax (see 3.1). Next up in the hierarchy of semantic structures come the bare templates. According to Wilks, there exists "a list of messages that people want to convey at some fairly high level of generality...The bare templates are an attempt to explicate a notion

of non-atomistic linguistic pattern". It is the task of the program to find the underlying message for a piece of text and elaborate it by substituting the generalised semantic markers in the ACTOR, ACTION and OBJECT positions of the matching bare template with the semantic formulae of the corresponding words. The task of delimiting the template boundaries lies with the fragmentation routine (FRAGMENT) which breaks the original sentence into smaller chunks of text. The actual template matching (PICKUP), as well as the establishing of further dependencies between the main components of a template and the remaining words in a fragment (EXTEND), are very flexible processes which rely extensively on the notion of preference. The fragments of surface text are expanded into (tentative) semantic structures (full templates), their number depending on the possible combinations between the word-senses semantic formulas corresponding to the surface words. Links are established between the main elements of the template ("John drinks water"), as well as between them and possible dependents ("sly fox", "jumps quickly",...) by consulting both the individual semantic formulas: "drink" expects, preferably, *ani subject and (flow stuff) object+; "sly" can qualify *ani entities. etc., and the list of bare templates: [*pot cause *ent] is a valid bare template onto which "John drinks water" is matched. Both the number of links thus established and their connectedness define the 'semantic density' of an interpretation of a fragment. An applicative rule such as "densest match wins" selects, or prefers, the most plausible interpretation. Since the full template contains semantic formulas corresponding to individual word-senses, possible disambiguation results from this procedure: consider Wilks' example

The policeman interrogated the crook.

where two possible templates (corresponding to man force man and man force thing) can match onto the bare template *pot force *ent; and it is the preference restrictions on the formula for "interrogate" that finally establish the winning link:

((man subj) ((man obje) (tell force))).

It is important to realise that "preference is always between alternatives; if the only structure derivable does not satisfy a declared preference, then it is accepted anyway" [Wilks72b]. This is the basis for metaphor handling, and to my knowledge, is so far the only attempt to incorporate in a NL analysis system the ability to deal with instances of extended language use.

The same semantic principle underlies the next stage of processing (TIE): the integration of the individual representations of different fragments into the final form of Wilks' interlingual representation (a semantic block), which is then handed over to the generation routines (after possibly being processed by 'common sense inference rules' - directed basically at pronoun resolution). To aid this process, another type of semantic structure is utilised: the paraplate. Paraplates are stored in the dictionary under the keywords which aid

---

+ expressed in terms of Wilks' primitives, or classes of these: *ani (animate) = [man, beast, folk], *pot (potential actor), *ent (entity), etc.

the fragmentation process, mainly prepositions, and specify predicate functions over skeleton templates - governor and dependent. In case two templates in the fragment satisfy the predicates thus specified they are linked by a case link which is also suggested by the paraplate. All possible links between combinations of template pairs are established, and then the resulting semantic structures are examined by the same density techniques as were applied one level down, within the individual fragments.

Wilks never really makes clear what the population of alternative template fragments for the texts is that his system handles: but, for the preference semantics density checks to work, a choice between alternatives must be offered. The question is what is the point beyond which the handling of alternatives becomes uncomfortable and the choice procedure starts to choke? Possibly the breaking of the parsing process into a series of passes over the text: fragmentation, template match, expansion, paraplate match, is intended to solve the problem to a certain extent. This, however, raises some further issues. The communication between the consecutive passes is via a representation which is a collection of templates; indeed, every language construct is reduced to one or more templates. For Wilks this is justified by the fact that the template is the basic building block of the interlingual representation. But, in practical terms this means that the bare templates are expected to perform too many functions: they are a parsing device, they carry the basic semantic content of a fragment of surface text, they are a disambiguation device, they are a skeleton semantic structure, a blueprint for the final semantic representation. Under these circumstances the templates become too general and inevitably lose in resolution power (see below: 3.6, 3.9). This is made up for to a certain extent by the consecutive stages of processing, at the cost of introducing unnecessary parallelism, as already noted. Furthermore, the requirement that the template in its strict form of an ordered triple with a simple actor-action-object relational structure should be the only semantic pattern, imposes restrictions on the organisation of the semantic density checks - as it is, these do not necessarily operate within the natural limits of a linguistic unit.

Clearly, the problem could be solved by introducing some low level syntax - but then the justification for the fragmentation process comes into question: Wilks claims that no syntactic preprocessing is performed, but FRAGMENT is essentially a syntactic recognition box, its operation guided by syntactic features. Even then, it is not entirely clear what the exact nature of the processes is which will, in particular, fragment texts like the following:

I heard an earthquake ¦ singing ¦ in the shower,
I heard ¦ an earthquake sing ¦ in the shower [Wilks75b],

and, in general, will deal with the extended situations of structural ambiguity (types B. and C. - see 1.1).

What is clear, is that Wilks' system has got nothing to lose from the introduction of some explicit syntactic recognition mechanism; we might as well admit the fact and make good use of it. This is another

of the basic ideas behind this project, and it will be elaborated in detail, in chapters 3 and 4. It remains to be said here that although the work of Wilks has greatly influenced my own research (see 3.1), there are some substantial differences between the two respective approaches, which will become clear in due course. The aspects of Wilks' approach to natural language analysis which have been adopted here are the general strategy of semantic pattern matching, the underlying concept of preference semantics, and the set of semantic primitives, together with the formalism for constructing semantic formulas over wide range of meanings for surface words. Consequently, this work does not make all the claims explicitly or implicitly made for Wilks' system. "Understanding without proofs", "preferential, pattern-seeking semantics for natural language inference", "making preferences more active" are issues which are not discussed in this paper at all - the emphasis is on the analysis of NL text and the problem of multiple choice in particular, viewed in the light of establishing the proper interaction between syntactic and semantic processing; however, the system is designed in such a way as to allow for further extensions.

2.2.2. Semantic Expectations (Riesbeck).

Riesbeck's parser [Riesbeck74] was originally developed as an integral part of Schank's MARGIE system (Memory, Analysis, Response Generation and Inference on English) [Schank75], and is, in essence, a program which maps English sentences into Conceptual Dependency (CD) structures - a primitives based, language independent, general purpose representation of meaning as developed by Schank. Schank's theory has the ambitious aim of serving as a model of human language understanding; consequently the view presented by Riesbeck is that the parser should make predictions as to what is left of the (expected) conceptual image to be delivered by the message, and actively seek this information in the incoming text - just as people do. In doing so, the program concentrates on manipulating conceptualisations, which are essentially semantic objects; "other tasks, such as discovering syntactic relationships, are performed only when essential to decisions about meaning" [Riesbeck74]. Riesbeck rejects pattern-matching as a parsing mechanism; the basic device in his system is the expectation. Situations that might happen later in the course of parsing are anticipated. The main activities during the process of analysis are the building of partial (skeleton) structures, filling gaps in these as more information becomes available, and integrating them into larger conceptualisations. The process itself is guided by requests - test-action pairs attached to each word; the actions are performed only if the current state of analysis fulfils the conditions associated with the anticipated situation. The program maintains a list of active requests which are constantly checked and rechecked, and the way in which conditions become satisfied as new words of the text are scanned determines the overall flow of control.

Although requests are attached to each word in the dictionary, most of the work is done by the pieces of program associated with the

verbs. Therefore, the identification of the main verb is crucial to the whole process - it provides the skeleton conceptualisation corresponding to the clausal unit and sets up the overall contextual expectations. This is a very sound approach in principle (see 3.8), but a closer examination of the nature of the requests reveals that they are, in essence, questions whose objects are syntactic entities. A test such as "is the current NP an action?" implies a command conceptually equivalent to (PUSH NP/ ...), or (PARSE NP). In other words, each verb entry has to specify its expected syntactic environment. This is both difficult, because an exhaustive syntactic environment corresponding to the different uses of the verb is difficult to provide, and inefficient, because similar surface syntactic patterns have to be specified separately for different verbs (in contrast to the ATN's capacity to merge such patterns into common paths through the network).

However Riesbeck cannot have a packet of requests, one for each verb-sense, because this implies that as soon as a verb is encountered, its intended sense must be identified so that the appropriate request packet can be activated; and Riesbeck's philosophy is that pending alternatives is not allowed. Instead Riesbeck distinguishes what he calls 'sense' from what he calls 'meaning'. For him the 'sense' of a verb is characterised by the whole packet of data and program associated with the verb; the different 'meanings' are separated by local context tests within the packet: is the object of "break" a physical object or an obligation ("break a chair" as opposed to "break a promise"). This is in effect applying Katzian selectional restriction rules [Katz64], and is, in general, as far as Riesbeck's implementation goes towards solving what I have called the problem of lexical ambiguity (1.1) (for Riesbeck's word 'meanings' are my word 'senses'). As for structural ambiguity, obligatorily used postmodifiers are dealt with by the individual requests; Schank [Schank73] hints at a general strategy for dealing with optionally used prepositional phrases, which, however, is open to criticism (see 3.10 below, also [Wilks76]). Satisfactory uniform approach towards dealing with structural (or, as Shank refers to it, syntactic) ambiguity is difficult to sustain in the face of such assertions as: the parser should be deterministic because thus it will model most faithfully the process of human comprehension of texts, and, no syntactic processing should be done beyond the bare minimum; for these assertions may imply inappropriate forced choices. All these issues are discussed in greater detail later in this work; it is only in connection with the last point that some further questions are raised here.

These concern the place of syntax in the overall system structure. Riesbeck's claim that syntactic relationships are sought only when they provide pointers to conceptual information seems rather vacuous because syntactic processing in his system is used more than he cares to admit. It is true that "a decision about meaning" can be made after the semantic content of noun, prepositional, or any other, phrase is analysed within the global context, but before that the phrase itself has to be recognised - which is clearly a syntactic operation, triggered by syntactic prediction.

There are not so many common elements between Riesbeck's approach to natural language analysis and mine, and what there is in common, is on a very general methodological level: the verb is central to the clause, analysis should be performed with regard to the overall context of a textual unit, and so on. The CD parser has been outlined here because it is the most widely known representative of the active parsing strategy (as opposed, for example, to Wilks' passive parsing), and as such, establishes a frame of reference for further more detailed discussion.

chapter 3.
Background To The Analyser Design.

3.1. Choice of Semantic Theory

This project is strongly influenced by the work of Wilks (Stanford Artificial Intelligence Project: [Wilks73b], [Wilks75b]) which placed primary emphasis on semantic processing in language analysis. Apart from acknowledging the fact that his approach is one of the most distinctive (and promising) in the field of NL analysis, I was particularly attracted by his theory of "preference semantics". It seemed in the beginning of this research work, and was confirmed by the overall system performance towards its final stages, that the theory is

a. flexible enough to withstand adaptation - i.e. to be embodied (with possibly not major modifications - the reason for these can be found in the general comments on Wilks' system in chapter 2; and will be discussed in detail in the following sections) in a framework methodologically different from his own, and

b. general enough to still retain its semantic judgement and resolution power.

Some features of Wilks' approach are especially attractive and potentially useful: the set of semantic primitives, and the formalism for assigning a 'meaning' for each sense of a word by constructing a semantic formula subject to strictly defined rules, which is obviously a necessary prerequisite for any system attempting the task of lexical disambiguation. Provided these features could be embodied in a general framework capable of imposing a more rigorous structure than Wilks' templates and semantic blocks on the result(s) of the analysis, it would be possible both to attempt the resolution of structural ambiguities, and to concentrate on the content and format of the intermediate semantic representation and its subsequent use, in this case for generation.

The nucleus of Wilks' semantic theory, the construction of semantic formulas which correspond to the different meanings of a word, using a basic inventory of around 100 primitive semantic elements, can be briefly outlined as follows. The outline is intended only as a guide sufficient to support my own application of these ideas; for more details the reader is referred to [Wilks73b], [Wilks75b] and especially in [Wilks77].

3.1.1. Semantic Formulas.

In op. cit. Wilks defines a process for formalisation of the meaning(s) of a lexical item, resulting in a list of semantic formulas - one to each distinguishable (see below) sense. For example, the dictionary might contain two definitions for the verb "grasp":

grasp1 ("grasp the block"):

```
   ((*ani subj)
      ((*physob obje)
         (((this (man part)) inst) (touch sense))))

grasp2 ("grasp the idea"):
   ((*hum subj) ((sign obje) (true think)))
```

and two definitions for the noun "crook":

```
crook1:
   ((((notgood act) obje) do) (subj man))

crook2:
   ((((((this beast) obje) force) (subj man))
      poss) (line thing))
```

It is clear that a (semantic) formula like any of the ones above is more or less naturally interpretable in English: "grasp1" is an action of tactile feeling, preferably done by an animate agent to a physical object, using a (body) part of the agent. Similarly, one of the meanings for "crook" is "a man who does bad acts".

However, it is important to stress that the semantic expressions are

a. language independent,
b. subject to rigorously defined syntax.

The latter property transcends the boundary of the simple discrimination and categorisation of a lexical item and acts as the basis for (developing) an unambiguous semantic representation [Sparck Jones65]. Viewed in the light of the former property (point a.) such a representation is the basis for the claim made in the second half of this work, namely that the generator developed can be used equally well both for translation and paraphrase (see chapter 5).

The formulas in the dictionary are trees of left to right dependencies relating individual elements or whole subformulas, where a subformula is a left-right pair whose two members are either a primitive or another subformula. There are seven distinct types of these subformulas: assertion (full clause), transitive action, action, case, nominal (substantive), qualifier (adjectival), and adverbial subformulas. Each of these is associated with groups of semantic elements which can be used to construct the left and right elements of the pair, and a set of rules as to how the dependency between the elements of a pair should be interpreted depending on the type of the subformula and the groups categorising the participating semantic primitives.

These groups of primitives are:

action elements: feel, give, make, tell,...
substantive (nominal) elements:
man, act, thing, part,...
nominal classes: *hum, *physob, *inan, *pla,...
case elements: subj, inst, goal, poss,...

qualifier+ elements: true, well, same, when,...
It must be noted here, that for Wilks, the semantic primitives are useful classificatory notions, not necessarily absolute; the (best) justification for their introduction and use is the fact that they work.

In addition, there are certain standard subformulas: (flow stuff) denotes liquids, (thru part) aperture, (get sign) any monetary unit, etc: these function more or less as if they were primitives.

Generally speaking, a semantic formula is a collection of binary bracketed, but hierarchically interpreted case subformulas. These can specify the preference restrictions (see chapter 2) on other items of the lexical item being defined (which are not to be confused with the selectional restrictions of Katz and Fodor [Katz64], [Wilks72b]). The rightmost element in the semantic formula is called its 'head' and serves to give a first approximation to a general semantic characterisation of the lexical item. Case dependents can alternatively be other propositions (full assertion or action subformulas), which within the context of the whole formula further express and define the meaning of the word sense to which they are attached. It is clear that in theory there is no limit on the depth of the semantic formula - and this is an advantage of Wilks' system of semantic coding. It is also clear that the actual depth of detail will be determined by the chosen level of representation which depends on the required system performance and overall objective (this is discussed in more detail in 3.11). As Wilks puts it, the process of encoding a word sense into a semantic formula "assumes a common sense distinction between explaining the meaning of a word, and knowing facts about the thing the word indicates. The formulas are intended only to express the former, and to express what we might find in a reasonable dictionary though in a formal manner" [Wilks73b]. Needless to say, this "common sense distinction" implies that a complete natural language analysis system will need mechanism(s) for providing, accessing, and utilising another type of knowledge: facts about the world. This is the function of Wilks' inference rules [Wilks75], and, more recently, pseudotexts [Wilks78]. My particular system does not go so far in the processing of text as to need such information since it concentrates, insofar as this can be done with no or little real world knowledge, on the syntactic and semantic processes involved in constructing an unambiguous internal representation, and their interaction. Still, it is important to acknowledge the fact that any (non-trivial) further extension of the system will need to have such knowledge incorporated in it. It is also important to design the system in a way which makes sure that it is possible to extend it in this direction, and there is no reason to suppose that Wilks' type of formula cannot be used in this way.

Two further points related to Wilks' semantic primitives and coding formalism are relevant (and important) to my project. No

---

+ these can qualify: actions (true, well, ...), specific actions (hear, see, touch, ... sense), substantives (many, same, notsame), or some specific substantives (when, where,... point).

restrictions are imposed on the dependents of the semantic primitives, as are imposed by Schank for example. This is in contrast to the view that after surface verbs are related to the underlying primitive actions, all the participants in their case frames should be obligatory [Schank73]. As noted by Wilks [Wilks76] a relaxation of this requirement makes the formulas very flexible, and in particular, leads to increase in their expressive and resolving power, thus avoiding the failure to discriminate between some (quite different) verbs. In addition to this, Wilks is (to my knowledge) the only researcher in the field who has developed a theory which allows semantic representation on a more detailed level not only of verbs/ actions, but of nominals/ objects and adjectives/ qualifiers: Schank's approach, for example, is quite inadequate here. Wilks provides for both concrete and abstract nouns (such as "father", "gun", "decision", "proposal") and goes beyond the (relatively) simpler process of attaching features to nouns (human, inanimate, liquid,...) or classifying adjectives on a STATE (VALUE) scale as Schank, for example, does.

The formalism for encoding word senses into semantic formulas, together with the basic principles of 'preference semantics' ("preference is always between alternatives; if the only structure derivable does not satisfy a declared preference, then it is accepted anyway": [Wilks72b]) are the only aspects of Wilks' NL processing system I have taken over without major changes. Their application in parsing differs substantially from Wilks' fragmentation and template matching. As indicated earlier, and will be discussed in more detail below, my system is based on syntactically driven text analysis and semantic routines organised around verb contextual frames (see 3.9). Semantic formulas and the intermediate products of the semantic processing are incorporated into an unambiguous internal representation in the form of a hierarchically organised dependency structure which is only a distant relative of Wilks' semantic blocks.

The specific ways in which this project departs from the approach presented in [Wilks73b], as well as the reasons for these, are described below.

3.2. Choice of a Mechanism For Parsing

The choice of the augmented transition network (ATN) formalism as a parsing mechanism was motivated both by pragmatic considerations and by the history of my project.

The work in which I was initially interested was an attempt to develop and implement an efficient purely syntactic ATN parser and to test its recognition and structure-building power for certain classes of syntactic structures. The power and potential of ATN parsing were very attractive, but Woods' original purely syntactic application of the idea appeared very limited from the point of view of language processing as a whole. The ATN model imposes a specific representational form on a natural language recognition grammar - the equivalent of a finite machine with a high degree of branching - with the result that parsing is highly non-deterministic, especially when it is confined to syntactic processing without reference to any semantic or pragmatic factors. This wasteful and unintegrated processing is both practically and theoretically unsatisfactory, and the question of how to incorporate semantic judgement within the framework of an ATN parser became one of the central issues of this research.

Now as the ATN formalism has the full power of a Turing machine, it is computationally equivalent to many other formalisms. It is implicitly equivalent to, say, Wilks' fragmentation routine, or Riesbeck's parser. This latter is, in effect, a disguised ATN, with the current 'situation' (see [Riesbeck74]) formally equivalent to a state in the network; the transitions between states are determined by the expectational devices of 'requests' which specify the tests and actions required to set up new situations. As was noted in 2.2, Wilks' and Riesbeck's parsers are predominantly semantic ones; thus it could be argued that some (admittedly implicit) integration between semantic judgement and ATN parsing has been achieved. Nevertheless, it was still an open question how easily semantics could be openly handled by such an extremely powerful but very general formalism, which, even though seeming appropriate to the task of computational analysis of natural language, had been extensively used for developing predominantly syntactically based natural language programs.

On a completely different level, the ATN mechanism as originally proposed by Woods, has features which seemed particularly useful in relation to the linguistic phenomena on which this project focusses, as well as generally helping for the design of a versatile and efficient analyser.

The most important of these features is the ability to assign partial analyses or the results of the recognition process to registers and later interrogate these. The effect of this is twofold. Decisions can be delayed and so can the process of structure building,

until enough information is accumulated to make it feasible to attempt disambiguation. In this situation the registers provide the global context which is crucial to the lexical disambiguation; they also collect and hold the immediate components of a constituent until enough (at least the bare minimum) are available to guide the structure building decisions resulting in the semantic representation of the currently analysed constituent. This can be regarded as a semantic extension of the wait-and-see approach [Marcus75] i.e.passive analysis based on semantic pattern matching, as opposed to the predict-and-possibly-go-wrong active alternative. Clearly the wait-and-see approach implies a bias towards a particular parsing strategy, which will be discussed in more details below (3.7).

One of the major problems inherent in a more conventional dynamic pattern matching approach to parsing is the presence of certain features, as well as the semantic content of the constituents, that the pattern-matcher (analyser) should be sensitive to, rather than the surface ordering. Clearly, in this situation, the ATN registers, which allow one to wait and see, can be of great value.

As noted, Woods' original ATN application was to an extensive grammar of English covering a wide range of syntactic constructions. The grammar itself is irrelevant to the purposes of this discussion, but its existence and successful use in the LUNAR project [Woods72] show that the ATN formalism can conveniently capture the regularities of a non-trivial subset of (English) natural language. Thus we can be sure that the choice of ATN as a parsing mechanism will not be an obstacle to the designing of a versatile analyser insofar as the analyser invokes conventional syntax. It has been suggested that the ATN model, while very good at noun phrase level, is not adequate enough for dealing with clauses. My view is that this judgement relates basically to the process of syntactic structure building, rather than to the actual recognition; and it is not an objection to my approach since my analyser does not attempt to construct purely syntactic intermediate structures.

A final advantage of the ATN formalism is that it encourages efficiency in the recognition process, since it allows a combination of the two parsing strategies: top-down - by means of the PUSH/ CAT/ WRD arcs; and bottom-up - by means of the tests on the arcs. The efficiency comes from being able, without much fuss, to look for the right constituent in the right place.

The discussion so far makes clear the reasons behind the choice of parsing mechanism. The next section begins to answer the question raised above: how much semantics can be incorporated within the traditional ATN framework.

3.3. Purely Semantic Networks

On the basis of the choices outlined in the last two sections, the extreme approach was attempted initially: designing a purely semantically driven ATN which should be able to handle Wilksian "preference semantics" structures. This seemed feasible because on an abstract level the ATN formalism is simply a device for specifying a set of recognition (and structure building) rules, and is independent of the ATN interpreter; and a primarily semantic rather than syntactic method of analysis seemed a plausible approach to natural language processing.

Two approaches were tried before enough evidence against their feasibility was accumulated. These will be discussed in some detail below as they proved to be useful experience in their own right, and influenced the further development of the system: it will be noted later that the general strategy adopted here represents a salient difference to a purely semantic approach.

The first of these was directed towards elaboration of the "template" i.e.message form idea of Wilks, leading to a paraphrase of his fragmentation - template matching procedure as an ATN operation: look for an entity in ACTOR position, look for an entity in ACTION position (plus possible modifiers and optional constituents), and so on. Bearing in mind the possibility of various qualifiers on semantic referents, and embedded clauses and clause adjuncts, an extended network at top level (the basic message triple) will, necessarily, initiate "subroutine calls" to auxiliary subnetworks. The "prediction" arcs of the ATN - CAT and PUSH - should be governed by the primitive semantic elements of Wilks, rather than by syntactic categories.

There are some immediate problems the foremost of which concerning the "entity" referred to above - what is it; and what is the common denominator justifying the labelling of the underlined phrases as ACTOR in

(*) The man killed last night was identified by ...

(**) To disregard common sense rules is to beg for trouble?

Is it just that both appear in the SUBJECT position of the sentence and can be referred to as noun phrases (in some general sense)? Next comes the issue of recognising and labelling them - (*) can be referred to as a "man", and (**) as an "act". Different labels (which are necessary for the template matching procedure) suggest different subnetworks; on the other hand there is no justification for introducing separate "man" and "thing" procedures to account for "the man killed last night" and "the car stolen last night" - they are both noun phrases. This has been pointed out by Ritchie in [Ritchie78] who suggests that semantic generalisation in this sort of approach comes

naturally after an initial recognition step, which as the discussion implies, is syntactic in character.

On the other hand, even assuming that some sort of semantic prediction could be embodied in this sort of approach (though it is not very clear how this should be done, because predictions are normally set up by the verbs, and thus naturally belong to the dictionary, rather than the grammar), there is still the problem of low level recognition. Consider for example "predict", and for the sake of argument let us assume that it expects an "event" class. This will be true in both cases:

He predicted the general election,

He predicted that there would be a general election.

thus posing the question of specification of how these different surface constructs are to be specified in terms of semantic primitives only.

Similarly, predictions can be used for specifying instructions like (PUSH LOCATION/ ...), (PUSH DURATION/ ...), etc., but in the light of the obligatory use of prepositions (see 1.2), some of these instructions will have to be decoupled from the grammar, and placed on the individual verb entries in the dictionary. This has the unpleasant effect of splitting the grammar, without helping much.

Adding to these difficulties there are some other minor, but still subtle problems, such as "do"-support, special uses of "have" and "be", questions and passives, existential "there". All this suggests that the desire for a set of recognition rules oriented towards realistic texts will require some syntax - possibly disguised in one form or another: we may call a verb an ACTION, an adverb a HOW, and an embedded clause an ACT or EVENT - but we are nonetheless using conventional syntax.

In the other approach an attempt was made to rewrite Wilks' inventory of bare templates as an ATN i.e. to incorporate the templates into the ATN rather than simply look for ACTORS, ACTS and OBJECTS. Arcs will again be governed by primitive semantic elements (or classes of these); the set of basic semantic messages will be mapped on sequences of arcs: in a generalised ATN notation (omitting the tests on the arcs and the register setting and manipulation) the subset

```
.........
*ani feel    *mar
*ani think   *mar
*pot tell    *mar
*pot force   *ent
*pot want    *ent
*pot give    *ent
*pot strik   *physob
*all please  *ani
.........
```

will be specified as

```
(TEMPLATE
   (PUSH *ani (TO AGENT1))
   (PUSH *pot (TO AGENT2))
   (PUSH *all (TO AGENT3)))
(AGENT1
   (CAT feel (TO OBJ1))
   (CAT think (TO OBJ1)))
(AGENT2
   (CAT tell (TO OBJ1))
   (CAT force (TO OBJ2))
   (CAT want (TO OBJ2))
   (CAT give (TO OBJ2))
   (CAT strik (TO OBJ3)))
(AGENT3
   (CAT please (TO OBJ4)))
(OBJ1 (PUSH *mar (TO END)))
(OBJ2 (PUSH *ent (TO END)))
(OBJ3 (PUSH *physob (TO END)))
(OBJ4 (PUSH *ani (TO END)))
```

At first sight this seems to be fine, but a closer investigation of the PUSH concept poses certain problems. The desire to design the "*pot" subnetwork in such a way as to be able to recognise both (*) and (**) above, will reduce it to something which looks suspiciously similar to the traditional NP network (with the additional constraint of rejecting certain referents: "point" cannot belong to the class of *pot; "act", "thing", etc. are not *hum, and so on). This is exactly the problem we had with the previous approach, and must be bypassed if we insist on keeping the ATN purely semantic. The alternative is to break down the *pot class into its components: "man", "thing", "stuff", "act", etc. Forgetting for the moment the issue of justification for introducing separate subnetworks for each of these ("father", "car", "water", "play"), we are faced with the problem of categorisation of a lexical item into overlapping semantic classes: where does "boy" belong - to *pot or *ani? The answer is clearly both:

The boy wanted ice-cream;

The boy felt pain.

(because these will match onto different templates - see the list above).

In theory this should not be frightening: by analogy, in the traditional syntactic version of network grammar, "call" can be a noun or a verb. The ATN formalism is non-deterministic by nature and a bit more should not hurt. But the point is that due to the fuzziness of classes of semantic primitives and the degree to which they overlap, as well as to the general polysemy of words, a very highly non-deterministic environment is created, with the unwelcome effect of a proliferation of alternative templates, to an extent which (almost) defeats the purpose of the whole exercise. As a side effect there is

the problem of establishing preferences in such a highly non-deterministic environment, and consequently that of the presence of common analyser subpaths requiring separate treatment because they belong to different alternative templates. The non-determinism is further increased by the fact that similar surface phenomena can be associated with more than one physically detached state in the ATN: imagine extending the network to handle indirect object constructions which are associated with the primitive actions "give", "tell", "get"; or copula constructions - associated with "be", "feel", "cause"...; or the most general case of postverbal modifiers. The network equivalent of a paraplate match, no matter whether implemented in a passive or active form (generalised PUSH to a PP subnetwork with all paraplate tests spelt out in a highly branching structure, or explicit search for certain postverbal constructs keyed on the particular action or surface verb) will introduce further non-determinism due to the already mentioned fuzziness of semantic classes, and thus make the problem of structural disambiguation very difficult indeed. The lexical ambiguity of words does not help either - what is "crook" - "thing" or "man"?

In such a highly non-deterministic environment relatively simpler issues become problems on their own: there are difficulties in isolating the template components and template boundaries. Also, it is not very clear how to handle relationships between the templates constituting the input sentence as a whole.

Some of these difficulties could be slightly reduced if the main action element is known, but in a sequential scan of the text it is usually some time before it is known.

A major remaining problem, however, for the attempt to build a purely semantically driven ATN is the embodiment of the applicative interpretation of preference semantics basic principle: prefer the normal, but accept the unexpected, if no alternatives are available. It is difficult to imagine what would be the network equivalent of a dynamic creation of a new template - "my car drinks gasoline" for example.

Ritchie, in [Ritchie78] argues that due to the hidden ambiguity of the PUSH concept - it specifies both a "goal" and "means" to achieve it - the proposal that the ATN formalism could be adapted for purely semantic processing does not seem viable. This is only one of the problems. The discussion above, related to a particular environment (3.1 and 3.2) clearly shows that the idea collapses even before getting that far, since an ATN based on preference semantics alone would be extremely clumsy and inefficient, offering no gains whatsoever.

Speaking with some degree of generality, a semantic network implies writing a parser based on semantic expectations*; thus Riesbeck's parser is organised along these lines in certain respects.

---

* because the ATN is essentially an expectation/ prediction device: see chapter 2.

This raises the issues related to active and passive parsing approaches in this particular context: semantic expectations vs. semantic pattern matching. With the same degree of generality, two major drawbacks can be associated with an expectations based system:

1. It is too inefficient, because expectations are attached to individual lexical items, rather than expressing generally occuring situations. This makes it difficult to make generalisations, in contrast to to the efficiency of representation of the ATN model due to the ability to merge common parts (of more than one context-free rules).

2. It fails too easily when an unpredicted situation is encountered; what is worse, no natural and efficient mechanism for dealing with the unexpected can be incorporated.

The alternative, strongly suggested by the discussion so far, is a system based on the semantic pattern matching of constituents, recognised by a syntactic component. This will overcome the difficulties which a purely syntactic pattern matching based analysis system faces (see [Riesbeck74] for criticisms on these) and equally well these of a wholly semantic approach. In addition the introduction of some conventional low-level syntax will effect a number of gains: among other things, the painless identification of constituents, i.e. slot fillers in semantic structure which may or may not be Wilksian templates; greater efficiency in identifying the main verb element of a clause; less proliferation of alternative semantic structures and no or few blind preference tests, because decisions can be delayed; a natural way of putting structural constraints on the invocation of the semantic judgement routines; and a framework in which it is easier and more natural to incorporate a default option for fallback in case no primary match is found. All of these are clearly very desirable.

An analyser incorporating these features is described in the next sections.

3.4. Nondeterminism (and Backtracking).

Before any attempt is made to write a computational grammar for English and to decide on a particular control structure and parsing strategy, an important issue must be considered, that of non-determinism in a natural language program. Basically, the phenomenon is a consequence of situations in which possibly more than one decisions can be made at a certain stage of the computation. In the framework of the ATN model, it is manifested in the availability of more than one arc (analysis path) that can be followed from a given state.

It can be safely accepted that in general the issues of non-determinism and structural ambiguity are connected. If we were dealing with the problem of lexical ambiguity only, this would not necessarily imply a non-deterministic control structure. A deterministic parser does not, however, allow for structural multiple choice (see 1.2) - instead some structures will be wrongly perceived: for example, the prepositional phrase will be attached to the same constituent (verb or noun, depending on the particular ATN interpreter) in both

(*) Max returned the book with speed, and

(**) Max returned the book with the pictures,

while some (other structures) will represent only one of at least two different, but nevertheless valid, structural interpretations: consider the structural ambiguity of

Kissing aunts can be boring.

On the other hand, a genuinely structurally ambiguous text will be parsed into multiple representations if and only if it is handled by a parser which allows some sort of backup (or equivalently, parallelism), i.e. is non-deterministic.

The actual strategy adopted by the parser in following up branching options (depth-first vs. breadth-first) is irrelevant. The fact remains that in a non-deterministic parser wrong decisions are sometimes taken and blind alleys followed until a dead-end is reached. The unpleasant effect of non-determinism is that a mistaken course may be pursued for an arbitrarily long time - which is clearly inefficient - and when it becomes clear that a wrong decision has been made somewhere along the line, it is impossible, in general, to deduce from the mere fact that it is wrong, how and where the analyser has gone wrong. So, whatever strategies for backup have been offered, they rely on an unintelligent, exhaustive search among the decision points. The inefficiency may be further increased by the phenomenon of a constituent being recognised over and over again in the course of the backtracking - on different analysis paths.

It is hardly surprising that it has been argued (e.g. by Riesbeck [Riesbeck74]) that a natural language analyser can be deterministic*, specifically by not being purely syntactic. After all, an analyser which "grasps" the intended meaning - lexically and structurally - straightaway; which is foolproof and reliable, because by design it is not supposed to make mistakes, is close to everybody's dream. The question is: are these suggestions justified, and by what?

Riesbeck takes a rather extreme approach: "there is no backing up to decision points... a decision point mechanism (implicitly claims) that when people make decisions, they expect them to go wrong... problems that cause backup are not expected - they are surprises" [Riesbeck74]. Wilks, however, has demonstrated [Wilks76b] that Riesbeck's parser (still) fails on sentences which are not necessarily "tricky or bizarre" and suggests that "simple unfettered expectation is not enough unless one can be sure one has got one's criteria right, or one has some breadth-first way of considering alternatives, or one has complex backup".

Situations like the ones which make Riesbeck's parser stumble are better handled by Marcus' system [Marcus75], who also takes the rather strong approach that backup should be rejected as the standard mechanism for parsing ("no backtracking can take place unless the sentence is consciously perceived as a 'garden path'"). In an attempt to avoid blind, non-productive searching, Marcus has designed his parser so that it will "only build a grammatical structure it is sure it can use".

Before making any comments on such an approach, let us examine the situations which have to be accounted for in any attempt to design and build a deterministic parser. These are closely linked (not surprisingly) with the different types of structural ambiguity as discussed in 1.2.

(A.) The possibilty of recognising the same constituent at different levels of analysis. This ties up with type (A.) of structural ambiguity, and is implied by the fact that "the parser should be capable of recognising a prepositional phrase as (possibly) modifying any of the preceding heads" (contrast to (*) and (**) above).

(B.) When looking for a constituent of a certain type (noun group, or verb group, for example), different portions of the input string can match against the recognition rules because of different distributional classifications of the elements of a structure: this is function of type (B.) structural ambiguity*.

(C.) The same surface string can be recognised, and 'consumed', by different recognition rules - thus being interpreted as representing more than one constituent. This accounts for the phenomenon of transformational ambiguity.

---

* this is not to be confused with the deliberate strategy of designing a parser capable of backtracking, but which takes the first successful parse (analysis path through the text) to be the correct one.
* sometimes this is referred to as "end of constituent problem"

Woods, in [Woods70] argues that any ATN can be converted to an equivalent "deterministic" one, with the exception of the pushdown operations. This does not really help much, because these are the causes for the situations enumerated above: types (A.) and (B.) result from the existence of more than one positions in the text string from which a POP may occur; (C.) happens when predictions clash: more than one PUSH initiated (or more than one CAT arc followed).

In the early stages of this project, an attempt was made to develop, or at least see how far one could proceed with, a purely depth-first parser. This was based on combination of certain features of the ATN formalism, which allowed the use of registers to hold (partial) substructures, delayed structure building decisions, and arcs (leaving a state) ordered in such a way that the first one to be followed was the correct one under the circumstances - i.e. the parser always made the right choices. The experience was very valuable, because it gave rise to a method for reducing non-determinism in the traditional ATN model - which is explored in resolving the ambiguity of prepositional phrases, as described later. It also prompted certain conclusions regarding Marcus' approach.

Central to Marcus' system are the complementary processes of recognising a situation (the situation in which the parser is), and, if it sets up more than one working hypothesis, applying "differential diagnosis" to decide between them. A conceptually similar idea underlay my first depth-first parser. This was directed at dealing, deterministically, with situations of type (A.), and consisted, basically, of recognising postmodifying constituents in isolation, only once - in such a way several syntactically valid structures were implicitly covered by a single analysis path, at the end of which the correct and semantically valid one is dynamically constructed, thereby avoiding much of the backtracking (or equivalently, parallelism). This, however, does not completely cover the case when Marcus' parser only builds a structure it is sure it can use. Consider, for example,

John admitted to the girl that he loves the truth,

with the text pointer after "to", and the current situation actively seeking a noun phrase. What is it to be: "the girl", or "the girl that he loves"? Clearly, this is type (B.) situation, in which POP-ing or the equivalent constituent structure building process can occur equally well in two different places. Similar problems arise on closer examination of Marcus' expectations - sometimes there is not enough data available (imagine the beginning of a sentence), and sometimes no differential diagnosis will help to decide on which hypothesis to pursue, as with

Kissing aunts can be boring.

Admittedly, these are genuinely ambiguous examples, but they are perfectly natural ones, and demonstrate clearly that it seems unlikely that a purely deterministic parser will be able to account satisfactorily for situations of type (B.) and (C.).

The conclusion to be drawn from all this is that in dealing with

realistic texts and a (syntactic) grammar of average size, the desire to account for a wide range of linguistic phenomena does not leave much choice for the parsing strategy - nondeterminism is unavoidable. In which case there are certain other matters to be attended to:

a. the actual low level strategy - is parsing implemented as backtracking or parallelism; depth-first or breadth-first,

b. (possible) ways of reducing the non-determinism - representing more than one syntactically valid structure by a single analysis path; blocking potential dead ends as early in the analysis process as possible,

c. making necessary backtracking more efficient - by a well formed substring table (WFST).

An interest in psychological modelling may be a justification for rejecting backup; but without other knowledge that we do not currently know how to supply, no backtracking is not a practical proposition.

3.5. The Problem Of Structural Ambiguity.

It will be clear by now that what is being proposed is a NL analysis system starting with a syntactically driven, non-deterministic ATN parsing. In such a context, the problem of structural ambiguity becomes a major one. It is also clear that even though the problem is associated with syntax, syntax alone cannot do enough about it.

Ambiguity has to be dealt with by introducing the principle of semantic judgement, i.e. semantic judgement is applied to the structures delivered, or to be delivered (see below) by the syntactic processor; before any structural representation of the textual unit is assembled, and labelled for further reference, the parser must

a. check its consistency as a whole,
b. check its compatibility with respect to the overall environment and global context,

both essentially semantic operations.

In a way this is the demarkation line between syntactic and semantic processing, and the way the line is drawn gives an overall characterisation of the system: the system is based on a syntactically driven ATN processor incorporating strong semantic judgement. There is of course the question of organising the interface between the two different classes of processes - recognition and structure building - which will be discussed in chapter 4.

This section is primarily concerned with the issues arising from the principle stated - even though it may seem so simple conceptually. Before the obvious (and comparatively lower level) question of how to perform these important checks can be answered, there is the fundamental concept of 'structure' as referred to above to be considered - together with all its implications. Basically, it all comes down to: how to define 'structure' for the purposes of checking; or, alternatively, - at what level to perform the checking.

The options for the unit which will delimit the scope of the semantic routines are: word, group, clause, sentence. Clearly, semantic checking at word level is of no use at all - it creates no structural, but plenty of lexical problems, and there is no context to help. The other extreme, semantic checking of the structural representation(s) of the whole sentence, is exactly the approach I wished to avoid: for one thing, there is too much disjoint information floating about, i.e. information will be supplied for one part of the sentence which is irrelevant when another part is being validated. Further, checking the whole sentence implies the possibility, (if not the necessity), of the traditional approach adopted by Woods for example - complete syntactic recognition is followed by semantic analysis which is unsatisfactory as was shown in chapter 2. This is not to say that no checking at all is to be done at sentence level;

after all, the construction of a semantic representation for the whole sentence is the ultimate goal of the analysis process and this representation has to be checked like that for any other constituent. The question is how to do this in a sufficiently compact, efficient, and intelligent manner, having got rid of some of the effort with lower level constituents.

Clearly, the bulk of the semantic work has to be done by routines operating at group and clause level. It is actually difficult to make a clear cut between checking for consistency, and checking for compatibility (see a. and b. above), because the semantic routines operate hierarchically, - so at level N+1 a (tentative) structure will be analysed for consistency - does it make sense; i.e. are the constituents from which it must be assembled contextually compatible. A positive outcome will result in the structure being assembled, labelled appropriately, and handed one level up, where at level N it will be analysed within the context of the encompassing structural unit, i.e. for compatibility. The hierarchical organisation of the semantic routines is a result of their operation both at group and clause level - more or less following the hierarchical recognition procedure carried out by the pushdown mechanism of the syntactic ATN preprocessor: while processing a noun group, an embedded clause might have to be analysed; in order to build up a structural representation of a clause, its constituents - noun groups, prepositional groups, embedded clauses - must be already processed. The semantic checks operate basically at clause and noun group level, whenever these units are found. Prepositional groups are never analysed on their own since a prepositional phrase means very little on its own (see 1.2). There is a certain conceptual likeness between the semantic specialists for the various constituents: the verb is central to the clause (see 3.8) and so the clause specialist is organised around the verb. Similarly, the noun group specialist starts with the head noun.

The questions asked at this point of analysis thus are: what structure can be assembled out of the constituents available at the current level, and will it be a semantically valid one. Note that this implies that the internal computational representation of a structure before the semantic routines are applied is not an explicit syntactic tree. Even though the very low level (front end) of the parser is a syntactic recognition box, recognition is just all it does. The register mechanism of the ATN model means that the step of syntactic tree building can be completely omitted as it turns out to be unnecessary. This also implies that a 'structure', as a generalised entity to which a semantic routine can be applied, is equivalent to a syntactically well formed constituent, whose recognition has been carried out in a more or less traditional way by the ATN grammar.

It must be stressed that the approach described does not necessarily mean the explicit construction of a representation for all syntactically valid constituents. As far as the when and how of applying the semantic routines is concerned, some general points have to be made. In the environment of a non-deterministic parser there is a better approach than simply building a syntactic structure blindly

and then checking it for semantic consistency. It is more rewarding to attempt a modified bottom-up strategy, that is an extension, along semantic lines, of Markus' "wait-and-see" principle as follows: recognise, pick up, and store component constituents; only when enough (at least the bare minimum) are available, try to assemble them directly into a structure which makes sense, and which does not have to be an intermediate syntactic one. This clearly demands more of the semantic routines - they are now actively, dynamically constructing a semantic representation of the textual unit, rather than just passively confirming (or rejecting) a hypothesised syntactic configuration.

Systematic application of this principle will make sure that all structures, at all levels of the analysis, are well formed; it will also rule out (potentially) ill-formed ones, before they are built up. Further, the strategy described means that it is not necessary to follow all syntactically valid paths through the network - these will be blocked half way through, thus achieving increased efficiency and reduced backtracking.

The strategy also provides a background for developing, as an extension, a special mechanism for dealing with the most often encountered type of structural ambiguity - the problem of determining the scope or level of a prepositional phrase modifier and its case function. This is based on the idea of 'intelligent' extension of a basic semantic structure by adding further pieces of information to it only in such a way that it remains a valid and interpretable structure. Again the overall effect is that the degree of non-determinism of the parser as a whole is reduced. The specific properties of the grammar required, as well as the operations involved, will be discussed in later sections.

The general strategy as outlined in the preceding paragraphs thus provides the basic mechanism for dealing with structural ambiguity: potentially nonsensical structures are not constructed at all so the corresponding analysis paths are aborted. This means, for example, that in the analysis of

(*) Kissing aunts can be boring, vs.

(**) Singing songs can be pleasant

the underlined phrases will be recognised in a similar manner by two different subnetworks. But whereas in the case of (*) the semantic routines will construct two representations corresponding to

*.1 (the act of) someone kissing aunts,

*.2 aunts (who) kiss someone

in the case of (**), an interpretation similar to *.2 - "songs (which) sing something/ someone" - is rejected even before being constructed, thus completely aborting the alternative analysis path.

It is important to realise, though, that the building routines which are the subject of this section are not solely concerned with the problem of structural disambiguation. I have not made explicit

so far what exactly is involved in a "semantic check for consistency", for example the exact mechanism which would hypothesise, and confirm (**.1) above, while rejecting (**.2). But it is clear that such mechanism will require knowledge about the components of a structure in semantic terms. Specifically, at a certain point in the hierarchy of checks it becomes necessary to know the semantic formulae of the individual surface words. Then, given dictionary entries with multiple word-sense definitions it is clear that the problem of structural ambiguity is not an independent one (as was noted in section 1.1). The two processes of structural and word-sense disambiguation cannot be viewed and carried out separately. The same routines that perform, extensionally, the structural analysis of a constituent will, intensionally, perform word-sense disambiguation. A semantic consistency check implies the resolution of lexical ambiguities; alternatively, unambiguous semantic representations for words or substructures are used to construct a well formed higher level structures. There is no separation of processes for the two purposes; and in the program individual functions like NPBUILD (build a noun phrase), or SBUILD, both test and construct.

Word-sense disambiguation is considered in more detail in the next section. It should however be noted here that due to the dual function of the semantic routines it is not feasible to introduce separate specialists, one for each (possibly) structurally ambiguous construct: thus there is no separate "-ing" phrase specialist, or "-to" complement specialist. General semantic routines are defined and introduced, which operate at clause and (noun) group levels. Low level syntax is used to tailor the various syntactic constructs to a standard form, after which the appropriate specialist can be applied: "-ing" phrases, "-to" complements, relative clauses, and the like, are handled by the clause group specialist; possessor structures, deleted nominals (to be passed down to relatives), etc., by the noun group specialist. Thus considerable effort is saved, in the same time designing a general and powerful specialist and keeping the system compact and tidy.

3.6. Lexical i.e. Word-sense Ambiguity.

This section presents an approach to the treatment of word-sense ambiguity*, which is later used to provide a framework for designing algorithms and parts of the analyser intended to deal with lexical ambiguity.

Generally speaking, the process of disambiguation is not necessarily choosing from a set of well defined, pre-set meanings, which somehow represent (the) different interpretations of a word. It is not even clear if such a set can be specified for all the lexical items in a person's (system's) vocabulary. In considering word-sense ambiguity and specifically noun ambiguity in the context of its treatment by a language processing program Hayes [Hayes77] points out that "given two usages of 'head' there is no definitive answer to the question whether those usages constitute different senses of 'head'. Much less is there a definitive list of the senses of 'head'". In other words rather than having a number of distinct word senses, one is faced with a more or less continuous spectrum of interpretations. A word-sense can be regarded as a segment of this spectrum with loosely defined, and possibly overlapping boundaries.

A difference in meaning does not necessarily imply multiple dictionary definitions. Context and conceptual images play important role in the interpretation of the particular use of a word. This point is made, for example, by McCawley [McCawley68]:

Is Brazil as independent as the continuum hypothesis?

and Lyons [Lyons77]:

John likes brunettes

John likes marshmallows.

It is not very clear whether this emphasis on context helps, because with no fixed boundaries between meanings of a word commonly accepted as different, the task of a natural language analysis system becomes very hard indeed. Taken to the extreme, this view is bound to encourage attitudes like the one taken by Simmons [Simmons73]: "varied sense meanings of a verb can be accounted for as varied implications of a given event-class that the verb designates, under the differing circumstances signified by different choices of semantic classes of arguments... For example, the verb "run" is taken to designate one event class - that of rapid motion - in all of the following environments:

John ran to school,

---

* as was noted earlier (chapter 1), and will be discussed again in this section, what is considered here is word-sense ambiguity over the whole range of syntactic categories.

John ran a machine,

The machine ran."

I see no justification for such a view, because in most cases it is difficult to find a single "event-class" general enough to cover all the verb meanings: consider

John ran a school,

A flower bed ran along the school;

on the other hand, it creates a feeling of false security; thus Simmons' system does not engage in an explicit disambiguation process, but in no way demonstrates a capacity to relate context to word in a manner which reproduces the felt differences of text meaning.

The problem is further complicated by the fact that ambiguous lexical items can represent events/ actions, objects, characteristics, relational markers (i.e. verbs, nouns, adjectives, prepositions). Hayes argues that the task of finding the correct interpretation for an ambiguous word as an object i.e. for a noun is equivalent to finding the piece of pre-stored world knowledge which represents that interpretation. On the other hand, characteristics (adjectives) like "green" or "independent" can hardly refer to any specific pre-stored piece of data, even if nouns can; without the objects they qualify, they are typically impossible to interpret uniquely. Similarly, Hayes' argument is doubtfully applicable to ambiguous words representing events (verbs). A word sense here is more geared to contributing to the interpretation of the text as a whole, rather than of the word on its own - consider "give" in

John gave Mary a beating,

John gave Mary a beating stick,

John gave Mary a beating heart,

(admittedly, the last sentence is a metaphor). This is the view taken by Riesbeck in [Riesbeck74].

Two points emerge so far:

1. A specific view on the ambiguity of lexical items is necessary, depending on the overall system objective. A translation system, for example, might allow (depending on the target language, of course)

```
independent1 = independent2
like1 = like2:
    ((*hum subj) ((*ent obje) (please feel)))
```

On the other hand, a paraphrase system might require that these be treated as a separate word-senses:

```
like1:
   ((*hum subj)
      ((*ani obje)
```

```
        (((same *ani) for) (please feel))))

like2:
   ((*hum subj)
      ((*inan obje)
         ((((please feel) cause) goal) want)))
```

Once this fact is acknowledged, two assumptions can be made for the practical purposes of a translation/ paraphrase system:

1. lexically ambiguous item <=> multiple dictionary definitions.
2. lexical disambiguation <=> (the process of) choosing the correct dictionary definition.

In other words, the view on ambiguity adopted here is that each ambiguous lexical item corresponds to a list of pieces of syntactic and semantic data, each piece related to a different formulaic interpretation of this particular item. The justification for such an attitude comes from the necessity for separate surface treatment (in the target language) for the differently interpreted items, where these different interpretations come from the semantic representation of the analysed text containing the underlying different semantic formula(s) (see chapter 5).

2. A realistic NL system should not concern itself with only one type of lexical ambiguity: objects (Hayes), events (Riesbeck),... - a wide dictionary sample including fairly sized vocabulary with multiple definitions of verbs, nouns, adjectives, prepositions is a necessary prerequisite. To my knowledge, Wilks' system is the only one attempting a general solution to the problem, though not testing it very far.

However, Wilks' approach is not adequate in certain respects. Closer analysis of Wilks' dictionary sample (dated 1972, from the Stanford Artificial Intelligence Project) taken in conjunction with his approach* in general, reveals that for Wilks lexical disambiguation is concerned either with deciding whether a particular word was used in its verb or noun, or verb or adjective, or noun or adjective meaning ("father", "box", "block", "colour", "left"); or, alternatively, with choosing between different noun meanings ("crook", "bar", "lock"). The primary mechanism utilised in the former case is the template matching procedure; in the latter - preference (selectional) restrictions on the verb definitions; in extreme cases paraplate matching is exploited. There were very few occurences of polysemous verbs - in fact the only one in this particular sample was "grasp". Presumably the basic disambiguation tool in this case would be template matching, but there is no evidence that the analyser in general cannot deal properly with, say, more than one sense of "call" or "ask". This is not simply because multiple word definitions are absent from the dictionary - three different formulas for "ask" and six for "call" can easily be constructed. The important point is that

---

* as implemented and tested at Stanford, and subsequently discussed in his papers.

Wilks' template matching procedure does not possess enough resolution power for serious word-sense selection.

Consider, for example

(1) John asked Mary ¦ a question.

(2) John asked Mary ¦ to come with him.

(3) John asked Mary ¦ for the book.

(strokes denote the points where fragmentation occurs.) On the template level, for each of these sentences the system will have to choose from

(*) man ask man
(**) man want man

in order to select the correct meaning of "ask" for each case:

```
ask1 (inquire):
   ((man subj) ((*ani obje) ask))

(ask2 (want):
   ((man subj)
      ((act obje)
         ((((man (please feel)) cause) goal) ask)))

(ask3 (request):
   ((man subj)
      ((*ent) obje) ((*hum from) want)))
```

Obviously, templates (*) and (**) are not going to be very helpful in this case. There is the further complication that in the case of (3), (**), which is evidently the wanted template, does not reflect the correct "underlying message"; this is more likely to be man want thing.

It can be argued that "ask for" and "ask about" could, conceivably, be accounted for by the paraplate matching mechanism, but this introduces another unwelcome point - which is in the same time a major theoretical issue. This is discussed in detail in the later sections on frames and default PP analysis; here it is sufficient to note that attempting to resolve ambiguity via paraplates means introducing very specific paraplate(s), in this example geared either to match the word "ask", or the whole (or a substantial part) of the semantic formula - in which case the justification for these paraplates (and their generality) is at issue.

Thus at best the analyser will wait for the paraplate routines in the hope of making a more definite decision in case any extra clues, i.e. extra templates, are available. At worst it will not know what to do. In both cases there is the additional effort (and cost) of carrying parallel structures through to the next phase of analysis.

The reason why Wilks' templates fail, is because of two conflicting characteristics they are supposed to have: they have to be general enough to justify the "gist <=> basic message" idea and carry out some sort of parsing from the surface text directly into an

interlingual semantic representation; in the same time they must be quite specific in order to perform word-sense disambiguation. This is why they can select between "a father" and "to father" in

Small men sometimes father big sons.

but not between "ask1", "ask2", and "ask3" above.

In a network based system such as the one which is the subject of this work, which incorporates (some) low level syntactic recognition, we might expect to find something equivalent to the following production rules:

S ===> NP (adv) verb (NP)
NP ==> (det) (adj) noun.

Thus "father" above will automatically be recognised as a verb because this is the natural syntactic function expected from a lexical item at its place: the ATN model is quite good at syntactic prediction. The ATN framework thus provides a natural solution to the problem of word-sense ambiguity manifested in lexical items functioning in different syntactic categories (see 1.1). The more difficult situation, however - that of ambiguous words functioning in same syntactic roles - requires more powerful selection mechanism(s) based on an organisational approach and strategy different from mere template-paraplate matching.

The device responsible for the lexical disambiguation will be the semantic routines discussed in the preceding section. Because there are separate specialists, functioning at different levels of analysis, the resolution power and the amount of (useful) work done by them will be different. The clause being the main unit which conveys a (more or less) complete piece of information, it is richer in context and thus allows decision-taking with a higher degree of certainty. In an environment where "green" can mean three different things, and "crook" two, the phrase "the green crook" cannot be processed by the noun group specialist any further than cutting down the number of possible interpretations from six (2*3=6) to two. Clearly, some care must be taken in distributing the work to be done by the semantic specialists between them, and in organising the interaction of various disambiguation techniques available in a natural and efficient way. All these issues will be discussed in detail later in this and next chapters (4.3, 4.4).

An advantage of the semantic analysis strategy outlined in this and preceding sections is that since a specialised semantic function naturally has limited the scope for its action, it is possible to work in a naturally ambiguous environment without having to explicitly build alternative structures corresponding to all the combinations between the possible word meanings.

A final point to be made here concerns the relative weight of syntactic and semantic information in the processes of disambiguation, both lexical and structural. The last two sections made it clear that the driving force behind these are the semantic routines. It is important to realise, though, that sometimes syntax plays more than

just a secondary part in the process of sentence analysis. In most cases of sentences with similar surface syntactic patterns being interpreted in different ways, semantic judgement has been the deciding factor. Occasionally, however, this can (and has to) be done on syntactic grounds alone. Surface syntax constraints force a single meaning to be assigned to

Fred gave her book,

which is different from the one derived from

Fred gave her a book.

(Contrast this with

Fred gave her money.)

Consider also

Kissing aunts can be boring,

Kissing aunts is boring,

Kissing aunts are boring.

Similar observations are aplicable to the process of lexical disambiguation. Thus the determiner in "a/the bill" makes the "person" meaning of "BILL" unlikely. Words used in different syntactic positions sometimes mean different things:

Bobby feels great.

Bobby is a great chess player.

Words used in different syntactic environments can have different meanings:

John stopped to help Mary,

John stopped helping Mary.

Of course, I am not implying that lexical disambiguation can be performed on syntactic grounds alone; situations where syntax is the only clue to the meaning are rare (although they have to be accounted for). Still, in a wide range of cases, syntax can be useful, and save a lot of trouble. Consider

John called Mary,

John called Mary a fool.

The fact that a use of "call" involves two objects as opposed to one helps immediately; although, of course, it does not offer a hundred percent certain solution: thus compare

John called Mary a taxi.

Similarly, in the analysis of

John admitted Mary to the house, vs.

John admitted to killing Mary.

the specific syntactic construct in the second example gives a clue

right away: the intended meaning is obvious: "communicate, with possible feeling of guilt".

Examples like these above show that sometimes reliance on syntax is necessary and unavoidable, and in some other situations it may be quite useful. Furthermore, it should be easy to incorporate some sort of a mechanism making use of this fact because the syntactic recogniser is intended to work at a very low level within the parser as a whole.

Predominantly semantic systems (insofar as they have tackled disambiguation in practice) tend to ignore this fact, or at least not explore it systematically: some effort in the cases of ambiguous verbs is invested by Riesbeck [Riesbeck74]; on the other hand, Wilks' system completely ignores the value of surface syntax for disambiguation.

A mechanism which incorporates these observations is discussed in 3.9: "contextual verb frames".

3.7. Grammar Properties and Parsing Strategy.

The discussion in sections 3.2 and 3.3 suggest a traditional transition network grammar for English. Woods [Woods72] has developed a syntactic preprocessor demonstrating convincingly that the ATN mechanism is capable of capturing (syntactic) regularities of natural language and that it is possible to extend the existing set of grammar rules to make them hospitable to even wider range of language constructs. The grammar i.e. set of syntactic recognition rules viewed within the context of the particular parsing strategy used in this project is a distant relative (although it keeps its overall structure and layout) of the one developed by Woods.

It is only a distant relative because it takes into account the fact that the system as a whole is aimed at very ordinary rather than technical texts, so some of the special care directed at dealing with certain constructs would not be justified at this stage of the work; also in many cases it is not possible to make the simplifying assumptions that Woods makes (the use of (syntactic) features on verbs for example: see chapter 4).

Also, certain properties and features are incorporated to allow the implementation of the ideas put forward in the preceding sections. These are to do with those aspects of Woods' grammar and parsing strategy which are incompatible with my principles and objectives.

Assuming that a Woods-type ATN parser will operate in a mode of pursuing all alternative paths (see 3.4), one source of increased non-determinism and unnecessary effort is the fact that - due to semantic analysis being applied to the syntactic tree of the whole sentence - the only way to terminate an analysis path is by reaching a syntactic dead-end. This has the consequence that well-formed but nonsensical strings are sometimes recognised over and over again, and, further, allow the analysis to proceed beyond them. By applying semantic routines at well chosen points of the analysis process, potential dead-end paths can be blocked as soon as a semantically unacceptable structure is recognised. This will reduce the amount of non-deterministic processing that has to be done as a whole, and considerably minimise the element of blind syntactic search. The "well chosen" points in the analysis clearly have to do with a well-formed syntactic constituents, however, they have to be carefully selected so that they do not force premature decisions making, which might have to be reversed later (see "semantic wait-and-see": 3.2, 3.5). The semantic routines applied at these points rely heavily on the ATN registers mechanism since the constituents recognised so far are kept in registers. There are no structure building actions for building partial structures on the arcs of the grammar itself. The semantic structure building is done after a complete syntactic unit has been recognised. For example, given (NP V NP PP) as a surface constituent sequence, the grammar does not attempt to POP a partial

(subj (v obj)) (or agent-action-object, or whatever is the equivalent to the basic predicate, or underlying template structure) before the following PP is recognised. The basic structure building actions, as provided by the ATN formalism, are embedded in more complex programs, semantic specialists, which undertake conditional constituent assembly depending on the outcome of semantic tests applied to the semantic properties of the items held in registers.

As a parenthetical remark, it is worth mentioning that the desire to avoid premature decisions and blind guesses, taken to the extreme, will lead to a strategy similar to lattice parsing [Woods75]. This is not a very good strategy, however, for several reasons. It implies a predominantly bottom-up approach - involving in one way or another a number of scans over the lattice components, each of which results in collapsing a sequence of constituents into a larger structure. A major disadvantage of such an approach is that the boundaries of a clause unit are not directly recognised, which, among other things, prohibits the introduction of a clause level semantic specialist (see 3.5, 3.6), makes the hierarchical organisation of semantic routines application difficult (see 3.5), and implies an active rather than passive parsing approach* (see 3.3). As a side effect, semantically invalid partial (sub)structures recognised half way through the sentence do not have the desired effect of aborting the analysis path (see above). Another disadvantage of lattice parsing is that it is difficult to embody context sensitive rules in the recognition component.

On the other hand, the parsing process as implemented in this project, is (strongly) guided by complex context sensitive rules on the arcs of the grammar (see below, 4.2). These function as bottom-up devices which, within the traditional predominantly top-down ATN framework, help to further minimise the unnecesary and unwanted proliferation of alternative syntactic paths.

As already noted, the parsing process guided by an ATN grammar like Woods' original one suffers from the disadvantage that it is highly non-deterministic, which follows from one of the most important features of the ATN model, namely the recursion mechanism. This is not to suggest that recursion should, or indeed could, be eliminated. But closer examination of some situations which may arise during parsing shows that the grammar can be modified so that the non-determinism is substantially reduced. In particular, much more nearly deterministic parsing can be achieved for the most common type of structural ambiguity - type A. (see 1.1 and 3.4), concerned with determining the scope of postmodifying prepositional phrases and certain other constituents like optional (post)modifiers. Briefly, the situation occurs because there is no pre-set limit to the nesting of subordinating endocentric constructions [Lyons77]: "the book on the table in the bedroom on the second floor... in the house", and because a PP can potentially modify any 'head' preceding it. In terms of formal grammar specification this results in mutually recursive

---

* passive approach works better if the boundaries of a textual unit are known

networks, equivalent to the following production rules:

S ===> NP V (NP) (PP)
NP ==> DET N (PP)
PP ==> PREP NP.

It was noted in 1.2 that the surface string (V NP PP PP) parses in five explicitly different ways by the rules above. On the other hand, if the mutual recursion is eliminated, by removing the PP option in the second rule, only a single analysis path will be found. This nevertheless identifies a set of constructs which allows the semantics to construct those of the five possible structures which are valid for the text.

It does not however follow that nouns cannot be postmodified by prepositional phrases. Prepositional postmodifiers are recognised at top (clause) level only, and are pended in a register - thus syntax makes no guesses as to which is the head, and which is the modifier. Only after the 'end of clause' signal is broadcast by the scanner, is semantic judgement (the clause specialist, aided by a prepositional phrases analyser) called to determine the proper head-modifiers(s) relationship and dynamically construct the corresponding semantic structure (or structures, if more than one interpretations are available, as for "man in the park with the telescope").

Woods claims that semantic judgement called during syntactic analysis does not make the whole process more efficient [Woods73]. Possibly one reason for this is the fact that his grammar has to produce all the five syntactic readings. Whereas for

Bill returned the book with pictures by post.

my system will construct only one structure directly, with the alternative syntactically valid bracketings implicitly represented by a single analysis path.

The partial elimination of mutual recursion is thus the basis of the mechanism for reducing the non-determinism in the parsing process. The situation does not apply to postmodifying prepositional phrases only. For example consider:

I heard an earthquake singing in the shower [Wilks75b]

I heard Fred singing in the shower;

Bill admitted to Mary that he loves Janet,

Bill admitted the fact that he loves Janet.

and also (see 1.1):

John admitted to the policeman that he killed Mary by strangling her,

John informed the landlord that he wanted to leave by writing a formal letter.

Instead of specifying in the grammar that an "-ing" phrase, a "-that" complement, etc. can modify both verbs (at clause level) and

nouns (at noun group level), it is better to recognise such items constituents only once, and, when the time is ripe, analyse them in conjunction with the overall contextual environment to establish their proper function. Clearly this requires that the corresponding semantic specialist knows, not only what can modify what, but also how this modification works: i.e. what is the relationship between the head and the modifier.

Finally, within the general framework of this principle, it is possible, and useful, to explore certain natural constraints. For example, although both "the man on the top of the bus in the park..." and "the dog that worried the cat that killed the rat ... in the house that Jack built" involve embedding [Lyons77], the extensive and careful treatment necessary for the first type of structure is not needed for the second. (Although beware of

> ... the punishment of the workers who were responsible for the strike,
>
> ... the punishment of the workers which will serve them right

.....)

3.8. The Verb Is Central To The Clause.

The addiction to the clause level specialist designed to operate on a semantic "wait-and-see" delayed decision principle, displayed in the preceding sections, is not an accidental or random one. It is an extension of the fact noted in 3.3 that one of the major difficulties in writing a semantically driven ATN is the need to suspend all the "interesting" work of the network processor (that is the work going on on top of the low level string processing) until the verb element is identified and isolated.

It is hardly an illuminating discovery to state that the verb is the main and central component of the clause; and this section is not intended to prove it. The fact has been recognised both in linguistics and AI research. I shall only mention here, among others, Chomsky's subcategorisation rules [Chomsky65], Fillmore's case grammar framework [Fillmore68]:

Sentence ===> Proposition + Modality
Proposition ==> Verb + Case1 + ... + CaseN,

Wilks' inventory of bare templates - organised, stored, and indexed by their action elements, the semantic networks of Simmons [Simmons73], the bulk of Riesbeck's request packets, as well as the control structure of Goldman's generation program [Goldman75], Woods' semantic templates... the list can be continued.

My feeling, however, is nevertheless that not enough emphasis has been placed on this fact in building analysers; and this section tries to analyse fully its relevance to and implications for the analysis phase in processing. Basically, two points emerge. Firstly, in a natural language analysis system which accepts the polysemy of words, and verbs in particular, it is crucial to isolate the correct intended verb meaning as soon as possible. No further text interpretation work can be done without this. But then, on the other hand, the centrality of the verb can also be used to our advantage. For one thing, with the main verb identified, further semantic interpretation can, with a certain degree of confidence, be guided along some specific lines. At the same time, since a "verb meaning" is identified only by its context, this very context could be used for selecting and/or preferring the particular, specific verb sense. In a way, this is the old "push-pull" (feedback) principle, with certain emphasis (as will be shown later) on the role of the verb in it.

The importance of the verb is responsible for the verb-oriented content of the dictionary, which was, as it happens, one of the starting points of this project. It also further highlights the weakness of the template matching mechanism mentioned in 3.6. Thus template matching as a device for disambiguating verbs becomes less important. There is certain rationale behind this, which will be discussed in the next section. Note that it does not, however, imply

that templates idea can be rejected altogether, only that they, and implicitly "preference semantics" as a whole require a more concrete basis.

In order to provide this firm foundation for analysis, more detailed information concerning verbs, verb meanings, their use, and relevant environment and context is necessary; and it has to go in the dictionary in some form. And, of course, when the semantic routines at clause level are activated, they (will) have to rely on some sufficiently powerful mechanism to make full use of that information.

Both these issues are discussed in the next section.

3.9. Contextual Verb Frames.

The fact that the verb element is the key to the meaning of the clause underlies in a very important way the design of the analyser. This section attempts to put together most of the considerations discussed so far by outlining the mechanism intended to perform global pattern matching based (3.3) analysis of the constituents at clause level (3.5, 3.6),possibly extending beyond the underlying template (actor-action-object) boundary (3.6); to concentrate on selection of the meaning of the verb (3.8) by examining the clause context; and possibly to use syntactic clues to help in the process (3.6), if not as a deciding factor, then at least as natural constraints.

Let us consider again the examples (1)-(3) in (3.6):

John asked Mary (a question) (about the book),

John asked Mary to come with him,

John asked Mary for the book.

The analyser faces several problems related to sentence interpretation and the construction of a representation (dependency) structure, the most immediate of them being verb disambiguation.

Assuming that there is access to the semantic formulas corresponding to the surface words, let us suppose that these are examined not only at the level of underlying templates (see 3.6), but in conjunction with a set of rules which for the time being and clarity of argument can be considered to be exhaustive with respect to specifying the possible syntactic environments, i.e. sentential constructs in which the verb "ask" can appear, and which can be sketched as follows:

```
frame1:
   *hum ASK (*hum) (@sign) (ABOUT *ent)
frame2:
   *hum ASK *hum TO *do (@act)
frame3:
   *hum ASK (*hum) FOR *ent.
```

In these rules words in capital letters identify words as they appear in the surface text. '@' and '*' indicate respectively semantic elements and classes of these; brackets denote optionality.

Let us examine these formulae in more detail.

(1.) The suggested rules are presented in terms of semantic primitives, rather than surface words. This allows for generality and better expressive power; a wider range of contexts can be specified in a neat and compact manner.

(2.) The only words that come directly from the surface text

(apart from the entry verb itself) are certain keywords - like complementisers and prepositions. But whereas in Wilks' fragmentation routines these keywords only signal the breaking of text into fragments, here they participate in the rule frame on the same basis as the other elements; in a way they are even more central because they trigger a search for an entity of a particular type.

(3.) A frame is an underlying pattern, and not a direct transcript of the order in which the items should appear in the surface text. Both

Mary asked a question last night about the book, and

John asked Mary with impatience

will map onto the same frame, frame1.

It is argued [Riesbeck74] that certain features of pattern matching, namely lack of communication between the different patterns resulting in lost effort when a pattern fails, ordering demands, inflexibility because a pattern matcher is based upon static classification, rewriting, delimit its possible applications in NL analysis systems. This is true, if pattern matching is the only mechanism employed, i.e. the system is a "pure" pattern matcher. In the approach presented here the frame is applied not directly to the surface text, but to the content of certain registers, holding the semantic information for units obtained by the syntactic preprocessing, which has itself been done in a relatively efficient and effort saving way (see 3.4, 3.7). The frame itself is not a rewriting rule, but a context match aimed at lexical disambiguation.

(4.) The frame is a static rather than a dynamic device. It does not require any specific expectation-cued syntactic operation from the front-end recognition device. This makes it possible to keep the syntactic recognition part of the grammar small and manageable in size, while still being able to capture quite a few of the regular surface syntax constructions of the English language.

If the frame was implemented as a dynamic device, this would introduce certain problems (also see 3.3):

(*) quite a lot of syntactic information would have to be organised around a particular verb entry. Apart from making the entries heavy and cumbersome, this would reduce the descriptive generality of the recognition component. For example, instead of specifying acceptable patterns that can match the surface 'V NP' construct, one would have to associate a noun phrase request with every (transitive) verb, and then put constraints on its semantic content: "ask" requests NP(*hum); "admit" requests NP(*mar) (sign, mark-like entity), etc.

(*) at least some of the recogniser's control structure would have to be organised around the verb definition, which would introduce unwanted decentralisation in the master driver routine.

(*) the designer would have either to make unnatural predictions regarding optional constituents (see 1.2) which may or may not appear and do not contribute to context evaluation as far as the verb disambiguation is concerned; or to incorporate (with difficulty) a mechanism for recognising and dealing with these optional constituents (see below: 3.10). Frames, on the other hand, as will be

discussed in the next section, provide a more natural solution to the problem.

(5.) A frame can be regarded, in a way, as an extended version of a template: thus the Wilksian actor-action-object triple is a part of it. However one notices that while the pure template matching mechanism lacks resolution power (the underlying pattern in all the three examples above is the same: "*hum ASK *hum"), the frame can do all that a template does, and more: i.e. it usually provides a single pointer to a more or less complete contextual environment, which need only be mapped onto the representational system.

It may be argued that by repeated processing through template segmentation and paraplate matching (i.e. PICKUP and TIE), Wilks could end up with a similar structure, but this would require, apart from too much processing effort, very specific tests under the paraplate entries; so specific that they would not be justified. In any case such tests refer not to a general situation which could be classified and stored under "about" and "for", but to a specific one which should be stored under "ask".

Notwithstanding the similarity between templates and frames, they cannot be completely mapped onto one another. There is no list of frames, as there is an inventory of "bare" templates. The reason lies deep in the basic difference between the two concepts: bare templates are organised around action element primitives, while frames are organised around surface verbs and their meaning in context. When not enough context is supplied, then the frame reduces to a template. (It does not follow, however, that the frames are not general enough - see 1. above.)

(6.) The reason why frames are surface verb oriented lies in their intended function. They are a disambiguation device, which also provides a blueprint for building a structural representation for the text (see below). All the three frames above are bound together by the common element "ask". Hence

$$FRAME_i <=> ASK_i$$

thus achieving complete disambiguation of "ask".

In connection with this, there are some further points to be made.

6.1 All environments of a particular verb have to be specified for the frame mechanism to be effective.

6.2 There is always the problem of the simplest case: "John asked Mary" - which necessitates a default selection strategy of some sort.

6.3 It is not necessary to have the same number of frames and verb meanings. Consider "admit" for example. Assuming that

admit1 = confess
admit2 = allow to enter

it is possible to specify

@man ADMIT1 @sign TO @man
@man ADMIT2 *ani TO *place
@man ADMIT2 *ani TO *org
@man ADMIT1 (TO @man) THAT @act
@man ADMIT1 TO *do-ING @act

6.4 Although predominantly verb centered, a frame can

help with the lexical disambiguation of any contextually dependent constituent: assuming

club1 = weapon
club2 = place/ society,

John admitted Bill to the club

will match onto the third frame above, i.e. "club" will be resolved.

6.5 Without going into details about the actual organisation and implementation of a dictionary frame entry*, it is possible to specify important relationships between the frame components:

John hit Bill on the back

The truck hit Bill on the highway

will be interpreted in different ways (structurally) given a frame

*ent1 HIT *ent2 ON (physob (part-of *ent2))

In addition, the possible ambiguity of "back" will be resolved, much in the same way as in

John recognised Bill as the crook.

due to

@man RECOGNISE *ent AS (same *ent)

In other words, apart from specifying relationships between the verb and its arguments - much in the way a template, or a formula, specifies selectional or preference restrictions - a frame can specify a relation that holds between the arguments of a verb*.

(7.) If no complete disambiguation is achieved after applying the available frames, two courses of action are open. In the case where no further information is available within the current scope of semantic routines apply a default plausible interpretation selection mechanism. Alternatively, we can proceed to the next stage of analysis which will (attempt to) deal with any optional constituents lying around; this involves further disambiguation within the current (clause) level and addition(s) to the partially built structure (see below: 3.10, 4.4, and 4.5).

(8.) The frame has a dual function: it is primarily a mechanism for verb sense disambiguation, but it also serves as a background providing information for semantic consistency checks (see 3.5) to be made by the semantic routines; it further serves as a blueprint when the semantic structure reflecting the context of the selected frame is built. Thus the frame is the practical application of the assertion that the verb is central to the clause (see 3.8), and is the major device used to guide the analysis process.

(9.) A frame must specify the relation between the verb and each one of the constituents (frame slot-fillers) around it: in

---

* for example, see Appendix (ii).
* this last point has been also made by Hayes [Hayes77] who calls it a VDIR (verb directed association) but does not pursue the issue any further, as he is interested primarily in association based disambiguation of words as physical objects.

@man ASK @man ABOUT *ent,
@man ADMIT @man IN *place

the semantic roles of @man, *ent, *place must be made explicit as recipient, subj-matter, location... A label like "subj-matter" appears crude, but is acceptable as long as the analyser (and generator) interpret it correctly. For more detailed discussion of this issue see 3.11.

(10.) The notion of a verb frame can be extended to apply to nouns - mostly cases where the head noun is followed by some clausal construct (but not a relative clause), for example

...the fact that John loves Mary.
...the idea of running away.
...the necessity for a student to work.

and also to predicate adjectives; thus one sense of "afraid" would have the frame form

*ani BE AFRAID OF *ent,

and "angry that", "angry with", "easy to", "eager to", etc. would be similarly handled.

As a trailer to a more detailed discussion (in chapter 4) this, in conjunction with the basic idea of a verb frame, makes it easy and natural to select the points at which the frames mechanism should be invoked. It is, as pointed out in 8. above, the backbone of the semantic checking, and so the natural place to invoke the semantic routines is at the structure building actions at clause and noun group level. The frame thus becomes not only a lexical, but a structure disambiguation device as well (see 3.5; also below: 3.10).

With the proviso that the frames notion is extensible to nouns and adjectives, I shall continue to refer to them as "contextual verb frames".

(11.) Frames make it relatively easy to handle idioms. It is sometimes assumed that an idiom must have a separate, spelt-out dictionary entry; but the detailed organisation of an idiom handling mechanism on this basis is far from simple. With frames, idiom handling is just another form of pattern matching, with more tight constraints.

From the discussion so far, the general definition of a contextual verb frame emerges: a frame is a static pattern which defines a more complete and detailed syntactico-semantic (contextual) environment for the verb (sense). Its function is to trigger a preference test coupled with pattern matching of constituent slot-fillers on a more global scale than the one provided by the simple template. The implication is that the suggested frame mechanism is more powerful than Wilks' template matching, embodying all its strong features, namely semantic generalisation and succint meaning characterisation, but overcoming its lack of resolution power.

The immediate effect of frame application is to speed up and facilitate the verb choice procedure; the overall effect is to provide solid background for the semantic procedures; both make the parser more efficient.

3.10. Default Treatment Of Prepositional Phrases And Other (Optional) Postmodifiers.

The analysis of prepositional phrases and other postmodifying constituents is a major problem in any computer based NL system. It was pointed out in 1.2 that the two different modes of prepositional use, obligatory and optional, express not just a notational difference, but are associated with the genuinely different conceptual roles that a prepositional phrase can play as it contributes to the meaning of a sentence.

The problem of prepositional postmodification - its scope and function (see 1.2) - is open to two approaches: active (i.e. expectation based) and passive (i.e. pattern matching). Using, for example, the parsers developed by Riesbeck and Wilks respectively it is possible to discuss these in more details.

The major advantage of active prepositional handling is the possibility of developing an efficient and elegant parser, which might save considerable amount of time and effort. There is no need at all to build any intermediate syntactic structures, and an effect of "intelligent text perception" is simulated; this is the support for Riesbeck's claim that his system "could also serve as a theory of human comprehension of natural language" [Riesbeck74].

There are, however, certain drawbacks in this approach. Each verb entry has to be separately written, which makes the dictionary bulky; and there is no way to capture generalisations or similar patterns of behaviour (which Woods' merging of ATN arcs or Wilks' semantic template patterns both achieve). Moreover, there is still no hundred percent guarantee that correct analysis will be obtained, as Wilks points out in relation to "John gave Mary to the Imam of Oudh" and "John gave his city his stamp collection" [Wilks76b]. Furthermore, a system based on specific entries relies on the assumption that "verb entry" <=> "verb sense", and the problem of selecting the correct verb sense has then to be dealt with, a question which Riesbeck does not discuss sufficiently fully (see 2.2). If semantic expectations or requests attached to the verb act as an active verb disambiguation device by looking for constituents with a certain semantic content and, upon finding them, preferring a particular verb reading, sense selection is achieved but at a high cost because of the extensive semantic parallelism inherent to the parsing process.

However a satisfactory mechanism for dealing with optional modifiers within the active framework is difficult, although not impossible, to implement. Since optional postmodification is essentially incompatible with the expectation basis of the active approach, the control structure of the parser has to be decentralised, with control distributed between the expectational data structures associated with the verb entries, and some sort of preposition-indexed

stacks representing the different possible conceptual realisations of optionally used prepositional phrases. These stacks cannot be avoided since they are the only mechanism capable of dealing with postnominal prepositional phrases modifiers in the general case; thus, apart from the distributed control, an active parsing system starts to exhibit properties inherent to a passive one.

This alternative passive approach has a number of important advantages where prepositional phrases are concerned. It allows for a simple form of the surface syntax rules, and a compact recognition grammar. Both first order prepositional phrase analysis and default (optional) PP analysis can be naturally incorporated within a unified control structure and dealt with by a general mechanism, which can be easily extended to account for postnominal modification. In such an environment, the analyser stands a much better chance of correctly 'understanding' an unexpected and unpredictable postmodifier. It can be argued that such a system faces the problem of multiple choice at its very worst, since no attempt is made to "understand each word as soon as it is read, to decide what it means and how it relates to the rest of the text" [Riesbeck74]. Similarly, at first sight it is not very clear where and when forward scanning of text should be interrupted, and it looks as if intermediate syntactic structures have to be built and later examined to see if they make sense. It is however possible, as noted in 3.4, 3.5 and 3.6, to avoid excessive syntactic non-determinism and semantic parallelism by delaying decision-making until enough context is available. Clearly the passive parsing approach is ideal for providing the global context of a textual (semantic) unit - it thus avoids a frontal attack on the multiple choice problem and instead allows a more 'intelligent', and also more efficient, resolution of word-sense and structural ambiguities.

It is clear that any realistic NL analysis system should be capable of providing equally good treatment for both obligatorily and optionally used prepositional phrases, and postmodifiers in general. In passive parsing approach the various techniques required to account for the different situations can all be handled by the pattern matching (see below); furthermore, these techniques can easily be accommodated within the same organisational framework. In contrast, in the active parsing approach the expectation/ request mechanism is not powerful enough to act as an organising principle in system design. This is yet another reason why the analyser described here relies on a generalised pattern matching of constituents within a certain context. It is also the rationale behind Wilks' paraplate stacks and TIE routines, which "establish dependencies between the representations of different fragments" [Wilks73b] (see also [Wilks76]).

The pattern matching idea must, however, be approached rather carefully. Wilks [Wilks76] goes further than Schank, who suggests that there is a short general list of functions of prepositions [Schank73]; for Wilks the fact that a preposition relates to the interdependency between modifier and modified (see 1.2) is reflected in his paraplate stacks, which are organised under a dictionary entry

for a preposition, but are functions of primitive actions as well, so that they are further organised as stacked (partially ordered) sublists, arranged under primitive action elements.

This approach does not, however, deal with some important linguistic facts. Firstly, nouns can be postmodified as well as, and independently of, the verb. This means that new types of entries will have to be added to the paraplate stacks - sublists, indexed by nominal elements or nominal classes. Neither the function nor the organisation and format of the paraplates will change, but the stacks will probably become more complex. This is a minor point, which nevertheless has to be made.

More important is the fact that some situations of prepositional postmodification are more a function of the idiosyncratic behaviour of the surface verb, rather than of the underlying primitive action element - a point already made in connection with the distinction between the optional and obligatory use of prepositions. Paraplate stacks are not the right place for this sort of information. The role of the surface verb also suggests that paraplates are not a suitable general device for first order (see 1.2) prepositional treatment. It can be argued that the paraplate mechanism as such has sufficient power to parse constructions with obligatory prepositions like "ask for the book" and "look for Mary". The problem is that this would require very specific tests within a particular paraplate which are not compatible with the rather general information stored there. Furthermore, the tests would have to be so narrow, as either to quote the surface verb, "look" in this case, though this information about postmodification by "AT" prepositional phrases should surely be supplied under the verb entry rather than under "AT"; or somehow specify a particular "looking" act, which would unfortunately make it very difficult to avoid assigning the wrong interpretation to

I saw John at Mary yesterday.

for example.

Two points emerge from the discussion so far. Firstly, contextual verb frames are ideally suited as a device for dealing with obligatorily used prepositions, hardly surprisingly, since this was one of the reasons for their introduction. Secondly, paraplate stacks extended to accommodate information about possible noun modifications will suffice as a default mechanism for parsing optionally used prepositional phrases. I shall refer to these as preplates.

Preplates are conceptually similar (if not equivalent) to Wilks' paraplates - in that both are structures whose function is to provide information about possible constituent postmodification (typically by a prepositional phrase). They are again partially ordered stacks of sublists specifying possible relationships between certain types of constituent, and providing the structure building routines with functionally labelled skeleton structures (or just functional labels defining the relationship). Normal preference semantic principles still apply; thus Wilks' examples "put the key in the lock" and "threw the key in the lock" [Wilks73b] are correctly interpreted. The

difference between the preplates and Wilks' paraplates is in their content and order of application.

Due to the fact that all of the specific information related to the obligatory use of prepositions is made explicit in the verb contextual frames, the preplates themselves can be expressed in a more compact and general form, than Wilks' paraplates. As far as the order in which paraplate tests are applied is concerned, Wilks does not make clear the exact sequence in which the possible relationships between constituent templates are tested. The implicit assumption, suggested by the general principles of preference semantics, is that all possible combinations (five in the case of

John picked up the glass of milk from the table after lunch)

are made explicit (the order does not matter here), after which the semantically densest is preferred. As was noted in 3.4 such semantic parallelism is both unwanted and unnecessary. In contrast to this approach, my system takes into account certain natural constraints, for example that no crossover modification is allowed in English; a prepositional phrase is most likely to modify the constituent immediately preceding it, and then the one preceding that, and so on. A failed test can, in certain situations, preclude further attempts to establish a preplate match (see 1.2) and instead invoke an algorithm specially designed to cut down the proliferation of alternatives and avoid the explicit building of all the possible structures. The algorithm achieves the effect, without the effort, of paraplate matches being applied to all the possible constituent pairs. The algorithm itself will be described in the next chapter (4.5), but the idea has already been discussed in connection with the special properties of the grammar; thus the algorithm is the reason for prepositional phrases (and other postmodifiers) being held on a linear list, and possible relations in the semantic unit being dynamically hypothesised and tested.

Preplate tests could be applied either as soon as a prepositional phrase is recognised by the syntactic component, or after all prepositional phrases in the (current) semantic unit are available. Which of these two strategies has been chosen is discussed in the next chapter. One final point, however, is worth emphasising here, namely that both contextual frames and preplates are parsing devices based on semantic pattern matching; thus they can be conveniently handled by the same control structure, making the analyser more efficient, and the grammar more compact.

The general discussion of this section does not refer only to the prepositional phrases. Similar problems, although on a smaller scale, are associated with some other post-modifiers. This was mentioned in 1.2 in connection with

* John admitted that he killed Mary by strangling her,

** John informed the landlord that he wanted to leave by writing a formal letter.

Consider also

Fred made a gun to kill Jill,

Fred made an effort to kill Jill;

Max told Sue that he wanted to go home,

Max told Sue what she wanted to make her happy.

In general, the preplates mechanism can be extended to deal with such situations. However, it is the noun frames for nouns like "fact", "effort", "opportunity" which suggest an alternative more convenient treatment for certain types of clause postmodifier. Again, a general preplate exists, specifying a "*do BY *act" pattern, which reflects the fact that a "by ...ing" phrase introduces (usually) a @@manner definition. However, in most cases it is more useful to have a special mechanism, as a part of a more general semantic routine which examines more closely the hypothesised relationship between the postmodifier in question and its potential heads. For example, it is specific knowledge about "causing to die" by "preventing from living", and about the impossibility of "moving" by "writing", as well as general semantic constraints on stative verbs (such as "want", "love", etc. as opposed to the dynamic "kill", "inform": see [Quirk72]) which reject a "by ...ing" phrase as a postmodifier, that together enable the system to parse (*) and (**) above as "(kill (@@manner !strangle))" and "(inform (@@manner !write))". Similarly, the analyser knows that a "to-" or "that-" complement, when it is to be interpreted at clause level, is attached to the main verb and plays one of a restricted set of roles: @@goal, @@act, @@mobject (mental object), @@reason,...; and it is both surface syntax, and the analysis of the semantic content of the constituents involved, that defines the role of the complement.

It is important to realise that such specific mechanisms do not violate the general principles of the parser: constituent slot-fillers are recognised independently (with no attempt to build intermediate syntactic structure(s)), and what follows is again a generalised pattern match: for example the complement analyser embodies rules such as

```
@man *do *inan TO <clause> => @@goal
@man *communicate (TO @man) TO|THAT <clause> => @@mobject,
```

where the labels @@goal, @@mobject,... etc. specify the functional link between the verb and the embedded clause. Clearly these can be considered as very special preplates (or very general verb, or action, frames), and are embodied in a particular (sub)specialist only in order to make the design of the parser more natural, increasing its efficiency, and improving its performance. Conceptually, the application of these specialists is no different from the application of a set of preplates or frame rules.

3.11. Semantic Representation.

The results of the analysis process are recorded in the semantic structure which is constructed by the parser and provides a meaning representation of the input text. Since any subsequent manipulation program - the generator in this case - has access only to this intermediate form of data, the source surface text being lost by now, it is crucial to make sure that the structures delivered by the analyser

a. are unambiguous,
b. clearly identify and represent the components of meaning of a sentence,
c. make explicit the relation between these as well as the function they perform in the overall textual unit.

These criteria together define the level of understanding of the system. Whether they are met or not depends on the design of the semantic routines. It is clear that the representation of a sentence will reflect the hierarchical application and operation of semantic judgement rules: it is constructed gradually, step by step, in a bottom-up manner, so that each (semantically) complete constituent is represented by a (self contained) piece of semantic structure. A semantic component is thus assembled from its subcomponents, which obviously implies that the semantic formulas of the surface words always appear in the final representation. It is on the choice of these from the sets of word meanings associated with lexical items in the dictionary, as well as on the decisions related to the hierarchical structure assembly, and the labelling of partial substructures, that the whole analysis process is focussed.

On the other hand, it must be emphasised that a semantic structure as such is not a goal in itself, but an essential intermediate product which is going to be used for subsequent manipulation. It is this fact which defines the level of representation.

The overall objective of this work makes it clear that the final phase of the system's operation will require a generation component. The essential question connected with this is, what will be the source of the target language (English) words? There are apparently two courses open:

(1) if the dependency structure carries with it some memory of the surface words of the input text, this could be utilised for the generation; or alternatively,

(2) we start with no knowledge of the target words, but deduce them by examination of the dependency structure.

It seems to me that both extremes are unsatisfactory. A hundred percent use of the surface words carried through will, at the best, result in a generated sentence textually identical to the input, which will hardly prove any point (apart from the obvious fact that this

does not work for translation). Alternatively, if nothing at all is known in advance about the specific output words, the system's operation as a whole becomes time consuming and lacking in expressive power (see below). The approach adopted here aims at a compromise: the semantic representation of the input contains, together with the full formula for particular meaning of words, its "shorthand" notation as specified by the dictionary (see 3.9):

...(ask3 ((man subj) ((*ent obje) want))...

This means that the problem of output word (in this case, verb) selection is reduced: not only is the set of possible words at any point of the sentence (before the word selection component of the generator - see below - has been activated) not open ended any more, because the word "ask" can be mapped onto "question", "inquire", "beg", "request", "want"; it is further restricted because the word "ask" with sense "ask1" can only be mapped onto "question" and "inquire". The compromise is thus more biased towards the first strategy as outlined above, rather than the second; however, differentiation of word senses and subtle relations between synonyms mean that the eventual identification of specific synonymic output is not a trivial task (see 5.2).

It must be emphasised at this point that thus if the semantic formulas for the different senses of "ask" are different, then clearly we could just by analysing them decide on which "ask" has been intended on input, and respectively where to look for the correct synonym to output, without carrying forward the trace of "ask". However, it is always possible that more than one input word may have senses defined with the same semantic formula (indeed this is likely as the semantic primitive formulas are not very specific). Sometimes, this will not matter (e.g. with senses of "say" and "tell"); sometimes, however, there will be a difference between the words not captured by having distinct formulas (as with senses of "say" and "admit"). And it is going to be very difficult in both cases for the generator to decide on which one has been meant originally, and thus organise the generator process correctly. This is what "expressive power" refers to: obviously "John said that he loves Mary" is more neutral than "John admitted that he loves Mary", but will the generator be able to detect the subtle nuance in meaning, and reflect it in the generated sentence?

The index on entries like "tell1" or "admit2" is thus designed, when taken together with the semantic formula itself, to provide enough information to put the generator on the right track, defining the overall sentential context and also saving considerable effort and trouble. It may be arguable, in this case, whether the semantic formulas, or the representation of the sentence, or both, are adequate enough to represent meaning, if there still remains something outside. I do not think this is the case, however, because this "something outside" is the overall paragraph or story context, which although (undoubtedly) relevant to the task of NL analysis in general, is outside the scope of this project.

The overall translation/ paraphrase objective introduces yet

another point: it does not imply extensive manipulation of the internal representation. In a system like Schank's MARGIE [Schank75] it is essential that the inference component is handed a surface-language free representation. The generation component has no way of knowing how the CD structure being handled has been derived. It cannot rely on making any assumptions; so it has no choice left but to try to work out the output words on the basis of analysing its representation input for its defining characteristics. In contrast, the proposed system does not lose anything by allowing the shorthand sense notation to participate explicitly in the semantic representation; it actually gains something more than mere efficiency: fluency and preciseness; in a word "expressive power".

The representation scheme used for this project is loosely based on Wilks' semantic blocks, which are intended to act as an interlingual representation of the meaning of a piece of text. There are, however, some non-trivial differences between Wilks' and my forms of representation. Dealing with the need to choose between different possible ways of combining text meaning constituents requires a more rigorously defined and organised representation structure than the mere collection of templates that a semantic block is. The system should parse

The crook hit the girl with the bat with vengeance

into something like (simplified*):

```
(hit
   (@@ agent: crook!man)
   (@@ recipient: girl)
   (@@ instrument: bat!racket)
   (@@ manner: vengeance))
```

Wilks' semantic block for the sentence would consist much more simply of three barely linked templates - one main, and two dummies. Dummy templates stand for prepositional phrases ([dummy pbe ...], [dummy pdo ...]) or certain other constructs with deleted (implicit) components ("...to go home...", "...painting his daughter...", etc). Their introduction is necessitated because the template is the basic semantic pattern matched onto the text and the only parsing device in a system which (attempts to) shortcut explicit syntactic recognition. However, once a low level syntactic preprocessor is integrated in the implementation, it is not necessary to insist on keeping templates

---

* the notation is a shorthand for the actual semantic representation (see 0.1, 4.5 for detailed examples) as delivered by the analysis component of the system, and should be interpreted as follows: the "man" sense of "crook" (crook!man stands for

```
...(crook1 ((((notgood act) obje) do) (subj man))) ...)
```

is linked to the main clause verb ("hit" with its corresponding verb-sense) by the dependency tie 'agent'. Similarly, the dependency link between the "racket" sense of "bat" and "hit" is 'instrument'; "vengeance" depends on (modifies) the verb as the 'manner' of the hitting, etc.

explicit in the internal representation. On the other hand, since a template in its pure form is an ACTOR ACTION OBJECT triple, no matter what the representation is, it will always be present, at least implicitly, in it, thus making it possible to still carry out all the preference based matches and manipulations associated with the use of templates. This makes the explicit dummy template concept unnecessary, and, indeed, eliminates the template as the primary formal unit used in the process of building of semantic representation.

With the explicit fragmentation into templates gone, it is possible to go beyond the predominantly linear structure of semantic blocks. The general shape of a frame, together with the fact that the structure building process is organised procedurally around the semantic judgement rules, and notationally around the main element of the current semantic unit (event, state or entity representation), suggests a certain hierarchical structure organised around the main verb element, with the other constituents within the semantic unit (see above) clustered around, and attached to the verb by explicit dependency ties labelling the relations involved.

Each verb has an associated set of roles defined by the verb frame, which are filled by the constituents (hence "slot-fillers") around it. Although the possible roles in the verb environment are not restricted to only five (as with Fillmore), it is clear that the notational system adopted here is an extension of the idea behind a system based on deep semantic cases. I shall not argue here whether relations like @mobject, @similarity, @attribute etc. ("...say that...", "...feel like...", "...park with the telescope") are proper cases or not; they are at least plausible ones. The point is that each relation between a component and its direct governor is explicitly labelled by what I shall refer to as "a dependency tie"; and that such a tie exists between a governor and all its dependents within any subportion of the hierarchical structure. Neither am I arguing for the completeness of the set of cases used. Basically all of them are general enough and have sufficient amount of semantic content to justify their being included in the system inventory; more importantly, given the pragmatic approach being adopted here, they seem to work. Some appear to be more arbitrary than others: @neutral, @act, @essence, @subj-matter, but these are invariably part of the expected semantic content of a verb (noun) meaning, i.e. they participate actively in a contextual frame. Which is to say that it is not really the name that counts, but the relationship which will be present if this particular meaning has been intended; the generator will be expecting a certain constituent in this particular slot and will know what to do with it. It can be argued that no general inference rules can be posited about these more specific case relations, but then this could be expected since they are related to the surface verb, rather than to the underlying primitive action; their purpose generally is not to stimulate inference processes but to aid target language text generation.

All the relationships are made explicit, so no guesses have to be made by subsequent processes about deleted constituents, in contrast

to Wilks' simple fragmentation of text, which leaves the generator of French to puzzle out the implicit participants in a semantic unit - like inferring the missing subject in "...urged the women to leave", "...the girl that Bill asked to...", etc. [Herskovits73].

Regarding the verb as the main element of a clause (3.8) implies a certain importance attached to the clause unit itself. The clause boundaries delimit the scope of validity, and dependency of the functional ties. A clause marker is therefore needed, to serve as a demarkation line - this is equivalent, in a way, to Schank's STATE or EVENT headers. Also, it is important to make explicit the relationships between the main and any embedded clauses within the sentence. Hence labels like @mobject, @goal, @manner, etc. Not surprisingly, this requirement fits naturally within the frameworks both of contextual verb frames:

@man ADMIT (TO @man) THAT @mobject,

and dependency structure as discussed above:

```
(admit
   (@@ agent: ...)
   (@@ recipient: ...)
   (@@ mobject (kill John Mary))).
```

chapter 4.
How The Analyser Works.

4.1. Control Structure.

The overall control structure of the system described here is that of an ATN parser. A LISP program - the ATN interpreter - is responsible for maintaining the process of analysis during a strictly left-to-right sentential scan. The discussion in the last chapter has made it clear that the effort of system development was directed towards organising the recognition grammar, semantic routines, control (service) functions of the interpreter, and the interface between these in such a way that the semantic dependency structure for a sentence is built during a single pass through the sentence. This is achieved by invoking semantic processing at certain well chosen points (see below) at each level of the computation. The processing does not require explicitly built syntactic structures; in fact, no syntactic structures are built at all. The semantic routines, however, whenever active, construct portions of the final semantic structure, which are subsequently put together in a higher level semantic unit. All intermediate results of the recognition, or partial analyses, are kept in registers, rather than assembled in temporary structures, thus avoiding the need for blind structure building and potentially wrong decision-making. Thus the hierarchical format of the final dependency structure reflects the hierarchical manner of operation of the semantic routines; which in its turn reflects the process of syntactic recognition of the input text.

This sort of parser organisation and control structure (and also parsing strategy) developed as a consequence of the desire to avoid exhaustive searching techniques and reduce the chance of making wrong decisions now and having to unpick them later.

Given that a non-deterministic parser, as opposed to a deterministic one, is required (3.4), the question of search strategy needs consideration. Without going into details about depth-first vs. breadth-first techniques, it should be noted that non-determinism implies more than one search path through the network. Effort must be therefore directed towards terminating potential dead-ends as early as possible.

With this in mind let us consider the notion of a failure that terminates a search-path. The intuitive understanding of the term applies to situations in which an input word does not meet the conditions specified at a certain point of the analysis. In a purely syntactic parser these are basically syntactic category tests. However, since it is the transitions from state to state in an ATN that count in the end, it is apparent that these are controlled either by the success or failure in recognising

(*) words of specific categories (i.e. terminal symbols: is the

current word a noun?),

or

(**) syntactic constituents of specific meta-categories (i.e. non-terminal symbols: is the constituent a noun phrase?).

Failure at (*) is a category mismatch (or some other violation of low-level syntactic expectations). Failure at (**) is a meta-category mismatch, caused either by syntactic ill-formedness, which is reducible to the (*) situation (imagine for example an effort to recognise a "that-" complement in subject position in "That boy is very naughty"); or by semantic ill-formedness (trying to assess a possible skeleton clause structure "John admit Mary" during the analysis of

John admitted to the policeman that he killed Mary.)

This analysis allows us to extend the notion of recognition failure, which leads to earlier termination of dead-end search paths. Not only are syntactic constituents (made up of more than one lexical items) recognised, but their semantic coherence is assessed as well. Only in the case of a positive outcome of a series of semantic tests, is analysis permitted to continue - whereas in the traditional ATN grammar model (Woods' LUNAR) a recognition of a lower level syntactic unit (NP, COMPL, ...) would just pop a syntactic structure, a guess would be made as to how it could be integrated in the higher level syntactic phrase, and the analysis would continue.

It has already been mentioned that a transition is allowed by a condition specified at a certain point of the grammar. The ATN formalism makes it clear that "a condition" here is the conjunction of syntactic recognition and satisfying some test (specified on the arc). What this means is that the ATN mechanism allows for an efficient interaction between what is traditionally known as top-down and bottom-up parsing. This is not an entirely new idea (see [Ritchie77] on bottom-up devices). However, its implications have not been analysed in enough detail. Thus although it is another potential source of parsing efficiency, certain points have to be clarified first.

(1) Is it possible to have semantic tests in a (syntactic) recognition grammar? Where is their place? For example, Ritchie argues that no semantic tests (for semantic categories) can be specified on the arcs of the grammar [Ritchie78]. It is however suggested below that semantic tests can be placed on the actions.

(2) What sort of syntactic features are to be used during the analysis process? This question is connected with the previous one, and will also be discussed in due course.

I shall close this section with a brief summary of the basic instruction loop which embodies the principle underlying and representing the eventual operation of the parser: look for a complete syntactic constituent (of a type that could be expected at this particular position in the text string and this particular point of analysis, or one that is keyed by some specific syntactic clue:

preposition, complementiser, wh-word, untensed verb, etc.); then, as in its analysis and the building of an appropriate (semantic) representation semantic judgement becomes necessary, call the appropriate semantic routine; finally, construct the representation and hand it over for the next level up, where it will be stored in a register, ready for use when the structure building process for the new level is activated.

4.2. Syntactic Recognition (Grammar).

This project is not primarily concerned with developing a new, or different, grammar (of English), so I have not started from scratch in grammar building; the syntactic recognition rules used are of a conventional kind, and more particularly have been influenced by those of Woods' LUNAR grammar. Woods has convincingly demonstrated that sufficiently complex syntactic structures can be accommodated within the ATN formalism; my major interest has been in embedding semantic judgement within the network, and hence in the grammar modifications needed to allow this (though some modifications have been prompted by a desire for parsing efficiency). The issues of grammar modification, and possible ways of doing it have already been discussed (3.7). This section concentrates on some other issues.

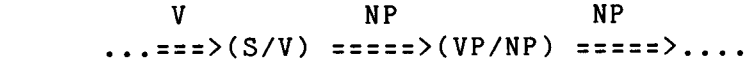
The first of these is the question: can we have semantic tests on the arcs of the syntactic network? Unfortunately, even though this is a tempting idea, the answer seems to be "no".

A low level syntactic feature can depend on, or identify related syntactic constructs in immediate context, but it cannot be predicted by a general semantic feature. This is no contradiction of my earlier claim (3.6) that low level syntax can help in the process of semantic resolution: in the case of

John called Bill a fool,

John called Bill a taxi,

both syntax and semantics play important parts in establishing that "call1" is different from "call2"; and further, in establishing exactly which of the six dictionary definitions for "call" has been used. If at the point of syntactic recognition of the verb "call" it was immediately clear which "call" was intended (say, "call2", as used in the second sentence above), the following piece of network could be devised. Using the notation (ATN formalism) as defined by Woods [Woods70], and presented in (2.1),



can be expressed and elaborated as

```
(VP/NP ....
   (PUSH NP/
      (AND
         (FEATURE CTRANS v)
         (ANIMATE (GETR obj1)))
```

```
(COND
   ((NULL (ANIMATE *))
      (SETR obj2 *))
   (T (ABORT)))........)...)
```

The symbol "*" refers to the most recently recognised constituent (resulting from the next lower level computation), and CTRANS is a feature to categorise verbs which have the following surface behaviour:

(Verb) (NP-ani) (NP-inan)

(I shall not attempt here to justify the introduction of this particular feature; syntactic features will be discussed in detail later).

This illustrative network demonstrates that although this is (one) compact way of putting constraints on the semantic content of constituents recognised by the grammar, it is not a practical one, because at recognition time there is no knowledge available to the analyser about the intended meaning of "call". All the analyser has at this point is a lexical item: i.e. a pointer to a multidefinitional dictionary entry. My whole idea is to work the other way round: only after NP1 and NP2 have been recognised and put into registers, can the knowledge that

a. "call" is used in a particular syntactic construct - followed by two noun phrases, and
b. there is a relation between the semantic properties of the two objects and the different intended meanings,

provide the basis for choosing the correct meaning of "call".

This does not, however, mean that semantic tests cannot be included in the actions on the arcs i.e. effectively decoupled from the transition: the tests decide which registers to pass down to an embedded computation (before the transition), or to which registers to assign a result of a computation (after it), and so on. For example, based on the diagram above, it is possible to write

```
(S/V ........
   (PUSH NP/ <test>
      (SETR obj *) ... (TO VP/NP)) ...)

(VP/NP ........
   (PUSH NP/ <test>
      (COND
         ((AND
             (ANIMATE (GETR obj))
             (NOT (ANIMATE *)))
            (ADD mods (BUILDQ (PP 'to obj)))
            (SETR obj *))
         ((AND (ANIMATE (GETR obj)) (ANIMATE *))
            (SETR obj2 *))
         (T (ABORT)))
       ........)...)
```

This piece of network will parse the similar syntactic patterns

Mary made John a cake,

Mary made John a cuckold.

realising that the former can be interpreted as "Mary made a cake to/for John", but deferring the interpretation of the latter until enough contextual information is collected, after which the clause level semantic specialist will have to select the proper meaning of "make":

```
((*pot subj)
   ((*ent obje)
      ((((same *ent) subj) (((own state) obje) change))
                                                cause)))
```

This selection is, however, a bit tricky. Thus imagine that "obj" contains an ambiguous lexical item:

John made Bill a crook.

The problem in parsing this sentence with the net given is that in effect the rules specified above define a single conditional action and thus a single path through the network. Nevertheless, a potential fork is created, but no backup will be initiated automatically by the interpreter control structure, unless this is explicitly prompted by some further test. The other alternative is to artificially create two states which are identical in substance, but split the test apart. The point being made here is not about how this particular case should be resolved. As the example shows, syntactic and semantic constraints have to be examined carefully before integrating ever more specific semantic tests in the grammar, and the possible dangers should be borne in mind; subject to these constraints good use can be made of network apparatus like this. For example, noticing that the net above is activated on an animate entity being held in "obj" register (sentences like

The Master gave the College his book collection

being recognised and analysed somewhere else); we notice that direct disambiguation can be triggered: a new function can be defined, which instead of passively testing (ANIMATE (GETR obj)), will actively pull out the contextually compatible (direct) object meaning: (GET-ANI-DEFN (GETR obj)):

Fagin made/gave the crook some dinner.

If no such direct and neat solution is possible, we can either multiply the meanings in different registers and do whatever can be done at recognition time, which is not much, letting the semantic routines worry about the rest at word-resolve, structure-building time; or, alternatively, we should not try to be too clever.

As all this shows, it is impossible to include semantic tests at word-recognition level, and rather difficult at syntactic constituent recognition level; and some care should be taken in the attempt to increase the bottom-up parsing power of the analyser by trying to elaborate the tests on the arcs beyond a reasonable limit.

Which takes us to the next type of arc tests: syntactic features. How many, and what sort of syntactic features should be introduced, and interrogated by the grammar?

In the context of a realistically minded syntactic ATN (the LUNAR grammar, for example), it has been found (as the final LUNAR report [Woods72] implies) that parsing efficiency can be somewhat improved attaching certain (predominantly syntactic) features on the lexical entries (predominantly for verbs). These are the domain of another class of possible tests that can exist on the arcs of the grammar. Possible examples are the traditional TRANS,INTRANS,INDOBJ,PASSIVE (marking the verb as one that can be passivized or not), or more exotic ones like TOCOMP, TRANSCOMP, SUBJLOW, etc. (see below). Also, there are groups of verbs marked COPULA, VPARTICLE, VTRANS (allowing a question as a direct object: "I know who won the race"). However, the concept of syntactic features has to be approached carefully. The whole idea works well as long as lexical items (verbs) are assumed to have single meaning. Then the correspondence between the item and an associated feature is one to one, and no problems arise. It is multiple dictionary definitions that make things difficult. The features now have to be attached not to the lexical item itself, but to its subentries. Syntactic features are associated with word senses. Consider for example the features PASSIVE attached to "weigh":

John weighed the letter.

The truck weighed two tons.

(examples from [Chomsky65]). Clearly, only the first of the examples above can be passivized. Similarly, TRANS cannot be associated with all the meanings of "call":

John called Mary,

John called on Mary.

Notice again that if we look at the phenomenon from a different angle, this very fact that a <u>verb meaning</u> exhibits certain surface syntactic behaviour can be used to disambiguate, as was already mentioned in connection with the definition of the contextual frames.

In principle there is nothing wrong with attaching features to the verb meaning itself, but this would require the subdefinition to be written in such a way that it could control the parsing - which brings us back to the idea of active parsing devices like Riesbeck's expectations; with all its objectionable consequences (chapter 3).

The situation with the more exotic features is slightly different

from that just discussed for straightforward ones, but nonetheless requires more attention than was originally given to it by Woods. The features are basically designed to allow the parser to operate "intelligently" when it has to deal with embedded clauses with deleted constituents (mostly "to-" complements after the direct object). Woods accepts that for most of these the subject of the complement is the object of the top-level (embedding) clause. The escape clause for exceptions ("promise" for example) is explicitly provided by a feature SUBJLOW which indicates that the subject of the main sentence has to be passed down. In addition, for TRANSCOMP verbs the object passed down to be subject remains as top level object ("persuade"), but this is not always true ("expect"). Thus a piece of network

```
(PUSH TO/COMPL ...
   (SENDR subj
      (COND
         ((FEATURE SUBJLOW v) (GETR subj))
         (T (GETR obj))))
   (COND
      ((NOT (FEATURE TRANSCOMP v))
         (SETR obj NIL)))
   ........)
```

which will parse, for example,

John wants Mary to go home,

John promised Mary to go home,

John expected Mary to go home,

John persuaded Mary to go home,

correctly: (John want (Mary go home)), (John promise Mary (John go home)), (John expect (Mary go home)), (John persuade Mary (Mary go home)). Unfortunately, an interpretation similar to the first one above will also be assigned to

John wants a car to go home,

which is clearly not what we are after.

The problem is not with different meanings of "want" (as it was above), but with the fact that the same verb (sense) is functioning in different contextual environments; and the task of working out the correct relation between the slot-fillers at different levels is more appropriately and easily done by semantic judgement routines than by introducing syntactic features.

On the other hand, there is nothing wrong with tagging a verb, "grow" for example, COPULA, although it may appear in other constructs (with other meanings) as well:

John grew,

John grew a beard,

John grew unhappy.

(Although, clearly, it should not be marked TRANS or INTRANS).

The examples above show that syntactic features to control the recognition (and parsing) can be very useful. They can be used as bottom-up devices; are potentially useful for the disambiguation process; and allow a concise definition of the surface syntax rules (patterns) without worrying about idiosyncratic behaviour of individual verb meanings (for constructs recognised by the grammar see Appendix (iii)). However, their use will be justified only as long as they

(*) either refer to the surface behaviour of a lexical item, rather than individual sense,

(**) or, when referring to different senses, do not specify mutually incompatible features (TRANS/INTRANS, INTRANS/INDOBJ, PASSIVE, ...).

In the latter case, the surface syntactic behaviour of the verb should be recorded somehow and used later by the disambiguation process. Which brings us to the next point about grammar design: flags.

Under closer analysis, and bearing in mind what has been said so far in this section, it becomes clear that the tests (conditions) on the arcs are effectively intended either to interrogate syntactic features of lexical items, or the context of the current textual unit. This last activity involves two different processes: checking the availability and/or setting of certain named registers, or determining certain semantic characteristics associated with the registers (their semantic content); then there is the analysis of the surface syntactic pattern within the boundaries of a well formed syntactic constituent.

In order to be able to do this, it is necessary to distinguish between two types of registers; or, alternatively - two modes of register use: for structure holding (with pointers either to dictionary entries, or to partial substructures) and flagging. The use of registers as flags is important because it allows the same recognition (and analysis) network to be used for two (or more) similar surface constructs; in addition it allows a surface syntactic pattern to be recorded in a compact manner. While the first of modes of use, structure holding, is a common one (see e.g. [Woods70]), the second, flagging, has not been regarded as important. But it is a key feature of the design of the present system, because it embodies the low-level interface between syntax and semantics. As has been emphasised, syntax and syntactic constraints are important for subsequent semantic processing: the mechanism of flags and the syntactic information (clues) they represent makes it possible to exploit the syntactic constraints.

This brings us to the semantic routines.

4.3. Semantic Routines And Semantic Processing.

The discussion in the previous section on tests, and on including semantic tests and features within the framework of a syntactically driven parser more or less suggests the natural location for the semantic routines. They are incorporated on the POP arcs of the grammar.

As discussed in the opening sections of the thesis (judgemental vs. structural aspects of semantics) the function of the semantic routines is to disambiguate (surface) lexical items and to construct a semantic representation of the syntactic constituent at the current level of computation (or, alternatively, to pronounce it ill-formed semantically and thus block a syntactic analysis path). Thus the fact that the full semantic operation of the parser is delayed until a complete syntactic unit has been recognised is simply a semantic version of the "wait-and-see" principle [Marcus75]. There is no semantic backtracking, though there is semantic parallelism since, given word polysemy, and possible structural ambiguity, we have to acknowledge the need to choose between, and possibly carry over (to the next level of computation) more than one semantic structure.

Thus the semantic routines operate within the boundaries of a syntactically well-formed constituent. The data on which they operate (apart from information provided by the dictionary*, i.e. semantic formulae, contextual (verb) frames, preplates lists, etc.), is basically of two distinct types.

The data may consist just of a set of syntactic markers (delimiters): named registers essentially containing entry points to the dictionary which provides the corresponding semantic information. These markers project onto a set of surface words which could, if semantically confirmed, be regarded and represented as a single semantic unit, to be treated as one single (semantic) entity in subsequent, upper level, semantic routines. For example "every single one of the students who passed their exams in their first year ..." will be recognised by the NP network a a valid noun phrase, with the head noun "student" , and hence, as a valid slot filler with head element @man. (Note that "student" is a surface text word, whereas @man is a semantic primitive - in other words there are different levels of interpretation of an entity).

The other type of data for the semantic routines is the semantic entity just mentioned - the semantic representation of a syntactic constituent. This is already separated from the surface text; English words appear only in conjunction with the semantic definition for their recognised meaning (3.11), syntactic labels such as "adj",

* see Appendix (ii) for examples of dictionary entries.

"noun", ... are abandoned, and there is an overall structural constraint associated with the entity, namely that it must be handleable as a single semantic unit.

Actually the distinction between these two types of data reflects the different phases of analysis that a (linguistic) constituent goes through. Initially its components are picked up through the ATN grammar; in the process of syntactic recognition the constituent being represented in the program working space by a set of registers which hold parts of the input text together with corresponding syntactic labels. When there are enough of these constituents lying around, i.e. enough to make it worth attempting some semantic resolution, the attempt is made to assemble them into a semantic entity, calling the semantic specialist routines which utilise the information in the dictionary entries of the words picked up by the syntactic recognition component.

The directive which has initiated the low level computation (PUSH NP/, PUSH S/, PUSH REL/ ,...) determines the semantic specialist involved. Basically there are three major semantic specialists:

clause specialist - it is known to the program as a function, SBUILD, and is invoked on the POP-s from sentence, relative clause, complements ("to-", "that-", "for-to", ...), "-ing" phrases, and related constructs;

noun phrase specialist - this is the function NPBUILD and as the name suggests, it lives on the POP arc of the NP network;

prepositional phrase specialist - known to the program as MODIFY. It is known to, and summoned to help, both SBUILD and NPBUILD, and is responsible for the analysis and structural distribution of the postnominal/ postverbal prepositional phrases (and extensible to deal with other postmodifiers)

These specialists are both functions in the implementation which organise the semantic tests, and structure building actions on the pop arcs of the grammar.

The NPBUILD specialist is the lowest level one - mainly because a noun phrase sometimes has no defined meaning (i.e. has more than one meaning) outside context (for example, what does "the club" mean?), but also because the hierarchical structuring of the syntactic recognition rules makes the noun phrase the constituent which is recognised first - at the lowest level - and there is then very little extra information to work on. So the NPBUILD does some not particularly exciting things: testing for compatibility between determiners ("the john"), ordinals ("three johns"), adjectives ("green crook"), and other prenominal modifiers and the head noun; analysing the possible postnominal modifier structure - which can be a complement ("an effort to..."), a relative clause ("the girl whom John loved"), possible prepositional phrases (which cannot be postverbal modifiers as well); polling all the registers at NP level, and constructing the semantic structure corresponding to the nominal group. In case of semantic ambiguity - the compatibility constraints

(see above) are examined; if some selection can be effected, it is carried out; alternatively, parallel structures are carried up to the next level in the analysis.

The SBUILD specialist is obviously the master semantic routine. It is organised around the verb (3.8), and the basic data on which it operates, apart from the slot fillers already isolated by the recogniser, and kept in registers to which it has access, are

(*) semantic formulas of the lexical item (3.1)
(*) list of bare templates [Wilks73b]
(*) the contextual verb frames indexed by the verbs known to the system (3.9)
(*) the syntactic clues which might be relevant to the process of semantic analysis (3.5, 3.6, 4.2)
(*) certain specific semantic knowledge (3.10)

This is the level at which most of the work of both lexical and structural disambiguation takes place.

The prepositional phrase specialist MODIFY can be accessed from both the SBUILD and NPBUILD functions. Its goal is to put the information about the modifiers picked up by the recogniser together with what it knows about the possible way a constituent slot-filler having a certain semantic content (most often this is a verb or a nominal group) can be modified. This information is provided by lists of "prepositional meanings" - preplates - indexed by the preposition itself. It was pointed out earlier that these are conceptually similar to Wilks' paraplates, but because certain (obligatorily used) prepositions and prepositional groups which are more strongly linked to the verb receive special treatment (3.9, 3.10), the preplate lists are more compact, more general, and thus correspond more fully to the notion "meaning of a preposition" than Wilks' paraplate stacks. (On organisation of preplate stacks see 4.5).

No matter what the particular semantic specialist involved is and what specific operations it must perform, there is a common principle underlying the design of all the semantic routines: namely that they should be capable of interpreting the current environment correctly (contextual environment, that is) and responding to it in a proper manner (this will be referred to as the principle of using up all the information available at a given decision point). The required information (as much as may be necessary) is represented by the names of the registers, and in the flags set up during the process of syntactic recognition. The semantic functions are sensitive to the registers and flags, and must make sure, before they terminate that all the information available at this decision point has been used. This is the only way in which we can be certain that there is some point in applying the semantic "wait-and-see" principle (see above). (Note that the semantic routines must be programmed in such a way, that control is passed over to the ATN interpreter only after all registers - both structure holding, and flags - have contributed to the context evaluation and to the content and format of the (final)

semantic structure - see 4.5.)

All in all, the design of the semantic specialists just discussed has the advantages of

(*) isolating the relevant pieces of information related to the semantic content of the currently processed constituent,

(*) localising the semantic units participating in a preference test (this was one of the biggest pitfalls of the purely semantic network of 3.3),

(*) reducing the number of such tests to be performed,

(*) allowing global pattern matching of constituents against the contextual (verb) frames of the dictionary, thus utilising to the maximum the information collected by the individual constituent analyses.

These features of the specialists make it possible to concentrate on the main objective of this work: resolution of lexical and structural ambiguities.

4.4. Techniques And Principles For Disambiguation.

As already reiterated, the task of resolving an instance of lexical or structural ambiguity falls wholly on the semantic routines. Although the two types of ambiguity are interrelated and can not properly be treated independently (1.1, 3.5), I shall discuss them separately, starting with structural disambiguation.

This is effected by two strategies: active and passive, which can be summarised as follows.

Active: given a basic skeleton structure, and a list of additional items which must go along with it (constituents which are optional, according to the grammar rules, but which have been recognised by the front end scanner), attempt, dynamically, to identify those structures which are allowed by the skeleton context. This process is being guided by the semantic content of the optional constituents, and in essence represents an attempt to add them where they would make most sense.

Most often these optional constituents are prepositional phrase modifiers in postnominal or postverbal position. The situations in which this method is applied are those illustrated by such examples as

I saw the man in the park with the telescope,

John hit the girl with the red dress,

John hit the girl with disgust,

John admitted the girl in the house,

John saw the girl in the house.

Note that following the original specification for the project, the analyser works on isolated sentences, and must therefore account for instances of real multiple readings. Whenever such a situation occurs, the output of the analysis phase explicitly gives all the sensible readings. The actual choice obviously depends on the global context, and this remains in the domain of a higher level resolution apparatus, whose design and implementation is not the task of this project. So, for the examples quoted above, the analyser will produce (abbreviating the actual dependency structure - see 3.11):

```
(1)
    (see
       (@@ agent: myself)
       (@@ recipient: man)
       (@@ location: park)
       (@@ instrument: telescope))
```

```
(see
   (@@ agent: myself)
   (@@ recipient: (man (@@ location park)))
   (@@ instrument: telescope))

(see
   (@@ agent: myself)
   (@@ recipient:
      (man
         (@@ location: park)
         (@@ attribute: telescope))))

(see
   (@@ agent: myself)
   (@@ recipient: man)
   (@@ location: (park (@@ attribute: telescope))))

(see
   (@@ agent: myself)
   (@@ recipient:
      (man
         (@@ location:
            (park
               (@@ attribute: telescope))))))
```

```
(2)
    (hit
       (@@ agent: John)
       (@@ recipient:
          (girl
             (@@ attribute: (dress (@@ state: red))))))

(3)
    (hit
        (@@ agent: John)
        (@@ recipient: girl)
        (@@ manner: disgust))

(4)
    (admit
        (@@ agent: John)
        (@@ recipient: girl)
        (@@ location: house))

(5)
    (see
        (@@ agent: John)
        (@@ recipient: (girl (@@ location: house))))
```

The apparatus involved in these processes is distributed between the MODIFY specialist and the collection of routines applying the idea of the contextual verb frames. Thus (1) to (3), and (5) above have gone through MODIFY, whereas (4) was derived entirely through the

DISAMBIGUATE-VERBS procedure (see 4.5).

The passive method is based on the instruction: given a possible skeleton structure, confirm (or reject) its semantic coherence. It is clear that this approach requires a closer interface with the syntactic recognition front end of the analyser. It relies on the non-deterministic control structure which automatically considers possible ways in which the surface text can be segmented. These are not necessarily all the possible ways because semantic rejection of a structure in the early stage of recognition blocks out the complete analysis path (3.5). In the case of semantic confirmation, the routines still have to construct the meaning representation, but this is a passive process because no further additions to this basic structure are involved.

This is the mechanism which will reject the second reading of (1) as opposed to (2.2) below:

(1) John admitted to the policeman that he killed Mary,

(2) John admitted to the girl that he loves the truth.

The various readings are represented by

```
(1)
   (1.1)  (admit
             (@@ agent: John)
             (@@ recipient: policeman)
             (@@ mobject: (kill John Mary)))

(2)
   (2.1)  (admit
             (@@ agent: John)
             (@@ recipient: girl)
             (@@ mobject: (love John truth)))

   (2.2)  (admit
             (@@ agent: John)
             (@@ recipient: girl [in] (love John girl))
             (@@ mobject: truth))
```

In the case of (1) the SBUILD specialist has been offered for consideration, among others, the proposition "John admits Mary". Considering the information about the possible contextual environment(s) of "admit" supplied in the verb frames and the preference restrictions specified in the semantic formulas for the acceptable meanings of the verb, SBUILD rejects the reading, and as a result, aborts the whole alternative analysis path which in the case of (2) derives the second reading (2.2).

The same passive mechanism is responsible for deriving, and confirming, the two readings of

Kissing aunts can be boring,

as opposed to the single interpretation of

Singing songs can be boring.

Note, however, that there is no strong semantic judgement involved in the derivation of a single reading of

Kissing aunts is boring, or

Kissing aunts are boring.

Here termination occurs comparatively early in the recognition phase, due to syntactic inconsistencies.

In the discussion so far it has been implicitly assumed that the semantic data on which the structure building process is based - i.e. the semantic word-sense definitions - is readily available when needed. In other words, the process of lexical disambiguation has already taken place. This is not exactly so, because as already pointed out, a word takes a concrete meaning only in context; while on the other hand it is clear that the best way to assess and evaluate the contextual effects is by examining the final semantic representation which, in its turn, needs the meanings of the individual words (3.5), thus initiating a potential vicious circle. In the program, the two processes run in parallel, rather than being organised in a linear or hierarchical fashion. For example, during active structure building (see above), the semantic coherence of possible different structures is examined and evaluated, bearing in mind all the possible meanings of each lexical item. Only (semantically) well-formed partial structures, remaining consistent under further structural augmentation, are considered; finally the acceptance of semantically coherent structure(s) establishes the correct and intended meanings of the participating lexical items - since this is a way in which a reading can be accepted, then these must be the intended use(s) of the words. (For more details see the specification of MODIFY in 4.5.)

It is clear that this particular approach to lexical disambiguation is very closely, indeed inseparably, related to the problem of structural disambiguation. I shall refer to it as the principle of maximum semantic coherence. There are, however, certain other techniques for lexical disambiguation, which may be applied at an earlier stage.

This earlier stage is the process of building the skeleton nominal or sentential structure - the basic action of NPBUILD or SBUILD (see the next section). The disambiguation techniques are based on

(*.1) selectional restrictions,
(*.2) preference expectations, as specified in the semantic formulas (but see 3.1),
(*.3) list of bare templates [Wilks73b],
(*.4) lists of contextual verb frames,
(*.5) constraints on possessive structures.

These were all discussed or referred to earlier (with exception of (*.5) for which the reader is referred to 4.5). Some of the

techniques operate at sentence level, some of them at NP level. Thus for instance (*.1), (*.2), (*.3) and (*.4) above are employed by SBUILD, and (*.1), (*.2) and (*.5) by NPBUILD. All of them, working together, should be able to (eventually) deliver the correct readings of the ambiguous items. However, there is the important problem of the organisation of their procedural application.

Ideally, and if we were attempting to simulate in parsing the processes going on in the human mind, there should be a global parallel assessment of a word applying all the relevant criteria and using the context of as much as is available of the whole sentence. However, it is difficult to implement such a scheme; partly because the system discussed here is organised around the hierarchical application of semantic routines; partly because no computer based NL processing system can process globally the text presented to its input - this is the nature of the text processing by machine and we simply have got to live with it. On the other hand, the "one word at a time" approach to disambiguation is characterised by inefficiency, waste of processing time, and combinatorial explosion. This is another area in which a pattern matching-based analysis system (viz. Wilks) is superior to an expectations-based one (viz. Riesbeck). Since the system discussed here is based on Wilks', but extends it to a more global pattern matching, it will be clear what sort of general strategy the program uses. In a way, this is the ultimate justification for introducing the notion of a contextual verb frame: it provides a way of carrying out at least some of the lexical disambiguation processes in parallel. However, this is a compromise between the two extremes, and the prospect of combinatorial explosion is still present. As a partial solution to the problem my parser applies the very important principle of semantic screening (not to be confused with Woods' screening in [Woods75]) which imposes an ordering on the disambiguation techniques employed. In order to keep the number of possibilities to a minimum and to avoid the risk of combinatorial explosion, the most discriminating disambiguation technique should be applied first, then the next one down, and so on.

This not only requires that the verb frames are applied before the bare templates; but also explains why bare templates have weak resolution power (3.6, 3.8) and so have low status as a disambiguation device. Experience with the current system has shown that a bare template can rarely achieve a complete disambiguation in a highly ambiguous environment: it can confirm a meaning, rather than select it from competing candidates. This same screening principle also requires that the semantic coherence of possessive structures is checked before applying any selectional restrictions: consider "John's big green crook" (where both "green" and "crook" have alternative, and obvious, meanings.)

With all this in mind, it is easy to imagine the overall process of disambiguation as the application of a series of filters each of which attempts parallel interpretation of some features of the raw information supplied, and tries to do as much of the work as possible.

When there is nothing else the current sieve can filter out, the partially screened information is passed on to the next one (for more specific details see 4.5).

The implementation of these principles and techniques, as well as their operation within the framework of the complete system is the subject of the next section.

4.5. The Process Of Sentence Analysis.

The principles and mechanisms discussed in the last two chapters are embodied in a computer implementation - an analysis program, often referred to as 'the parser'*. "Analysis" does not necessarily mean syntactic or semantic analysis only; as already emphasised more than once, these two are closely related, and the term is thus used as a common name for the whole transformation process - from surface text to an underlying meaning representation (dependency structure).

The parsing strategy adopted in (3.2, 3.7) suggests that an ATN interpreter will be required; indeed, this is implemented as the kernel of the program, which gives shape to the overall control structure of the parser. The analysis process is guided by the tests and actions on the arcs of the grammar. The grammar is itself a separate data structure, organised so that a series of operations is defined, which aim to perform the analysis during a strictly left-to-right scan through the text. The emphasis is on the semantic processing, with syntactic constituent recognition taking place in the background and providing the basic raw data for the semantic routines. These are not invoked until enough contextual information has been made readily available as a result of previous processing and conveniently accessible via the registers mechanism. The analyser starts off by pursuing purely syntactic predictions: the initial instruction carried out by the processor is (PUSH S/ ...) which on its own part initiates embedded searches for constituents - possible sentence openers: noun phrases ("John", "John's green crook"), certain types of complements ("To make stupid mistakes"), nominalised ("-ing") clauses ("Shooting elephants"), etc. (see Appendix (iii)). The fact that this may in itself result in initiating an embedded (PUSH S/ ...) computation is irrelevant at this point. What is important is that it is not until the end of the syntactically well formed unit that the appropriate semantic specialist will be called.

As already noted (4.3) there are two major independent semantic routines: NPBUILD and SBUILD. They both check the semantic coherence and well-formedness of a complete syntactic constituent, and deliver the corresponding meaning representation. The points in the analysis process when these could be invoked are suggested by the procedural organisation of the grammar. For example, apart from the obvious grammar fragment

```
(some-state
   (PUSH NP/                    <== initiate search for NP
      ........)
```

---

* occasionally 'the system' is used as well, but this is not entirely precise, as the whole system in reality comprises an analyser and a generator.

```
   ........)
(NP/ ........)              <== beginning of NP network
(NP/POP
   (POP (NPBUILD) T))      <== end of NP network
```

which means "build a semantic representation (if possible) for a noun phrase recognised by the syntactic preprocessor", NPBUILD is to be found in places like

```
........
(NP/HEAD
   (CAT POSS (GETR n)
      (SETR possessor (NPBUILD))
      (TO NP/DET))
   ........)
........
```

or

```
(NP/HEAD
   ........
   (PUSH REL/
      (OR
         (WRD (WHO WHOM WHOSE WHICH THAT))
         (AND
            (CAT PREP)
            (WRD (WHICH WHOM WHOSE) NEXTWORD)))
      (SENDR type 'relative)
      (SENDR rel-nominal (NPBUILD))
      (SETR rel-clause *)
      (TO NP/POP))
   ........)
```

The first of the fragments above constructs the possessor structure in phrases like "the lazy dog's wagging tail"; the second initialises a register for an embedded computation providing a representation for the deleted nominal in the relative clause: "the lazy dog over which the sly fox jumped".

However, no matter at which point of the analysis the NPBUILD specialist is called, it follows a phase of preliminary syntactic recognition, upon completion of which a set of named registers hold pointers to the definitions (semantic formulas) of the surface lexical items which fall within the boundaries of the noun phrase. The task of NPBUILD is to construct the semantic representation of the recognised constituent.

On the one hand, it is clear that in an ambiguous environment which gives rise to alternative readings for the input (i.e. a situation of multiple choice) building the representation is, in general, a non-trivial task. On the other hand, the routine applies the principle of using up all information (see 4.3) available at that

decision point (and at the current level of computation). Normally, a complex noun phrase cannot be disambiguated until top-level clause/ sentence constraints are considered and evaluated. Still, blind carrying over (to the next stage of analysis) of all the possible readings is a bit excessive and unintelligent. In other words, NPBUILD will not necessarily hand over a single structure, but at the same time it will deliver only semantically coherent interpretations within the context of the current level (see 4.3 for semantic parallelism). Only the structures corresponding to these coherent interpretations are constructed dynamically while the routine polls all the registers at noun-phrase level.

The process can be illustrated, for example, by tracing the analysis of "the green crook". At the time when NPBUILD is activated, the syntactic recognition process initiated by (PUSH NP/ ...) is over, and the list of register contents (*REGLIST) contains

```
((det . THE) (adjmods GREEN) (n . CROOK)).
```

On entry to the function, several variables are initialised. *HEAD-NOUN-DEFS holds the information provided by the dictionary indicating the possible semantic content of "CROOK":

```
(  (crook1
      ((((notgood act) obje) do) (subj man)) )
   (crook2
      ((((((this beast) obje) force) (subj man)) poss)
         (line thing)) )   )
```

*ADJ-MODS is set to the list holding the prenominal adjectival modifiers, in this case (GREEN), and ADJ-MOD-DEFS is successively (see below) set to the semantic formulas corresponding to the adjective (premodifier) which is currently the focus of NPBUILD's attention:

```
(  (green1                    %C  as for colour
      ((*inan poss)
         (((man subj) (see sense)) (obje kind)))  )
   (green2                    %C  inexperienced
      ((man poss)
         (((*mar obje) (notmuch (true think)))
            (subj kind)))  )
   (green3                    %C  as in "green with envy"
      ((man poss)
         ((much (notplease feel)) (subj kind))) )   )
```

A basic loop is then set up around the contents of *ADJ-MODS (imagine for example the phrase "the big heavy green crook"). The function of the loop is to try and match the available meanings of the loop 'variable' (or, rather, evaluate them) against the formulas in *HEAD-NOUN-DEFS. An auxiliary function (ADJ-NOMINAL-MATCH) assesses the compatibility between all possible combinations of adjective and noun meanings. Thus within the "GREEN" binding, all the six variants will be analysed:

```
green1 vs. crook1
```

```
green1 vs. crook2
.....
green3 vs. crook2.
```

Closer analysis of the semantic definitions reveals that, for example, "green1" is a characteristic of inanimate objects, to the class of which "crook2", but not "crook1", belong. Thus the following 'connectivity table' will be set up on the property lists of the candidate meaning elements definitions:

```
          green1  green2  green3
crook1              *       *
crook2     *
```

This table is then used for replacing the existing *HEAD-NOUN-DEFS with partially built structures, each containing a noun definition and further augmented with the possible adjective premodifiers:

```
(  (crook1
      ((((notgood act) obje) do) (subj man))
      (@@ state
         (green2
            ((man poss)
               (((mar obje) (notmuch (true think)))
                  (subj kind)))) ) )
   (crook2 (<defn>) (@@ state (green1 (<defn>))))  )
```

In order to get this far, a dynamically created list of possible adj-noun pairs is maintained (and dynamically updated) in the program space, of the form

```
(  (crook1 (mod2 mod3)) (crook2 (mod1)) ),
```

to the elements-pairings of which preference criteria are applied next: since "green3" is part of an adjective frame "BE GREEN WITH *mar" (as for "green with envy"), which obviously does not match the current context, it is unpreferred; hence "crook1" can be modified only by "green2" (see above).

After *HEAD-NOUN-DEFS is updated, the cycle is repeated again for the next adjectival premodifier ("heavy" for example). Upon completion of the selection of adjective definition(s), it is the partial nominal structures contained in *HEAD-NOUN-DEFS that are passed over to the subsequent processing.

Note that such a technique allows an efficient way of searching through the space of all possible modifiers-noun combinations, without actually explicitly constructing all of them. If the next premodifier was "wooden" for example, it would prefer that partial structure in *HEAD-NOUN-DEFS which is organised around the thing definition of "crook". If the noun phrase in question was "John's green crook", the principle of semantic screening (see 4.4) will require the examination of the constraints on possessive structures (4.4) to be performed

before the whole ANALYSE-ADJ-DEFS cycle as described above is initiated - and for obvious reasons, on entry to it *HEAD-NOUN-DEFS will be initialised to "crook2" only. Similarly, an embedded call to SBUILD (see below), needed to process a relative clause, might perform a complete disambiguation of the head noun, and indeed, of the whole noun phrase, thus enabling NPBUILD to return a single structure to the upper level computation. Of course, complete semantic resolution at noun phrase level cannot always be performed, in which case NPBUILD returns the alternative readings, whose further disambiguation will be performed at the next level up, where more contextual information is available.

What has to be emphasised again at this point, is that the NPBUILD specialist is designed in such a way that

(1.) it makes sure that all the data provided by the syntactic preprocessor and the dictionary at the current stage of the analysis is fully utilised (the principle of using up all information - see above); and

(2.) it is very sensitive to the current contextual environment and form of the input surface syntax, and is capable of responding to these in a flexible manner by organising its internal flow of control in a way which reflects the principle of semantic screening (4.4).

The same principles underlie the design of the clause level semantic specialist SBUILD. Although it is not explicitly called from many different places in the grammar, it is the final instruction terminating different analysis paths through the network. The basic clause unit, all types of complements, relative clauses, "-ing" participle phrases,...; all have associated subnetworks, which at some point or other, after performing the necessary preprocessing and initialising certain registers, transfer control to and merge with the basic clause (S/) network, thus inevitably coming to an instruction

```
(S/POP/S
   ........
   (JUMP S/POP/S ........
      ........
      (SETR s-pop-val (SBUILD)))
   (POP (GETR s-pop-val) (GETR s-pop-val)))
```

Note that interpreted strictly according to Woods' definition of the ATN formalism [Woods70], a NULL result of the SBUILD specialist, which corresponds to rejection of a hypothesised clause unit, will block the POP arc, and terminate the analysis path through the network which initiated the search for an embedded clause. This is the practical embodiment of the passive approach to structural disambiguation (see 4.4), and is the reason behind the rather long-winded way of saying

```
(S/POP/S
   ........
   (POP (SBUILD) T))
```

I must point out here that in the actual grammar (see Appendix (iii)) the termination of the NP/ network is organised according to the same

principle - so that the first of the grammar fragments quoted in this section (see above) in reality terminates in the following way:

```
(NP/POP
   (JUMP NP/POP
      (AND (NULLR np-pop-val) (NULLR np-built))
      (SETR np-built T)
      (SETR np-pop-val (NPBUILD)))
   (POP (GETR np-pop-val) (GETR np-pop-val)))
```

This is why during the analysis of

Shooting elephants can be dangerous,

the following sequence of operations takes place:
(PUSH S/...),
(PUSH NP/...),
recognise "shooting elephants",
call NPBUILD,
try to build a structure for "elephants who shoot",
NPBUILD resorts to SBUILD to process the relative,
SBUILD returns NIL,
NPBUILD returns NIL,
the test on the POP arc on NP/POP yields FALSE,
the POP arc is blocked,
(PUSH NP/...) fails,
this particular analysis path of (PUSH S/...) is terminated.

Alternatively, following the same sequence for

Kissing aunts can be boring,

the (PUSH NP/...) returns

```
((trace (clause v agent))
   (clause
      (type relative)
      (tns present)
      (v
         (kiss1
            ((man subj) ((*hum obje)
               (((please feel) goal) (touch sense))))
            (@@ agent (aunt (fem man)) (@@ number many))
            (@@ recipient (someone (man)))))))
```

(where the clause structure is assembled by the SBUILD specialist, and NPBUILD just tidies this up and provides a pointer - via the trace mechanism - to the head noun of the noun group), and the processing at top (S/) level continues.

The SBUILD specialist starts off by polling all the structure holding registers (see 4.2) and initialising certain variables to the semantic content of the major clause components: *SUBJECTS, *VERBS, *MODS, etc. Normally, these are lists containing more than one semantic formula (structure), and the routine will have to select the ones compatible within the current context, and assemble them in a coherent semantic unit. The skeleton of the clause structure is set up next, with marked nodes in it, which will be replaced later by the

relevant semantic representations of the clause constituents. This is the background preparation for both active and passive structural disambiguation (4.4).

SBUILD is sensitive to the various flags set up during the process of syntactic recognition - some of these record the overall surface syntactic pattern of the clause. The types of sentence (clause) structure distinguished here are as defined in [Quirk72], and serve both as an indicator as to which branch of the program (SBUILD) to follow, and as a pointer to the category and properties of the verb (stative/ dynamic, intensive/ extensive, transitive (mono- and ditransitive), intransitive or complex-transitive). This accounts for the switch in the function definition:

```
(COND
   ((FIND-ON-REGLIST 'adj/pred)
      (STRUCTURE-WITH-PRED-ADJECTIVE))
   ((OR
       (FIND-ON-REGLIST 'adj-comp)
       (FIND-ON-REGLIST 'compl-trans-flag))
      (COMPLEX-TRANSITIVE-STRUCTURE))
   ........
   (T
      (STRUCTURE-BASIC)))
```

This is the point at which the disambiguation between "call" in

John called Bill,

John called Bill a fool,

John called Bill a taxi,

effectively starts: the mono-transitive use of the verb stands out, separated from those meanings used in the second and third examples.

The mechanism of contextual verb frame application is central to the design and implementation of SBUILD. It is distributed procedurally between the two equally important functions DISAMBIGUATE-VERBS and FILTER-NOMINALS, which are accessible from all the different structure building auxiliaries of SBUILD (see above). These are handed lists of suggested verb meanings or nominal structures, possible combinations of which are evaluated; the process, however, is not a blind search, but is strongly guided both by the information contained in the verb frames, and by the principle of 'semantic screening'. A 'preference-weight', and 'connectivity pointers', are associated with every meaning candidate and updated, as indicated by the results of semantic tests, in the process of 'screening' through the series of semantic filters (see 4.4).

Imagine the sentence

John asked the girl for the club

is to be analysed. At this point of the analysis the main constituents of the clause are conveniently accessible via the registers mechanism: 'subj', 'obj', 'mods' hold results of embedded computations (PUSH NP/...) or (PUSH PP/...); 'v' holds a pointer to

the dictionary entry for "ask". The questions that SBUILD has to answer are which meanings of "ask" and "club" are to be incorporated in the final dependency structure, and what is the functional relation of the phrase "for the club". The contextual verb frames for "ask" (see 3.9) are supplied by the dictionary, and this is what *VERBS is set to:

```
(ask1
   ((man subj) ((*ani obje) ask))
   (cues INDOBJ)
   (preps (ABOUT *ent subj-matter)))
(ask2
   ((man subj)
      ((act obje)
         ((((man (please feel)) cause) goal) ask)))
   (cues TOCOMP))
(ask3
   ((man subj) ((*ent obje) want))
   (compulsory (FOR *ent object)))
```

After the embedded call to NPBUILD from within the (PUSH PP/...) directive, *MODS now holds the possible semantic interpretations of "club":

```
(club1
   ((((((notsame man) obje)
           ((notplease feel) cause))
         (subj man))
       poss)
      (line thing)))
(club2
   (((this man) obje) wrap) spread))
```

The following program fragment (the essential part of DISAMBIGUATE-VERBS) can be used to illustrate the process of frame application (MAPC is a LISP system function which applies the function specified as its second argument to every element of the list supplied as the first argument of MAPC):

```
(MAPC *VERBS (FUNCTION FORCED-CHOICE-PREP))
(MAPC *VERBS (FUNCTION DELETE-SPECIAL-VERBS))
(MAPC *VERBS (FUNCTION ANALYSE-CUES))
(MAPC *VERBS (FUNCTION FORCED-CHOICE-COMPL))
(MAPC *VERBS (FUNCTION MATCH-CASE-PREFERENCES))
(MAPC *VERBS (FUNCTION MATCH-TEMPLATE-REQUIREMENTS))
(SETQ *VERBS (ITEMS-WITH-HIGHEST-PREFERENCE-FROM *VERBS))
```

The function DELETE-SPECIAL-VERBS 'unprefers' those verb meanings which are obviously incompatible with the current syntactic environment: for example during the analysis of the last two example sentences with "call" above, it will 'delete' all meanings of the verb which cannot be followed by two objects; in an environment where 'pred/adj' flag is unset, it will 'delete' the meaning of "grow" which

is tagged COPULA, etc. It should be sufficiently clear what the other functions do. All of them embody certain tests which inspect the semantic content of the items. The application of these tests is determined by the specific interest of the function currently active; the results of the tests control the setting and resetting of the 'preference weight' of each item. ITEMS-WITH-HIGHEST-PREFERENCE-FROM is another function accessible from many places within the SBUILD specialist, which embodies the principle of maximum semantic coherence: "densest match wins" (see 4.4).

For the example above, FORCED-CHOICE-PREP senses the requirement of the third frame for an obligatorily used "FOR" prepositional phrase, and since such a phrase has been identified by the ATN preprocessor, a check is made to see whether its semantic content is compatible with the contextual requirements of "ask3". The preference weight of the semantic items (formulas for "ask1", "ask2", "ask3", "club1" and "club2") are reset accordingly, and the final result of the (disambiguation) process is

(*) selection of the verb meaning:

```
ask3
  ((man subj) ((*ent obje) want))
```

(*) selection of the noun meaning:

```
club1
   (..... (line thing))
```

(because thing is an entity, and spread is not), and

(*) specification of the functional tie (dependency link) between "ask" and "club": object.

Further analysis of the semantic content of the available constituents, carried out within the context of the already established verb meaning (see 3.8) suggests that "John" is the agent, and "the girl" - the recipient of the asking. The result of SBUILD is

```
(clause
   (type dcl)
   (tns past)
   (v
      (ask3
         ((man subj) ((*ent obje) want))
         (@@ agent (John (mal (indiv man))))
         (@@ recipient (girl (fem man)))
         (@@ object (club1 (..... (line thing)))))))
```

Note that in the process of semantic analysis no attempt was made to attach "for the club" to the immediately preceding item, "the girl", even though on purely syntactic grounds this would be a well formed noun phrase. Nor was there an explicit enumeration of all the possible pairs 'ASK_i' and '$CLUB_i$'. The routine was able to carry out its structural and judgemental decisions with minimum effort and blind search. If the input was

John asked the girl a question,

it would be the function ANALYSE-CUES which would connect a candidate meaning ("ask1"), as specified in its contextual verb frame

*hum ASK1 (*hum) (@sign) ...,

with the surface syntactic pattern form - (cues INDOBJ) will match the 'indobjflag' set by the syntactic preprocessor. Further, on establishing a complete match between the @sign specified in the frame and the semantic formula (head) for "question" (tell sign), it will prefer strongly this particular meaning of "ask". Similarly, in the case of

John asked the girl to give him the the book,

it will be FORCED-CHOICE-COMPLement which will be woken up by the presence of 'tocompl' register. It will increase the preference weight of "ask2", and will suggest mobject (mental object) as the dependency link between the verb and the embedded clause, thus yielding

```
.....
(v
   (ask2
      (@@ agent: John)
      (@@ recipient: girl)
      (@@ mobject (girl give John book)))).
```

(Note that at this level of analysis, 'tocompl' register will hold a complete semantic representation of the embedded clause, which will be handleable as a single semantic unit: this has been constructed by an embedded call to SBUILD at some earlier stage (lower level) of processing. The deleted subject ("girl") of the embedded clause is passed down at recognition time after consulting the features list on the verb definition - "ask" is marked both TOCOMP and TRANSCOMP - see 4.2.)

It was noted earlier (3.9, 3.10) that the frames mechanism takes care of obligatorily used prepositional phrases and other postmodifiers. Optionally used prepositional phrases are parsed by MODIFY which is the third major semantic specialist employed by this system. It is not an independent one, because it runs after SBUILD or NPBUILD have completed most of the ground work (i.e. set up the basic skeleton structure of the semantic unit being analysed) and can only be called from within one of these. MODIFY is the device which carries out (most of) the active structural disambiguation (4.4).

The information about optionally used prepositional phrases is distributed: some of it can be found in the verb frames:

*hum ASK1 (*hum) (@sign) (about *ent);

and some in the preplate stacks (see 3.10 and below). The MODIFY specialist therefore consists of two auxiliary programs which run one after another. The actual definition of the function is

```
(DE MODIFY (PARTIAL-STRUCTURES)
   (EXTRACT-FIT-MODIFICATIONS
      (CONC-TOGETHER
         (MAPCAR
            (INITIAL-PREPLATE-TIES PARTIAL-STRUCTURES)
            (LAMBDA (STR/MODS)
               (COMBINE
                  (CAR STR/MODS) (CDR STR/MODS)))))))).
```

Without going into specific details (EXTRACT-FIT-MODIFICATIONS and CONC-TOGETHER are auxiliary functions which perform some post-processing on the result of the modification process; and MAPCAR is another standard LISP function, similar to MAPC, but returning as a result a list of the results of the individual function applications), the important parts of the definition above are INITIAL-PREPLATE-TIES and COMBINE.

The task is: given a list of partial skeleton structures, try to further augment each one of them with the postmodifiers from the 'mods' register (*MODS). Suppose the sentence to be analysed is

John asked the girl about the book.

PARTIAL-STRUCTURES will then contain

```
(  (ask1
      ((man subj) ((*ani obje) ask))
      (@@ agent (John (mal (indiv man))))
      (@@ recipient (girl (fem man))))
   (ask2
      (<defn>)
      (@@ agent (John (....)))
      (@@ recipient (girl (....))))   ).
```

"Ask3" is missing because the frame requirements have not been met (see the principle of applying a more discriminating disambiguation technique first). INITIAL-PREPLATE-TIES then examines for each prepositional phrase in *MODS its compatibility with each context as specified in PARTIAL-STRUCTURES, and in the frame for the corresponding leading verb. In the case of a match, for example for "about the book" - "book" being an entity, and 'ABOUT *ent' being specified in the verb frame for "ask1" - the partial structure is promptly augmented with

```
(@@ subj-matter
   (book
      ((((line sign) obje) wrap) (subj thing)))).
```

The preference weight of the partial structure in question is correspondingly increased, leading to the eventual prefering of "ask1" as the intended meaning of "ask" in the input sentence. The modifier "about the book" (or, rather, the semantic structure corresponding to it) is deleted from *MODS, which at any given moment of time should hold only unused postmodifying prepositional phrases.

After INITIAL-PREPLATE-TIES has completed its work, the results are kept in a list of pairs (these are indicated by STR/MODS in the function definition). Each of these contains in its CAR a (possibly augmented) partial skeleton structure, and in its CDR those prepositional phrases in *MODS which are still unused. In some cases no clues to postmodification will have been found in the verb frames, so no augmentation of partial structures will have occured; in other cases complete analysis of the modifiers will have been achieved through the frames application mechanism, and the CDR of the pair will contain NIL. Any intermediate situation is possible as well.

Every pair on the list returned by INITIAL-PREPLATE-TIES is passed to COMBINE: this is the function which embodies the general preplate matching mechanism.

The problem is: given a partial semantic structure and a linear list of postmodifying prepositional phrases (see 3.7, 3.10), construct those structures which are semantically valid from the point of view of the level and scope of the modification. The basis of the algorithm is the well established fact that in English a postmodifier has a tendency to be dependent on the constituent that is closest to it (and preceding it). It is expected that a fair number of the exceptions will be specified by the frames and trapped by the mechanism for frames application, so this is a conveniently general assumption. The algorithm consults the preplate stacks, which are indexed by the prepositions, extensively; the stacks are in fact the dictionary entries for the prepositions:

```
(BY
   (move @inst (move thing))          fly by plane
   (*do @timerel (when point))        return by the evening
   (*do @manner act)                  inform by writing
   (*ent @loc/static *inan)           the man by the wall
   ........)
(FOR
   (make @recipient *hum)             make dress for Mary
   ........)
(TO
   (*utter @recipient *hum)           talk to the crook
   (move @loc/dynamic *pla)           go to the club
   ........)
(WITH
   (*do @manner *mar)                 talk with enthusiasm
   (*ent @attribute *ent)             man with a telescope
   (*assault @inst *inst)             force with a knife
   ........)
```

The examples above should make the general format of the preplates clear.

The algorithm itself, as carried out by COMBINE, is a procedural embodiment of the following basic loop:

```
WHILE the list of unused modifiers is non-empty
DO $( take the next modifier on the list;
      TEST its compatibility with the constituent
           immediately preceding it,
      THEN link the two with the corresponding
           dependency tie;
           REPEAT
      ELSE TEST its compatibilty with the constituent
                one level up,
           THEN link; REPEAT
           ELSE ........
   $)
```

Provision is made, however, for situations when a modifier can equally well modify more than one preceding constituents. These might generate genuinely ambiguous structural interpretations, and according to the rules of the game (see chapter 0) must be accounted for.

It is clear that this approach embodies the active strategy for structural disambiguation (4.4): only semantically valid and coherent structures are dynamically constructed; a failed preplate test precludes any structure building, and in addition completely blocks the application of whole series of subsequent preplate tests. In principle, the algorithm above could be modified to apply the preplate tests as soon as a prepositional phrase is recognised by the front end scanner; there are two reasons why this is not done. First, this approach clashes with the general underlying strategy of pattern matching; second, the procedure relies on augmenting, but not initially constructing, a partial semantic structure - which cannot be made available until SBUILD (NPBUILD) has been invoked; and this, as already noted, does not happen until 'end of clause/ noun phrase' is broadcast by the scanner.

Note that the algorithm does not attempt to modify verbs or nouns only - it augments partial semantic structures by examining their content, rather than by worrying about their syntactic category membership. The preplate stacks provide information about both verb and noun postmodification (3.10). A single call to MODIFY at clause level will suffice to account for all the possible relationships between the verb and all the nominal groups following it. Thus the only time when MODIFY will have to be called from NPBUILD will be when constructing a semantic representation of a noun phrase in subject position, with prepositional phrases having been recognised before the main verb is reached ("the man in the park with the telescope saw me").

It is important to realise that the whole process of preplate tie application is carried out on alternative partial semantic structures. As the attempt is made to integrate more and more information into an already constructed, and hence coherent, semantic unit, there is a greater chance that an incompatible reading of a lexical item will clash with some already processed data and thus block a path being pursued by the semantic analysis processor.

As already noted (4.4) the processes of lexical and structural

disambiguation run in parallel. The principle of maximum semantic coherence is the one mostly responsible for the reasonably efficient parsing of

John struck the girl on the bank on the head with a club.

Considering that the dictionary contains 3 definitions of "strike" and "bank", and 2 for "head" and club", it is clearly going to be disastrous to attempt evaluation of

(3*3*2*2)*(14 structural interpretations)=504

alternative readings. The frames mechanism almost immediately rules out a meaning of "strike" - as in "it strikes me that ...", or "the idea/ decision/ solution strikes me as ...", and in addition strongly prefers a meaning

```
((*ent subj)
   ((man obje)
      ((*inst inst)
         ((((notplease feel) cause) goal) strik))))
```

because it is a part of a frame
*ent STRIKE1 @man (ON (body part)) (WITH *inst),

which is suggested by "on the head". Of course, the process of frame application also disambiguates "head" as (body part) as opposed to *place, and establishes the functional link between "hit" and "head". Further, INITIAL-PREPLATE-TIES will match "with the club" onto the same frame, thus confirming all the decisions made up to that point (and recording this by increasing the preference weight on the corresponding semantic readings); it will also disambiguate "club" as a thing (which is a member of *ent), as opposed to "club" as a place. At this point control is passed over to the more general preplate tie mechanism (COMBINE), which has only to decide on the meaning and function of "on the bank". The preplate stack under "ON" will be consulted, where the following entries will be found (among others):

```
........
(*ent @loc/static (where point))
........
(*do @loc/static (where point))
```

This will rule out the meaning of "bank" as an organisation (grain), or "bank" as a place (spread) where money (get sign) is kept, leaving us with the "river bank" meaning, and the corresponding functional ties (@loc/static in both cases). The corresponding dependency structures will be

```
(1)  ........
     (v
        (strike1
           ((*ent subj) ((man obje) ........ strik))
           (@@ agent (John (....)))
           (@@ recipient (girl (....)))
           (@@ loc/static (bank1 (.... spread)))
           (@@ loc/dynamic (head1 (.... (body part))))
```

```
            (@@ inst (club1 (....(line thing))))))
```

and

```
(2)  ........
     (v
        (strike1
           ((*ent subj) ((man obje) ........ strik))
           (@@ agent (John (....)))
           (@@ recipient
              (girl
                 (....man)
                 (@@ loc/static (bank1 (....spread)))))
           (@@ loc/dynamic (head1 (.... (body part))))
           (@@ inst (club1 (.... (line thing))))))
```

In situations like the one above, it is desirable to have some criteria on which a choice of a 'most likely' interpretation could be based. The actual choice mechanism has not been implemented yet; this is why the results of the analysis process, if more than one, come out in an unspecified order. However, it is possible to define heuristics which could be applied in situations like these. They will rely on the fact already mentioned above: a postmodifier tends to depend on the constituent that is closest to it. In the process of dynamically constructing the semantic representation of the clause or noun group by augmenting it with the items on *MODS list, each optionally used modifier will be tagged with a number, a 'jump-over factor', which can be greater than or equal to zero. This factor will show whether the constituent it tags modifies the one immediately preceding it (jump factor = 0), or the one preceding that (jump factor = 1), and so on. These factors will be used in generating a 'modification measure' of the whole structure: a vector $M(D_1,D_2,...,D_n)$ in which D_j denotes the number of constituents that are separated from their governor by j intermediate items. The two structures generated above will have measures M1(1,1,1,1) and M2(1,1,1,0). Thus, on the basis of a suitably defined metric, the details of which need not concern us here, reflecting the obvious desire for a semantically tight structure, the preferred interpretation will be (2), where "the girl on the bank" is a single semantic unit. The algorithm should analyse the tightness of optionally used modifiers only, and can further be made sensitive to the density of a match between a head and its dependent as specified by the outcome of a semantic test.

The MODIFY specialist in its current version deals only with prepositional phrases. In principle it could be extended to account for other optional postmodifiers as well, but for programming ease and convenience, a separate function has been introduced: CONVERT-COMPLEMENT-TO-CASE. This is handed a complete dependency structure - a representation of the embedded clause present in any type of complement, an "-ing" phrase construct, etc. The function tries to integrate this representation within the skeleton structure currently being worked on: the proper place for its attachment has to be found, and the dependency link established.

As already shown, some of the instances of postmodification involving embedded clauses will be trapped by the frames mechanism:

John stopped calling on Mary,

John made Bill strike Jill,

John admitted to killing Janet,

John made an effort to work in the garden.

It is the more general cases like

Bill decided that he wants to kill Fred,

Bill made a gun to kill Fred,

Bill killed Fred firing a gun,

I watched the man teaching Mary,

that have to be analysed here. Note that in the case of the last example the ambiguity does not arise at this point of the analysis, but is due to the different paths through the grammar:

```
(NP/HEAD
   ........
   (PUSH ING/RELATIVE (CAT VERB)
      (SENDR subj (NPBUILD))
      ........) ...)
```

will parse "the man teaching Mary" as a noun phrase, whereas

```
(S/POP/S
   ........
   (PUSH ING/PHRASE (CAT VERB)
      (SENDR subj (GETR subj))
      ........) ...)
```

will parse "teaching Mary" as a postmodifying ING-phrase.

The decision-taking process(es) of CONVERT-COMPLEMENT-TO-CASE rely on analysing the surface syntactic pattern, which points to general rules (part of the semantic knowledge embodied in the specialist):

```
*hum *do(stative) @act *do-ing @act ==> @while

*hum *do(dynamic) @act *do-ing @act ==> @manner,
```

in conjunction with the semantic content of the verb involved (see 3.10). As a result, the following structures are derived:

```
(kill
   (@@ agent: Bill)
   (@@ recipient: Fred)
   (@@ manner: (Bill fire gun)))
```

and

```
(watch
   (@@ agent: myself)
   (@@ recipient: man)
```

```
          (@@ while: (I teach Mary)))
```

(together, of course, with

```
        (watch
           (@@ agent: myself)
           (@@ recipient: man (who teaches Mary)));
```

see above.)

As this strategy was discussed in more detail in 3.10, the examples will not be elaborated further here.

chapter 5.
The Generator.

5.1. A Background For Paraphrase.

The natural language analysis system implemented and discussed here comprises a generator as a back end, performing postprocessing of the semantic dependency structure delivered by the analyser. Although the primary emphasis of the research has been on the process of sentence analysis, the generation phase is nonetheless important for two reasons. Firstly, within the framework defined by the overall objective of (machine) translation, it is necessary to demonstrate that the parser produces an internal representation of the input text which is

(*) correct from the point of view of sentence interpretation,

(*) adequately coherent as a semantic representation unit, and thus handleable by a separate and independent information processing program like the text generator,

(*) sufficiently distant from the input sentence not to impose unnecessary restrictions on the generator by the surface ordering of the input words,

(*) informative enough in meaning and functional labelling to allow for the smooth operation and performance of the generator.

Secondly, a text generation program not only eliminates the tedious job of manually examining the results of the analysis process: it allows much more reliable evaluation of the analyser output. Wildly different underlying meaning representations can look equally plausible; it is at any rate not easy to decide whether one is better than another. It is somewhat easier to decide whether a derived natural language representation is acceptable. Sentence generation from the dependency structure, in fact, automatically 'deciphers' the structure, reflecting in a naturally comprehensible form all the points of interest for text understanding, i.e. those features of the input providing challenges for the analyser.

It is clear that in the general framework of "design and implement a language processor capable of performing analysis of text of 'average' complexity*", a program working in "translate" mode can demonstrate the system's understanding capabilities, as well as providing an environment for the analyser design. The parser has been developed with this translation objective in mind: the front end of the system is geared towards solving the problems relevant to the task of translation - word disambiguation, alternative structure elimination, pronoun resolution+, and the generation of unambiguous internal representations. To prove that these problems are solved correctly, the system could generate either Bulgarian, or English (my knowledge of other languages being virtually zero). Considering the unacceptability of Bulgarian as a generally accessible test target language, the system actually performs as an English paraphraser.

---

* see appendix (iii) for the range of constructs handled

\+ not implemented; not even considered in detail

The decision to use English rather than Bulgarian as the target, did not, however, influence the design of the analyser. The system 'translates' English into English and this, as will be discussed in more detail below, is a process which is conceptually no different from translating English into Bulgarian, or any other language for that matter. Thus in order to generate Bulgarian as opposed to English output, for example, the only need would be for specific Bulgarian linguistic data. There would be no need to change the control structure and mechanisms of the generator, which would operate on the same intermediate semantic representation as for English.

What is important here is that as the system actually performs in 'paraphrase' mode transforming the analysed dependency structure back into English, three specific questions have to be considered:

(1) What types of paraphrases to handle: i.e. how should the range of paraphrases delivered by the system be delimited.

(2) How to define a good paraphrase.

(3) How to get more interesting paraphrases - what mechanisms will be needed.

Analogous questions of course apply to translation, but the questions about paraphrase are sharper: a boring translation is still a translation, but paraphrasing a sentence by itself, however genuine the processing involved, is not very convincing.

It is clear that the basic purpose of the paraphrases in the stated environment is to show that the analyser has been right in its disambiguation decisions, for example in treating the word "make" in the following examples in different ways:

John made a gun to kill Mary.

John made an effort to kill Mary.

John made Bill kill Mary.

Thus paraphrasing could show this by delivering, for example

John produced a gun in order to ...

John performed an effort ...

John forced/ persuaded Bill to ...

Such paraphrases may not be stylistically optimal, but are all that are needed to prove that in a translation environment the desired effects can be achieved.

The following criteria for 'successful' paraphrase might thus be specified:

(i) a paraphrase should be a grammatically correct sentence;

(ii) a paraphrase should express the correct reading if there is only one;

(iii) in cases of genuine ambiguity, a set of disjoint (not alternative) paraphrases should reflect the competing dependency structures, one for each reading.

Closer analysis of (i) and (ii) brings us back to the first ques-

tion stated above: what sort of paraphrases to produce? The possibilities range from simple single word replacement (lexical item substitution):

John struck Mary => John hit Mary,

to message reformulation of a much more extensive kind, perhaps involving explicit inference or summarising:

The President declared that he could not commit himself to endorsing ... =>

The President refused to endorse ...

Clearly the question is where to draw the line between the 'trivial' paraphrase and the 'glossing' paraphrase. As noted above, the purpose of the paraphrases is to demonstrate the adequacy of the dependency structures delivered by the analyser, and thus prove the correctness of the analysis. In practical terms this implies that the paraphrases must at least demonstrate

(1) that the analyser has selected the correct word senses (whenever lexical ambiguity is present in the source text); thus the generator has to be synonym driven i.e. it has to be supplied through a dictionary with at least one and preferably several synonyms for each (input) English word sense;

(2) that the analyser has established a proper structural analysis by determining the scope of modifiers and their level of modification:

John searched the room carefully =>

different sentences reflecting the fact that "carefully" can appear after any of "John", "searched", and "the room".

Similarly,

John saw Mary in the street at 5 o'clock =>

John saw Mary at 5 o'clock in the street.

but not

John saw Mary in the street with the two hotels =>

John saw Mary with the two hotels in the street.

Thus, after the analyser has identified the constituent units, the generator must be able to reshuffle them to clearly display the identified constituent relationships. The generator must therefore be able to manipulate whole constituents, which will demonstrate the correct understanding of the text by the parser, and the ability of the output routine to handle, and be organised around, connected pieces of meaning;

(3) that the output routine can generate correct English, and does this by manipulating not the input text itself, but the structure into which it has been parsed. This will also prove that the program is potentially able to generate (comparatively easily) some other language than English;

(4) that the generator can select synonyms appropriate to the particular sentence context from any set of synonyms supplied for an

input word sense:

John asked Mary => John questioned Mary,

John asked about ... => John inquired about ...,

John asked Mary about ... => John inquired about ... from Mary,

John asked to go home => John wanted to go home,

John asked Mary to go home => John begged Mary to go home.

Thus the generator has to be context driven as well.

These points above make it clear that what we need are synonym based, context-sensitive, grammatically correct (and meaningful) underlying representation transformations. It is further clear that paraphrases with the properties indicated can only be produced by a quite sophisticated generator capable of, for example, clever constituent manipulation. However, paraphrases like

John killed Mary => Mary was killed by John,

It was yesterday that he came => He came yesterday,

while they may exhibit some or all of those properties just discussed are clearly just not radical enough. It would be desirable to have generation of a more dynamic character allowing paraphrases of an extensive, rather than limited, kind. Thus the sentences in pairs like

John strikes me as pompous => I regard John as pompous,

I liked the play => The play pleased me,

John bought the book from Bill => Bill sold the book to John,

Fred struck Max => Max received a blow at the hands of Fred

(examples from [Chomsky65]), are clearly sufficiently strongly related in meaning to be accepted as paraphrases, though they have little surface structure or vocabulary in common: they can be described, informally, as interesting rather than boring paraphrases because they involve both semantic and syntactic variations of the original. An attempt to produce paraphrases of this more interesting kind would be a quite strong test for the language processor as a whole; because in order to produce the right hand side members of the pairs quoted above, the analyser must, before anything else, 'understand' properly their counterparts on the left.

We must therefore aim, in the range of paraphrases generated, for more extended treatment of

(a) context sensitive word variance and synonym substitution,
(b) constituent restructuring ("... be afraid to ..." => "... not want to ...")
(c) constituent manipulation and reshuffling,
(d) variance at the level of whole components: "strike" (in a particular sense) can be replaced with "receive a blow at the hands of", "prefer" - with "would rather", etc.

The generator developed here does not attempt anything in the most exotic end of the spectrum like providing summaries or spelling out inferences. However, it is not confined to mere syntactic rephrasing or single word substitution.

The intermediate at which the generator operates can be illustrated by the case of clause transposition in complex sentences. This does in fact represent constituent manipulation, but it introduces some essentially new points. Clearly, there is nothing interesting in

By walking fast, he got there on time =>

He got there on time by walking fast.

On the other hand, intensive complementation, or modification by "-ing" phrases, or certain other constructions, require more attention. Consider, for example,

```
        |bored     |
I was   |furious   | to hear about it  =>
        |indignant|

                     |bored me            |
To hear about it     |infuriated me       |  ==>
                     |          |bored    ||
                     |made me   |furious  ||
                     |          |indignant||

   ||bored     |         |
   ||infuriated| me      |
It |          |bored    || to hear about it
   |made me   |furious  ||
   |          |indignant||
```

(example from [Quirk72]). It is clear that an essential part of the paraphrase operation generating any of these alternatives is dictated by the properties of the predicate adjectives involved; but much of it is represented by syntactic changes. In other words, the syntactic changes are entirely dependent on the adjective/ verb sense, and the identification of the adjective/ verb sense involved is largely a semantic process. This selection is indeed one of the main tasks of the analyser.

I have included such clause transformation paraphrases in the list of transformations to be carried out by the system, which may therefore produce output like

John is easy to please => It is easy (for someone) to make John happy,

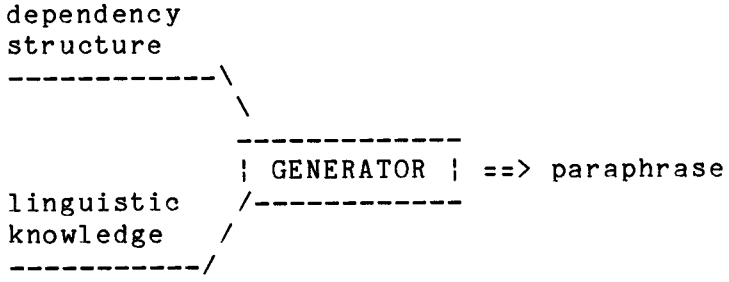
John is eager to please => John wants very much to make somebody happy.

This brings us to the last of the three questions set in the beginning of this section: how to produce these more interesting i.e. more dynamic paraphrases.

There are so many factors determining the nature of the paraphrase. There are things like the format of the dependency structure delivered by the analyser, and its content. There is also the whole apparatus of the output routine, i.e. the generator machinery, plus the linguistic knowledge which, together with the input dependency structure(s), drives the generator*.

The format of the dependency structures has already been discussed in detail (3.11): As to their content, it is arguable how much of, and how, it should be utilised in the generation process. The argument can be put in the form of the contrast between the approaches adopted by Simmons and Slocum [Simmons72] and by Goldman [Goldman75]. Simmons and Slocum preserve traces of the input text in the internal representation and utilise these in generation. Goldman sets out to eliminate traces of the input text since he maintains that internal representations must be freed from their text manifestations. Some of the points relevant to this argument, regarding the semantic content of the structures and the traces of surface text in them, have already been discussed in 3.11; the others are considered below. Overall, the case for manipulating the dependency structure not by individual components but by constituents, should be clear by now.

It is clear that the major factors determining the nature of the paraphrase are the generator control structure and mechanism(s), together with the linguistic data with which it is supplied. In general, the scheme



defines a quite rigid process. In order to be able to produce the more dynamic paraphrases as discussed above, this rigidity has to be broken, or at least relaxed. Two variables can be manipulated to bring that effect: we can either make the generator more flexible, or make the linguistic knowledge more versatile. As a matter of fact, these two factors are interdependent, for it is obvious that flexibility in the program's performance is impossible without rich data; and on the other hand, extensive linguistic knowledge cannot be correctly interpreted and fully utilised by a rigid non-imaginative program. However, for the time being I shall consider them separately.

---

* the reader is reminded that the system does not attempt the kind of inferences needed for

> John strangled Mary => John caused Mary's death by preventing her to inhale air,... [Rieger75].

The extensions to the generator structure are largely a computational problem, rather than a conceptual one. Nonetheless, the organisation of the control structure does require certain heuristics. The issues involved are mainly concerned with the problems of organising and structuring the output sentence:

(*) how to organise the information in the dependency structure into constituents - a constituent (in the target sentence to be generated) being a self-sustained conceptual unit (although this is a recursive definition: "the man in the park with the magnolias which..."),

(*) how to realise this information in the target language (surface text): i.e. which of the following to say

John's delivering of a speech at the Union amazed me,

I was amazed that John delivered a speech at the Union,

That John delivered a speech at the Union amazed me.

(*) how to structure the constituent components into a larger unit - mainly a problem of ordering:

The train had arrived (quietly) (at the station) (during our conversation).

Similarly, the linguistic knowledge available to the generator has to be enriched. This can be achieved mainly in two ways. Firstly, by developing and introducing some relevant heuristics in the control structure of the output routine (see above). Secondly - by designing the dictionary (a separate one for the generation phase) in such a way that the individual dictionary entry is quite informative, and in the same time very flexible. Take, for example, a verb entry. As discussed in chapter 3, every verb defines one or more verb frames. Each of these has certain syntactic requirements when it comes to alternative i.e. synonymous expression of the same 'message'. The entry for that frame should, therefore, state this rigid and compulsory syntax, but in the same time should, somehow, make provisions for the optional constituents (see 1.2, 3.10) which might, or might not, be present, and even if present, could be manipulated more freely. In other words, most of this problem is reduced to defining a format for a more dynamic dictionary entry.

Thus, the general framework for the design of the generator program is defined by

(A) introduce context-sensitive, surface syntax oriented linguistic knowledge in the dictionary, and

(B) make provisions for its correct utilisation in the process of re-expressing the meaning of the dependency structure in English.

5.2. What The Generator Does.

This section attempts to discuss in general terms the basic generator design considerations and decisions in the light of the task "sentence paraphrasing". A program to generate natural language from an internal meaning representation structure is, on a more abstract level, just a device for carrying out a representational change. (Which is hardly surprising, considering that generative grammarians [Lakoff72] suggest exactly this for generating natural language sentences (in this abstract sense of generation): start with a deep (semantic) structure and apply an ordered sequence of transformations until a surface string of words (phonemes, graphemes) is derived.) These discussions have not, however, provided much usable detail on what information, and in particular semantic information, the internal representation should contain, and how it should be utilised by the generation process. In computational linguistic research, too little attention has been paid to language generation. (The reasons for this vary: from "the primary interest in language processing lies in understanding, and this puts the emphasis on the analysis phase", to "many natural language analysis systems require little or no generation at all - LUNAR is probably the best example in this respect"; c.f. Goldman in [Goldman75]). The fact, however, remains that not many generation programs have been developed, and probably the one with the highest level of performance (or apparently highest level of performance) is MARGIE's BABEL [Goldman75]. This program incorporates some very unorthodox concepts, compared to the other efforts in the same direction like Klein, Winograd and Simmons [Klein65], [Winograd72], [Simmons72]:

(1) BABEL starts without a conventional syntactic representation of the sentence to be generated,

(2) BABEL starts without knowledge of the words to be used in the surface sentence.

In the light of the overall design objective of MARGIE - conceptual information processing - this is presumably the correct approach. That is, if Schank and his colleagues really want to prove the point that there is a conceptual base into which utterances in natural language are mapped during understanding, the input to the generator has to be 'language free'. However the design of my generator has been influenced by the rather different goal of my system, namely analyser testing.

The tasks performed during sentence analysis are
(1) resolution of lexical ambiguities,
(2) resolution of structural ambiguities,
(3) formation of a dependency structure.

As already explained, the dependency structure contains information about the meaning of each constituent and its function in the overall textual unit, and hence about the meaning of the whole

sentence. The basic principle of BABEL is that "it is the meaning of the generated sentence which must be specified and utilised to guide generation" [Goldman75]. It appears that the dependency structure, as it is, (see 3.11) can serve both as a result of the analysis, and a source for generation. This in itself is an extremely useful and important fact, but a close investigation of the nature of a dependency structure makes it clear that in order to generate natural language from it, a program is needed which accepts a meaning specification without a conventional syntactic specification. The dependency structure has no place for conventional syntactic concepts like 'indirect object', 'prepositional phrase', or 'complement'. If it had, the generator (among other things) would be restricted in its manipulation and ordering of the structure constituents, would possess very low resolution power, and would probably be very clumsy in design in order to compensate for the need to allow overt syntactic transformations. As it is, the generator must have ways of deciding on the correct syntactic representation of the dependency structure case relations like the 'recipient' or 'manner' relationship between the action element and a semantic sub-unit in the dependency structure.

Further analysis of the parser objectives (1-3 above) highlights a point discussed in the previous section: generation should serve as a test for the correctness of the dependency structure. Thus it should first show that the (word) disambiguation process has been successfully completed. It is very important to realise this because it has direct implications for the operation of the generator. Imagine the input consisting of

(*) John asked Mary about the book,

(**) John asked Mary for the book,

and assume that the analyser has correctly distinguished between the two different meanings of "ask":

```
(ask1 ((man subj) ((*ani obj) ask)))
(ask3 ((man subj) ((*ent obje) ((*hum from) want)))).
```

This distinction should be reflected in the generator's output. And while the effect will be automatically achieved in 'translate' mode, the program's flow of control in 'paraphrase' mode must be driven by a mechanism of 'synonym selection'. In other words the generator must seek, and find, "inquire" and "request", for example, in order to emphasise the difference between "ask1" and "ask3" in (*) and (**) respectively.

This was mentioned in the previous section, but is emphasised again here, because it is the initial point for the generator design: start with the dependency structure and produce a paraphrase of the input sentence, centered around the relevant synonyms of the words originally used.

There are several major problems to be dealt with in achieving this:

(1) Choice of synonyms/words to be used in the generated sentence. Obviously, the program should be able to make intelligent

use of the words it knows about in order to ensure that the generated sentence does not depart from the meaning of the dependency structure, and hence from the input sentence. Of course, the notion 'paraphrase', as it stands, is somewhat vague, and unfortunately there is no strict definition of what should be accepted as good paraphrase of a sentence, especially an isolated, single one. One man's paraphrase is another man's corruption of meaning, so I shall rely on the reader's intuition and common sense.

(2) Definition of the target language (English) syntactico-semantic relationships which tie the words chosen under (1) together. The syntactic relationship of the words in a textual unit sets up the rules for their combination and interaction in the output; the semantic relationship between conceptual units defines their ordering within a coherent whole; both provide the information needed to drive the

(3) Actual output of the generated target language sentence.

The alert reader will notice that this approach to generation is somewhat like that of Goldman's BABEL. This similarity is due to the fact that Goldman's program is an advanced one, and hence has some important points to make about the process of automatic sentence generation. However, on a more practical level, there are some considerable differences, which will be clarified later. The list of problems just given indicates both the relative importance of these points for the smooth performance of the generator, and the order in which their solution is attempted, following the program's flow of control. The next sections analyse and discuss these separately.

5.3. Choice Of Words.

5.3.1. Background

The process is mainly concerned with the selection of the verb around which the generated sentence will be organised. There are several good reasons behind this approach. Some can be found in the way in which the parser is structured and designed, others follow from the specific requirements of the generator.

The whole analysis process is organised under the fact that "the central element of the clause is the verb" (3.8). This is reflected first, in the structure and contents of the test dictionary, which is intended to provide an adequate, i.e. real dictionary sample, and as a result, is predominantly verb oriented; second, in the flow of control throughout execution; third, in the algorithm(s) for building the dependency structure, as well as in its format; and fourth in the introduction and use of the notion of a verb contextual frame as a device for speeding up and facilitating parsing. In fact, the whole initial phase of the system, and in particular the semantic routines, which are the backbone of the analyser, are verb driven. This clearly suggests an attempt to organise the generator along the same lines.

However, there is a stronger and more immediate reason for deciding on the verb selection as the first and foremost step in the generation process. It is closely connected to the next problem waiting in line - the definition of the syntactico-semantic relationships to be present in the output sentence - and, in fact, could be justified with the same arguments as those used for the introduction and definition of the verb contextual frames used to guide the analysis.

'Context' is the key word. In the process of analysis the semantic routines examine the syntactic environment of the main clause verb, the environment triggering the relevant preference test. The outcome of this test determines the correct verb sense selection and sets up the backbone of the dependency structure. The generator employs similar tactics, but in reverse: the dependency structure is regarded as specifying a certain context (with regard to the meaning it conveys), and it is the verb (a distinct verb sense, to be more precise) which carries large amount of that into the generated sentence. Once the verb has been selected to fit to the dependency structure, we can be sure that (at least some of) the 'message' will come through, and more than that, that this (target) verb will, in its own right, specify a definite and distinct syntactic environment, required by the target language grammar rules. All the generator has

to do from this moment on is to extract relevant bits and pieces from the dependency structure with which to fill in the empty slots around the selected verb, thus giving weight and content to the context and satisfying the environment requirements. It is for these two reasons: context definition, plus environment specification, (collectively referred to as the "information content" of the verb [Goldman75]), that its selection is of primary importance for the operation of the generator.

Let us go back to the examples in 5.2. In the case of (*) suppose the generator came up with "inquire" as the proper synonym. It certainly is a good choice, as far as the meaning of "ask" in (*) is concerned; introduces the correct context, and immediately sets up a certain pattern associated with it: "somebody inquires about something from somebody else", or

```
(*)   *hum
          INQUIRE
          ABOUT  *ent
          FROM   *hum .
```

Similarly, the other example will yield something of the kind:

```
(**)  *hum
          REQUEST
          *ent
          FROM *hum   .
```

It does not require a long explanation why this environmental pattern is referred to as 'syntactico-semantic'. Obviously it specifies the syntactic ordering of constituents which must satisfy certain semantic criteria and are inseparably part of the meaning and context introduced by the central verb.

At this point I could go one step further and after rewriting (*) as

```
(*')  @agent
          INQUIRE
          ABOUT @subj-matter
          FROM @recipient ,
```

observe that such a specification allows the generator to immediately identify the relevant subportions of the dependency structure, and to access them in order to extract the wanted slot-fillers. This, however, raises certain questions:

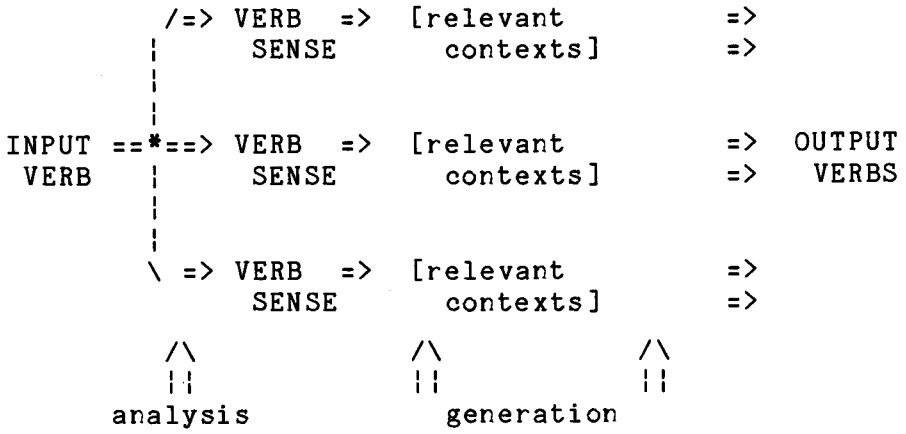
(1) How to make the information about the syntactico-semantic environment available to the generator?

(2) Which mode of formal specification - (*) or (*') - is more convenient?

(3) How can we be sure that things like '@agent' or '@subj-matter' will be present in the dependency structure?

However, this is looking a bit ahead, and in order to provide answers to all these questions, I shall initially consider the immediately important problem: how to select the verb synonym to be used in the generated sentence; and how to utilise information in the dependency structure in the process of doing so?

In an abstract and approximate way, a cross-section of the whole system along the line 'verb treatment' will produce something like fig.1. Each verb used in the input is subject to a 'mushroom explosion' of synonym equivalents for each of its senses and it is the generator's job to reduce the number of possible output verb synonyms for the sense selected by the analyser to just one.




```
ASK
  ask1 ((man subj) ((*ani obje) ask))
     QUESTION, INQUIRE
  ask2 ((man subj) (((please feel) goal)
          ((((notsame man) subj)
                ((act obje) do)) ask))
     BEG, WANT
  ask3 ((man subj) ((*ent obje) want))
     REQUEST
```

fig.1

The issues to be discussed here are concerned mainly with

(*) how difficult (or, alternatively how trivial) is the selection job;

(*) what information must be used to do it properly; and

(*) how is it to be used.

This brings us to the most essential question - the source of the actual target (English) verb. The decision to keep shorthand markers in the dependency structure (3.11) was based on the desire to keep this particular operation as easy as possible; the reader is referred to chapter 3 for more details.

5.3.2. Mechanism

I shall now concentrate on the actual mechanism of the verb selection. Since the dependency structure itself already points to, for example, "ask1", all that is needed to set the generator on the right track is to decide which of the possible synonyms for the particular sense of "ask", coded by "ask1", is appropriate for the specific sentence being analysed: "inquire" or "question" in this case. Although to a large extent these synonyms are mutually interchangeable, some discrimination, and therefore, critical judgement, is called for.

Let us examine the reasons for this. Going back to the examples from (5.3.1), it is possible to notice certain regularities in the pattern introduced by the contextual environment of the suggested synonyms. For example, "question" requires a human as a direct object; and this human is actually the recipient in the input sentence. If no human recipient has been found in the text, or, rather, the corresponding dependency structure, then "inquire" must be used. On the other hand, if it is not known what the conceptual object of "ask" is (in other words - what is the thing about which all the asking is being done), the use of "question" is preferred.

Similarly "ask2" (see the definition: fig.1) introduces another two synonyms: "beg" and "want", and the choice between them depends largely on considerations of the same kind: what is the agent asking for - an abstract entity (favour, advice, etc.), or a specific action - to do something?; who is the beneficiary of the action 'ask'?, etc.

It is obvious that the decision process must be guided by a careful examination of the dependency structure in the light of

a) what sort of semantic constituents are present, and
b) what conditions regarding the meaning (semantic content) of these constituents are fulfilled, or, alternatively, violated.

Point a) means examining the contextual framework of the dependency structure; point b) means analysing the contextual content. Again, I want to emphasise that (see 3.11) it would be possible to analyse the dependency structure without any explicit knowledge of "ask1" being the intended sense of "ask", and still arrive at the same conclusion - because even with the deletion of the "ask1" marker in the structure, the contextual content, and hence the overall meaning, is unique. But, as already pointed out (3.11), "ask1" in this case serves simply as a shortcut entry point to a predetermined part of some linguistic knowledge database, which knows all about the possible syntactico-semantic environments of a verb-sense ("ask1"), and provides the information necessary to drive the decision mechanism of the verb selection routines.

Going back (to the beginning of this section) to the analysis of

the contextual constraints imposed by the candidate synonyms, it is possible to define the process of verb selection as a process of generalised matching of the component slot-fillers of the dependency structure against the contextual requirements of the list of candidates, using some sort of rule as 'densest match wins' to select the most appropriate one.

It must be emphasised, however, that the generator does not seek to map a dependency structure onto a specific matching verb, but that it tries to find a synonym contextually compatible with the verb under whose meaning the dependency structure has been derived. This is one of the substantial differences between my design policy and BABEL's. It also allows for the limited, not uniquely defined, choice of verbs for the output. For example, if all the conditions regarding "inquire" and "question" (as specified above) are satisfied, given

John asked Mary about the book,

either synonym seems appropriate then:

John questioned Mary about the book,

John inquired about the book from Mary.

There is nothing wrong with having a range of alternative output possibilities, as long as the fact is recognised, and some selection in such cases is actually made.

The foregoing more or less solves the problem of verb-synonyms selection. I have not devoted so much attention to the choice of the other words to be used in the target sentence. The reason for this is mainly in the ultimate objective of the generator, namely demonstrating the correctness of the (word) disambiguation activities of the analyser. Thus, nouns in the input often generate the same nouns in the output: "John" => "John", "book" => "book", simply because they happen not to have been coded for more than one sense in the test lexicon. If the noun itself is ambiguous, a choice is made: for example "crook" may turn into "long thing" or "artful dodger", depending on the content of the dependency structure. Prepositions are generated according to the syntactic requirements of the specific context, and have absolutely nothing in common with any prepositions used in the source text, which get completely lost in the analysis. Predicate adjectives are treated much the same as verbs, following the policy underlying the parser: their analysis is carried out by adjective frames (see 3.9). Adverbs probably require more special attention, but at the present state of the program I chose to give them a rather superficial treatment.

Given this overview of the processes involved in choosing the words to be used in producing a sentence to express a dependency structure; we can now concentrate on the next step in the generation process.

5.4. Definition Of The Target Language Syntactico-Semantic Relationships.

5.4.1. How Will This Help?

Effectively the problem is: given a set of words chosen during the generator's first phase: ["inquire", "John", "Mary", "book"], how to produce the syntactically correct English sentence "John inquired about the book from Mary". The words themselves mean nothing to the program, as a NL system is not supposed to rely on words, but on underlying meanings. Suppose then that we possess further knowledge to the effect that in the case of this particular sentence John is the agent, Mary is the recipient of the action ("inquire"), and the book is the object being inquired about (i.e. the subject-matter). The semantic roles of the words, thus defined in the dependency structure, supply the information for a definite syntactic ordering of the output words. Let us imagine that somehow, certain syntactic knowledge is made available to the program; and that it knows, in particular, that

a) normally the agent appears before the verb and plays the role of the subject in the sentence,

b) the recipient is either the direct object, or a part of a prepositional phrase; and comes after the verb in the sentence,

c) this specific verb ("inquire") requires a subject-matter, which is the object of the sentence, immediately following the main verb; and is preceded by the preposition "about",

d) the recipient for this verb is optional; hence it is the last to appear in the sentence; and if it does, it is preceded by "from".

Guided by all this information, the generator will easily produce the intended sentence (see above). The problem is reduced to organising and utilising this knowledge.

5.4.2. What Is An Environment Network?

The example above clearly shows that there are two types of information to be considered: the semantic roles of the words, and the narrowly syntactic character or function they have in the sentence, which, among other things, determine their ordering.

There is clearly some relationship between those two distinct types of information. In most cases it is possible to make an educated guess as to what the surface syntactic role played by a semantic constituent, and its position in the sentence, will be: this

was suggested in the example discussed above. Very often the agent is the syntactic subject; the recipient of an action is either a direct object, or is introduced by a preposition which in most cases is "to" (or "for"); a 'location' denoting constituent is keyed by a preposition from a finite set of prepositions ("in", "at", "on", ...) and is determined by the syntactic requirements of the context defined by the main verb:

John admitted Mary to the house, but

John let Mary into the house.

A 'manner' constituent will be expressed either as a "with" prepositional phrase: "with enthusiasm", as an adverb: "enthusiastically", or as an "-ing" phrase: "by doing something", and will follow the direct syntactic object (if it is present). A 'mobject' (mental object) or 'goal' labelled node in the dependency structure is readily expressed as an embedded clause: a "to-" or "that-" complement to the main verb:

John admitted to Bill that he loves Mary,

John made a gun to kill Mary.

The knowledge of these and other related facts, coupled with the specific syntactic framework defined by the main verb around which the sentence generation is organised, implies that in order to produce

John begged Bill to hit Jack with the crook,

a data structure of the following type may help:

```
        (agent . N2)
        (recipient . N3)
        (mobject . N4)
N2:     (noun . "John")
N3:     (noun . "Bill")
N4:     (action . "hit")
        (agent . N3)
        (recipient . N5)
        (instrument . N6)
N5:     (noun . "Jack")
N6:     (prep-phrase . N7)
N7:     (preposition . "with")
        (noun . "long thing")
```

The interpretation of the structure is straightforward; so is its general format. This is a collection of nodes (concepts) each of which holds a set of relation-concept pairs. The idea is to represent in compact form certain relations (primarily semantic ones) - deep case relations and other relational meanings - that conjoin concepts. This makes it possible to map the relation part of the pair onto syntactic relations and other relational words in the surface string, treating the concept part of the pair - another structure node, a lexical entry pointer, or one of a set of terminal grammar elements - as the value of this relation. All relevant information is extracted

from the structure to define, in accordance with the target language grammar rules, the surface string reflecting the relational ordering of the concept in the data structure, thus conveying its meaning in English (or any other particular language).

To make things more clear, the example sentence already discussed:

John inquired about the book from Mary

will effectively be generated from the following data structure:

```
(*) N1: (agent . N2)
        (tense . past)
        (form . NIL)
        (lexverb . "inquire")
        (subj-matter . N5)
        (recipient . N8)
    N2: (lexnoun . N3)
    N3: (English . N4)
    N4: ("John")
    N5: (prep . "about")
        (lexnoun . N6)
    N6: (English . N7)
    N7: ("book")
    N8: (prep . "from")
        (lexnoun . N9)
    N9: (English . N10)
   N10:("Mary")
```

There are several important points about this structure:

1) it makes the actual output generation of the target sentence a relatively easy and well-defined task. This is because the required surface words are explicitly available in the structure, accompanied by specific syntactic information (lexical category, tense, form, voice, etc.) facilitating the correct construction of English expressions. Furthermore, a closer look at (*) shows that the order of the pairs in each node defines the left-to-right order of components of noun and verb phrases and embedded clauses in the generated language string. Thus, provided the structure has been ordered according to some criteria (to be discussed later), a simple sequential, recursive scan will suffice to determine the order in which the generator will output the words of the sentence.

2) It can be observed that (at least at the clause node level - N1 for example) the data structure somewhat resembles the dependency structure presented as the input to the generator, suggesting that its construction might not be a difficult process. More important, the process is well defined.

3) The structures thus produced are closely related to Fillmore's proposals for case grammar (this is why they are related to the dependency structures as well). More important, they are conceptually identical with Simmons' semantic networks [Simmons73] and [Simmons72]. It has already been demonstrated (by Simmons and by Goldman) that such a network can be handed over to some sort of a grammar control component which will transform the network into a

semantically equivalent linear string of target language (English) words. The generated sequence will carry much of the meaning of the dependency structure, and hence, the original sentence.

4) The pairs-into-node organisation of the structure and its components provides a convenient way of accessing and manipulating the phrase to be generated as a whole, or some of its parts only. This is very important, because it allows the generation to treat a phrase as a unit; to reshuffle its parts in order to achieve a more natural output phrasing, while maintaining its relative position in the sentence; or to do some preprocessing of the component nodes (before the actual output of the sentence) in accordance with the target language grammar rules (for example the deletion of nominals in embedded relative clauses, pronoun substitution, suppressing the subject in embedded "to-" complements, etc.: see 5.5 for outline of the functions to output a sentence).

I have called this data structure, mediating the process of sentence generation, an environment network. It is the conceptual counterpart to what Goldman defines as a 'syntax net' in the gross structure of BABEL, but it contains not only syntactic information, but semantic information, relevant to the contextual environment of the main verb. Hence the proposed name.

The general strategy followed in generating is therefore:

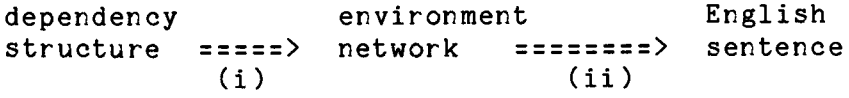


fig.2

The two processes (i) and (ii) can be regarded as transformations mapping one form onto another, preserving the meaning of the data structure they manipulate, and thus carrying the information initially present in the dependency structure through to its expression in grammatically correct English.

5.4.3. Construction Of The Environment Network.

The fact that the environment network resembles the dependency structure supplies some clues towards its production. At this point the reader is advised to go back and review all the arguments presented in defence of the statement that the generation must start with the choice of the main verb of the clause. As already emphasised, 'context' is the basic notion underlying the design of the generator, and in (5.3.1) it was pointed out that the main verb, even when regarded in isolation, introduces a certain context, setting up a definite syntactico-semantic pattern associated with the verb. Or, as Goldman says, "once a verb has been chosen, an entire semantic framework becomes known": "inquire", for example, suggests

(*) *hum1

```
INQUIRE
ABOUT *ent
FROM *hum2
```

(see 5.3.1). Going back to the first example (*) in 5.2, it was noted in 5.3 that given the dependency structure built round "ask1", the verb choice procedure will select "inquire" as the synonym round which the whole generation process will be organised. This selection process is strongly influenced by the presence in the dependency structure of such constituents as human recipient and entity-type subject-matter. In which case it is clear where we should look for the entities which will fill up the (*) framework and give content to the context. There is no doubt that these constituents will be available in the dependency structure, because, after all, it is mainly because of them that this particular verb has been chosen. To refer to Goldman again, "once a verb has been chosen, the form of the environment network can be predicted"; and furthermore, mappings between parts of the network and subportions of the dependency structure can be predicted as well. "The creation of the remainder of the net is a very strongly guided process, not a large search".

This is quite natural, and could be expected: in the analysis, the syntactic environment of the verb is extensively used to pick up its meaning and decide on what to do with the dependent constituents and what representation to build; the generator will employ the reverse process - examine the context and select the main verb, which will immediately set up the syntactic environment.

Three distinct types of information must be made explicit in the environment network:

1. What are the surface words that will be used in the sentence, and how to build syntactically coherent units out of them - this is the information contained in a node of the environment network (and its descendants):

```
N5: (prep . "about")
    (lexnoun . N6)
N6: (English . N7)
N7: ("book")
```

The syntactic equivalent to this network fragment generated will be "about the book".

2. What is the semantic role played by the constituent thus identified:

```
N1: ........
    (subj-matter . N5)
    ........
```

3. What is the position of this constituent in the overall phrase structure - the subject-matter in the sentence, or the noun in the noun phrase. In other words, what is the syntactically valid left-to-right ordering of the phrase components:

```
N1: (agent . N2)
```

```
.....
(lexverb . "inquire")
(subj-matter . N5)
(recipient . N8)
```

Also, there is additional, purely syntactic, information contained in some nodes (tense: past, form: NIL, English: $Node$, etc.) which I shall ignore until later.

The construction of the network begins with picking up the verb - "inquire" in this case. Examination of its contextual environment tells the program that it requires entities to fill up the 'agent', 'recipient', and 'subj-matter' slots. The corresponding constituent nodes are extracted from the dependency structure and the surface language (English) words to express them are chosen: "John", "Mary", "book", respectively. The syntactic framework is consulted again, and the generator realises that the semantic relationship 'action' <-> 'recipient', and 'action' <-> 'subj-matter' (for this particular verb) are realised syntactically by means of the prepositions "from" and "about". This allows the construction of nodes N5 and N8 (see the environment network in 5.4.2). The program also knows that 'agent' in active sentences does not require a preposition, thus N2 is constructed. The construction of nodes is a process which puts together all sorts of factual, linguistic, and syntactic information and incorporates this into the node or its dependents. This may involve such activities as deciding on determiners or pronouns, attaching adjectives or modifiers, or recursively building another environment network corresponding to an embedded clause (relative or complement, for example), a possessive structure, etc.

A point that must be made here is that there appears to be a certain inconsistency in the type of relations specified in the environment network: these can be semantic or syntactic. Intuitively one feels that a structure with uniform content might be better. However, let us consider the two possible situations:

(A.) Purely semantic network. This will in fact be inadequate, since sentence generation (unlike sentence representation) just cannot be done without explicit and detailed knowledge of syntactic features and relationships.

(B.) Purely syntactic network. This would certainly be helpful, and could be provided. However, it would require that a lot of information about any syntactic equivalent of a semantic component be put explicitly into the environment framework of the verb, and that the mapping transformation from semantic relation to syntactic function (which has to be done anyway), will have to be shifted in time, and become conceptually a part of process (i) in fig.2, and not of (ii) as I propose to have it. The structure would be uniform, but the flow of processing would be broken (syntactic manipulations would have to be done in the middle of processing of the contextual information). And the contextual environment itself, introduced by the verb, is better expressed in terms of semantic units, than with syntactic labels only. Apart from all this, the environment network, as it is now, bears much closer resemblance to the dependency structure, making its construction an easier and more straighforward task. As I pointed

out earlier, the transformation from semantic relation to syntactic function will have to be performed during the actual output of the generated sentence, but this generation is then accomplished by associating with each semantic relation, the transformation which forms it and its concepts into a syntactic constituent of a sentence.

Thus it is clear that the heterogenous nature of the environment network as proposed here need not deter us, and indeed seems more satisfactory from the point of view of consistent operation of the generator.

All that is left now is to specify the ordering of the constituents. This is quite important since natural text, unlike a generalised semantic structure (net), is a linear string of words. Things are made easy, however, by the fact that the rules of ordering are specified not in terms of individual words, but of coherent entities - syntactic units, complete with their logical position and (semantic) function in the sentence. There exists a pool of general linguistic knowledge, combining rules about the order in which facts are arranged in the conceptual unit (sentence, phrase, etc.), and presented to the recipient of any text expressing them. These rules are intuitively known and widely used, all the time. The problem is to present them concisely in an ordered list, which attempts to say, in the most compact form, things like: whatever the subject is (i.e. the semantic agent), it must come before the main verb, which is closely followed by the direct object; this may be followed by an indirect object, or recipient, or location, if any of these are present, and "to-", "that-", or "-ing" complements; the head noun of a noun phrase must come after the modifying adjective, and so on.

The actual list used for the program contains, among others, items like

```
fronted-ing
agent
dummy-agent
poss
quantifier
adj-mod
lexnoun
relative
tense
form
lexverb
object
recipient
pred-adj
location
mobject
.....
```

The list is by no means complete, but is quite rich in information content, thus allowing for the production of relatively dynamic paraphrases (see 5.1). The names of the possible functional labels, as well as their positions in the list (and hence their relative order

in the embedding textual unit) are self-explanatory, and the rationale for organising them in the way suggested is that we are seeking a formal expression of the relationship between a structural unit, its content and message, and its position in the sentence.

A possible problem here is how well the ordering of phrase components could be specified if these components have labels of mixed kinds - semantic and syntactic. An ordering defined in terms of syntactic labels only is well defined [Cater79]. An ordering specified in terms of labels of the two kinds above does not provide an easy answer to the question. Still, bearing in mind the mapping transformation from semantic relation to syntactic function (see above) an ordering could be specified.

This will contain information about the correct sequence in which components of a phrase (any phrase - sentence, noun phrase, prepositional phrase, etc.) must appear. So whenever the node of the environment network representing the concise handle to this phrase is constructed, the pairs in it - or in other words the handles to its component parts - are ordered by matching their relation fields against the order in which these relations must appear in the ultimate surface string representing the phrase in question. Provided the list (see above) is adequate, this guarantees that some technique for outputting words in the order defined by a recursive scan of the nodes in the environment network, will in fact output them in a way which forms a grammatically correct sentence.

This technique is the subject of the next section.

5.5. Outputting The Generated Sentence.

It has already been noted that once the environment network is complete, it should, given the mechanism of its construction, contain the linguistic information necessary and sufficient to produce coherent English. Another point of importance to realise is that the network is a linearly ordered, hierarchically organised, language dependent data structure, whose top node is the handle to the whole clause, organised around the main verb, and whose every node serves the dual purpose of identifying the phrase components and providing easy access to them. In which case, a recursive, left-to-right scan of the network structure, starting with the top node, and going all the way through to the terminal points, will get the surface sentence words into the correct order and in the process of doing so, will assemble the syntactic data required to know precisely how to output them.

This data is available implicitly in the relation field of any node - simply because whatever the (syntactic or semantic) relation, the target language grammar provides a way for representing it as a part of the string. Or, better still, any relation, which can be present in the environment structure, can also be expressed as a linear string of words which conforms to certain syntactic and lexicographic standards. Thus 'prep' means output a preposition which will be given as the value of the 'prep' relation; 'recipient' means generate a nominal structure, organised as a noun-phrase or a prepositional phrase, and based on the nominal phrase node found as the concept dependent on the 'recipient' relation; 'tocompl' or 'thatcompl' mean generate an embedded clause structure realised as a "to-" or "that-" complement. When the generator starts to work on the node, representing, say, 'thatcompl' embedded clause structure, it will (recursively) find a sequence of nodes, embodying the imperative "first output the agent, then the verb, then the direct object, then a 'location' expressing prepositional phrase, and so on". The information used to guide each minor generation process can be

a. all available in the node itself: (prep . "about") means just to output "about".

b. picked up and stored while processing previous nodes or pieces of the network: (lexverb . "inquire") will output, for example, "inquires", depending on the form, tense, agreement required with the agent already output and whose gender has been noted, etc.

c. available implicitly in the knowledge of the type of syntactic unit being generated: (agent .NODE^{i}) means organise the information in the specified node as a noun phrase.

The information assembled during the scan of the network, plus the information made available by the relation type of a specific pair, is thus all that is needed to process the dependent in the pair with regard to doing something sensible to the output string, and eventually getting all of it.

All this, as Cater notices, [Cater79], allows the treatment of a pair part of a node as a function-argument couplet; a node, with its sequence of pairs, as a small subroutine whose ultimate effect is to augment the output string with the phrase identified by that node. Moreover the whole environment network is treated as a complete and independent program to generate the sentence equivalent to the intermediate network, and thus, equivalent in meaning to the original dependency structure.

Thus the environment network serves the dual purpose specifying an intermediate structure constructed and utilised by the generation process in order to represent in compact form all and only the information relevant to the output of English, and of being the data taken by another set of functions as a precise formal specification of the order and type of the operations necessary to produce the target text. In which case process (ii) in fig.2 is dynamically defined by the environment network produced as the result of (i). All that is needed to carry it out is to define a set of functions whose domain will be the words to be used in the output string, plus some additional syntactic and lexical information, and whose effect will be the augmentation of the generated text word by word. Each function will be identified with a certain relation which can be found in a network node, and will perform specific manipulations directed at expressing the desired syntactic binding of the argument of the function to the portions of text already produced or to be produced.

These functions will embody the specific target language syntactic rules and requirements, and thus define a procedural generation grammar.

This is another area in which the generator discussed here differs from the more traditional approach (Goldman, Simmons and Slocum), where the transition from an intermediate data structure ("what to say", represented by Goldman's syntax net, or by my environment network) to the surface text ("how to say it") is achieved by employing a connected body of grammar rules, most often a single ATN generation grammar. My generator, in contrast, relies on grammatical information about the target language surface rules which is distributed among individual specialists. These specialists are powerful enough to express a particular construct in a particular context, but their modularity makes it very easy to define them, as well as giving the designer a free hand in organising the overall production of fluent output text.

The independent definition of many individual output specialists allows for an easy specification and flexible organisation of certain purely surface oriented processes: changes of form in embedded clauses; deletion of the head nominal from an associated relative clause; suppression of the output of certain constituents in certain circumstances; etc. Imagine, for example, that the higher level routines of the generator have decided that a certain constituent of the dependency structure can be expressed as an embedded "-ing" phrase. Under the approach adopted, it is possible to use the general mechanism for constructing a piece of the environment network

corresponding to (the content of) the embedded clause. Then, on the basis of the knowledge that it has to be expressed as an "-ing" phrase, specific information is incorporated in the relevant positions of the data record associated with the network node, to the effect that form has to be changed to progressive, and (on certain occasions) the output of the agent of the embedded clause can be suppressed. Thus the piece of program which is responsible for actually outputting the associated part of the sentence can easily be specified as

```
(DF ing-compl *node
   (CHANGE-VALUE-OF 'form *node 'progressive)
   (CHANGE-FUNCTION-OF 'agent *node 'dummy-agent)
   (TRAVERSE-CLAUSE-NODE *node))
```

The actual details of LISP programming need not concern us here; it is sufficient to say that *node points to the subnetwork corresponding to the embedded clause and TRAVERSE-CLAUSE-NODE is a generalised routine responsible for the output of the clause associated with *node, which clearly, among other things, is capable of correctly interpreting and using information like "perform morphology on the verb", and "do not output the subject".

If, under different circumstances, it becomes necessary to change the target language: i.e. switch from paraphrase to translation - all that is needed to adapt this part of the generator to the new environment, would be a revision of the order of phrase components (see 5.4.3) and redefinition of the set of environment network functions reflecting grammar relations.

A review of the format of the environment network nodes shows that these functions can have three distinct types of arguments:

1. another structure node:
   (agent . N2),
2. a lexical entry pointer:
   (lexverb . "inquire"),
3. a terminal grammar element:
   (tense . present).

This correspondingly defines three distinct types of functions. There are those that transfer the processing to the node that is in argument position. This specifies a 'subroutine call' after which the control is passed to whatever piece of program is specified to output, say, an 'agent'. After the constituent in 'agent' position has been generated (which might, in its own right, involve other 'subroutine calls'), the next function on the current level is activated (i.e. the next piece of the current phrase is dealt with). "Next ... on the current level" means left-to-right recursive scan.

Then there are those functions that add words directly to the generated sentence: output a preposition or an adjective, or output a verb (after performing the necessary morphological operations). These are the functions that have visible effect, but they rely heavily on the third group of functions.

These are the 'behind the scenes' coordinators of the overall process. Their major concern is setting up global variables (useful information) used to pass important data to other functions on the same 'subroutine level' - i.e. those concerned with processing of the same constituent. These are responsible for the setting of the tense and form, of the number of the head noun and the gender of the noun phrase, as well as its case and other important attributes (*human, *place, etc.)

All three sorts of functions are heavily interdependent, and provide the basis of a procedural generator, whose main strength lies in the fact that it is highly modularised in design and operation, and easy to reorganise when translation to another language is required: all that will have to be done is basically rewrite the network functions to accommodate the new grammar rules, and change some of the data files.

Apart from the set of relation functions just discussed, it will be possible to confine all resulting changes to changes in data files:

* the information used to trigger verb/ word selection,
* the actual target language words i.e. the generator dictionary,
* the contextual environment specified by the verb,
* the phrase components ordering list.

5.6. The Structure (And Organisation) Of The Generator.

As already emphasised, a major principle during the development of this program (and indeed, the whole system) has been keeping the data separate from the program. On the one hand this is good programming practice: it is then easier to devote independent effort to the development of the program or to that of the data, while hopefully maintaining an efficient interface between the different processes involved. More important, such an approach makes it easier to see what sort of knowledge is required to perform a certain task, and how best to organise it and present it to the corresponding routine.

Much more important, however, and more relevant from the point of view of the objectives of this particular NL system, is the desire to be able to use the same sort of program mechanisms to produce 'paraphrases' in different target languages. Then the linguistic data needed has to be changed, but the mechanism, in the optimal case, will not be changed at all, or at any rate, not much. Further, having a (general) interpreter separate from the data file(s), which are themselves under no constraint except for being written in a suitable notation, and specifically are independent of the control structure and thus can be changed without overhead, is one of the most attractive features of the approach adopted.

This section traces the processes which underlie the generation of a sentence, thus revealing the structure of the generator program, and simultaneously pointing out what sort of specialised linguistic knowledge is required, and made available, during the various stages of the conversion of the dependency structure into an (equivalent) English sentence. As specified in 5.2, three main steps are distinguished:

1. Selection of the main verb, and the rest of the target language words,
2. Definition of the syntactico-semantic relationships,
3. Actual output of the generated sentence.

The following pattern can be observed:

(*) main verb selection:
requires detailed knowledge of the contextual requirements of the separate verb senses that the program knows about. This is the mechanism of 'fanning-out' (see fig.1), which will effectively decide which synonym of "ask1" to use: "inquire", "question", "examine", "review", "scrutinise", etc. (the synonyms are taken from Roget's Thesaurus).

(*) environment network production:
this makes use of two distinct types of data:
a.) general linguistic knowledge - examples of this were given in 5.4.1: semantic 'agent' is the syntactic subject; semantic

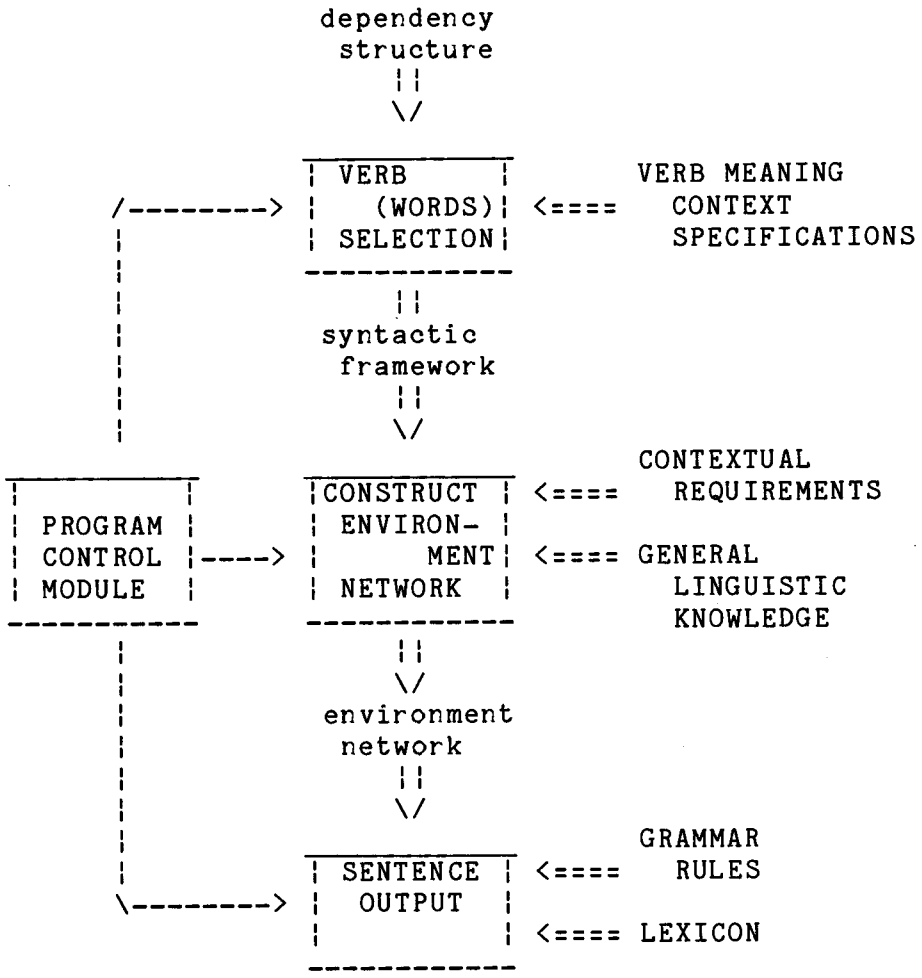
'recipient' is either a direct object, or a "to" prepositional phrase; the verb comes after the agent and before the prepositional phrase specifying location; a 'mobject' (mental object) is most often expressed as a "to" or "that" complement phrase; etc.

b.) specification of the syntactic framework of the target main verb, which will have to be consulted in order to resolve potential syntactic ambiguities (in the output: is this going to be a direct object or a prepositional phrase), and thus guarantee that the surface syntactic rules and the contextual restrictions are not violated.

(*) sentence output:

this phase makes extensive use of the target language lexicon and grammar rules (agreement, morphology, pronoun substitution, etc.), and makes sure that the generated sentence is (at least) a syntactically well-formed string of words.

In the light of these data requirements, the use of the data as outlined above, and the processes which access and utilise it (see also fig.2), the general structure of the generator takes the shape of fig.3. For more specific details of the program implementation the reader is referred to Appendix (i).



chapter 6.
Review, Perspectives and Conclusions.

6.1. System Review.

The discussions in the preceding chapters clearly attempt to provide answers to more than one question, quite different conceptually, but related within the overall objective of this work. Some of the points discussed were general principles underlying the design of the system. Some of them were the justifications for these principles and hence for the approach adopted. On this basis some specific mechanisms were proposed as devices for applying the principles. These mechanisms were developed within an overall framework for automated computer-based text processing, to obtain a natural language processing system capable of the task of translation/ paraphrase.

There are thus several aspects to this study. There is the general problem of organising any natural language processing system. Then there is the problem of identifying an adequate approach to the task of text analysis. The focus of the investigation has been on the place of semantics in a computer-oriented framework for language processing, and particular interest has been paid to the issues involved in the judgemental aspect of semantics: the resolution of linguistic (lexical and structural) ambiguities is the primary objective of the system, followed by the construction of an unambiguous meaning representation. The object of the program is therefore semantic analysis, and the claim made by the work is that the gap between surface text and internal semantic representation should (and, indeed can only) be bridged by the joint efforts of a syntactic processor and semantic judgement specialist routine(s). The method is by combining the syntactic recognition of a constituent with semantic evaluation of its coherence as a unit and its compatibility with other units, based on global analysis of the contextual environment. This procedure is applied recursively, in a framework of preference semantic theory incorporated in an ATN syntactic processor. The underlying strategy in the semantic analysis is that of semantic pattern matching as opposed to semantic expectations, thus showing a bias towards passive rather than active semantic parsing. In order to support this strategy, the notions of contextual verb frames and preplate stacks have been developed, in addition to the bare templates and semantic formulas taken over from Wilks. These are static data structures embodying some of the semantic knowledge of the parser. The rest of this knowledge is distributed procedurally in the program implementation: various techniques and principles for disambiguation are employed. These control the application of semantic tests which are guided both by the static knowledge of the analyser and by the semantic content of the component constituents. The tests evaluate contextual requirements within the constituent being analysed and construct a meaning representation for the constituent. The effect is the dynamic construction of only those semantic structures which are valid within the current context. The evaluation of the final dependency structure corresponding to the meaning of the input

sentence is accomplished by subjecting it to an inverse transformation, aimed at re-expressing its meaning in English. Thus automatic paraphrasing is effected. This is carried out by a generation program which is carefully designed to segment the process into two largely independent phases (environment network construction and sentence output) thus making it possible to utilise the same program control structure and most of the code for translation instead of paraphrase.

6.2. My System In Relation To Others.

It has already been noted that the system described here is a hybrid descendant of Woods' and Wilks' work in the field of text processing. Clearly, my attempt has been to circumvent certain disadvantages in these two projects and to amalgamate their respective approaches to language analysis within a methodologically sound framework serving as the basis for a parser which is flexible and capable of accepting rather complex syntactic structures; sophisticated and intelligent to treat various types of ambiguous text properly; open-ended, i.e. easy to extend both to handle new types of constructs and to accommodate further developments and/or changes needed to gear it towards performing other tasks in the domain of natural language processing; and efficient in terms of performance features and figures.

There are naturally similarities between my approach and those of Woods and Wilks. However, Woods has contributed only on the level of supplying the mechanism for parsing - the concept of an ATN grammar and interpreter; his treatment of language analysis as a two stage process (syntactic recognition followed by semantic interpretation) going through explicit intermediate syntactic (TG based) structures, is considered quite inadequate and inappropriate for dealing with real life texts, and for solving the problems with which this project is concerned. On the other hand, Wilks' influence on this project is much more significant. Several points of methodological interest are extracted from his system: the general strategy of semantic pattern matching, the representational scheme based on semantic primitives and allowing for a flexible system of semantic categorisation and representation of a word-sense meaning, the preference semantics principle for choosing between alternative interpretations. All these underlie in a very important way the design of my parser.

Substantial differences between my system and its predecessors are easily spotted. Woods' strategy for text analysis has been rejected altogether, so has been his theory of procedural semantics; the ATN grammar used by my system is similar to, but not the same as the LUNAR grammar, the difference being not only in the range and types of constructs handled, but in the parsing strategy adopted as a necessary prerequisite for efficient analysis of postmodifying constituents; the ATN interpreter has been developed and implemented with this strategy in mind, and is designed to provide an efficient interface between the syntactic recognition and the semantic judgement routines. This ATN-guided syntactic preprocessing of the text replaces Wilks' initial fragmentation. As a result, his bare templates do not now have to account, among other things, for surface patterns, and can be matched more freely onto the semantic content of the constituent slot-fillers. The logical extension of this idea is the notion of a verb contextual frame. The strategy of generalised, global, context-sensitive semantic pattern matching allows me to eliminate Wilks' consecutive

passes over the same piece of text. Indeed, great care has been taken to avoid both unwanted syntactic backtracking and unnecessary semantic parallelism in the processing, by designing versatile semantic specialists and establishing the points in the analysis process where they can be invoked and be of greatest use. The various causes for, and types of, lexical and structural ambiguity have been studied and specific mechanisms for dealing with each have been designed and implemented, all of which lie within the general strategy adopted.

Throughout this thesis, references have been made to related work in the field of natural language processing. Usually the reference was made in the context of discussing a specific point. I shall now, therefore, outline very briefly what the position of my system as a whole is in relation to other recent and current projects.

Considering the original ATN model, some effort has been invested by Woods in generalising the model and improving the parsing strategies (see bidirectional "island parsing" which can start at any point of the sentence [Woods76]). This is a theoretical development of the model which has been to a large extent motivated by the desire to develop an ATN parser as a front end to a speech understanding system. As far as semantic processing with ATNs is concerned, apart from Simmons' semantic networks (where "semantic" is a bit of a misnomer, as was already discussed in chapter 2), the most substantial effort is, to my knowledge, that of Ritchie [Ritchie77]. His aim is to develop a model of language "which allows linguistic description of English in processing terms". The ATN formalism thus comes in very handy as a notation, and the ATN grammar as a concept on which an analyser is based. Ritchie is however more interested in the details of the processing mechanisms, than in general questions of semantic processing, or the relation between syntax and semantics in a natural language analysis system. He has implemented a grammar rather than a parser whose purpose is to clarify the ideas involved in his model. Another possible extension of the ATN model application has been proposed by Steedman (Mark Steedman at a lecture at Essex University, 1977), who suggested that instead of being represented by one network, the grammar could be decentralised by associating small ATNs with the definition of each verb to drive the analysis process. The problem with this approach is exactly that associated with Riesbeck's approach to active parsing based on semantic expectations/ predictions, which has been discussed at length in the preceding chapters.

None of these approaches offers a definitive answer to the question: how to conduct the analysis process in a highly ambiguous and non-deterministic environment. The judgemental aspect of semantics, as noted in chapter 0, tends to be ignored. Even Schank, who discusses issues like "syntactic" and "semantic" ambiguity in the context of conceptual analysis [Schank73], does not offer a general framework within which the problem of their resolution could be solved. It is not clear whether the CD parser can actually handle wide range of instances of multiple choice; on the other hand, it is clear that Riesbeck's treatment of "sense vs. meaning" (see 2.2), as well as Simmons' extreme view on lexical ambiguity (2.1) are adopted more in order to circumvent the problem than to solve it in a general

way.

Apart from Wilks, who openly acknowledges the problem of word-sense ambiguity, the only project which makes an explicit attempt to solve it, is that of Hayes [Hayes77]. His work is concerned with defining a process of finding the correct interpretation for ambiguous words as objects (i.e. ambiguous nouns), based on associations triggered within the environment of data structures conceptually similar to Minsky's frames [Minsky75] (not to be confused with my contextual verb frames). The work has a definite merit of presenting a unified systematic approach to the problem. It is not very clear, however, exactly how the approach can be extended to deal with the other parts of speech: ambiguous verbs, adjectives, etc. Little attention is devoted to studying the role of syntax in the disambiguation process. Incidentally, Hayes uses an ATN grammar to parse his example sentences, but since the problem of structural ambiguity is ignored in his framework, the grammar is very simple (for example the noun phrase network forbids postnominal modification) and in effect the parsing can be carried out deterministically. This is the other problem with Hayes' work - it does not offer any clues for incorporating his technique for lexical disambiguation in a generalised, versatile parser intended to deal with more complex texts; nor does it suggest how a complementary approach aimed at solving the problem of structural multiple choice could be incorporated in his framework.

As far as the latter problem is concerned, there is one project which seems to offer a solution to it - this is Marcus' deterministic parser, which applies "wait-and-see" strategies in order to avoid making wrong decisions and then having to back up to the decision points. The idea is to exploit enough grammatical knowledge in parsing to ensure that a constituent is used only when and where it makes sense in the current context, thus avoiding blind syntactic nondeterminism. Clearly this is very close to the principle adopted in this work, and referred to as "semantic wait-and-see": decisions about the structure(s) to be built are delayed until enough contextual information is available. However, I do not go as far as Marcus in saying that a deterministic parser is a feasible proposition; in fact, as it has already been discussed, when one faces the problem of structural multiple choice, one faces the fact of nondeterminism as well, especially if one is concerned (primarily) with syntactic recognition, as Marcus' theory is. Hence the label "semantic" attached to my approach, signifying that the structure building process in my system is guided by the semantic content of all of the constituents available, rather than by syntactic features of the top few constituents in the input buffer. Again, the problems with Marcus' system are that his grammar seems to cover a not very wide range of language constructs, and it is not very clear how well his matching rules can be extended while maintaining his current level of system performance; in addition, he offers no treatment of lexically ambiguous items.

A not insignificant number of ideas, principles and mechanisms developed for natural language processing have lately been utilised in

developing front ends for data base systems [Harris77], [Hendrix78], [Mylopoulos76], [Waltz78]. This is one of the few situations at present in which a NL system has to stand up to real life texts, thus demonstrating how good and useful a particular approach is; however, such systems have to be judged carefully. Firstly, in these systems the natural language front end is tied to the very restricted universe of discourse represented by the data base, so it is not always clear how easy it would be to generalise any particular approach or device, or exactly how a particular framework could be extended for less specific situations. In addition, simplifying assumptions justified by the limited universe with which the data base deals, are quite often made, which allow the designer of the front end system to cut corners, but which are totally unrealistic, and hence unacceptable, for a general natural language analyser.

As an example, consider the PLANES system (Programming LANguage-based Enquiry System) which manages a database of aircraft maintenance and flight data [Waltz78]. The analysis of a natural language query involves syntactic recognition of phrases with a specific interpretation relative to the data base. The phrase patterns are stored and represented as ATN subnetworks - one for each different semantic object in the PLANES world. The phrase matching is followed by semantic pattern matching at query (sentence) level using concept case frames - patterns of questions understood by the system. At first sight these can be considered to be similar to my contextual verb frames, but this is so only on a very general conceptual level. While my frames are general devices aimed primarily at context evaluation, disambiguation and structure building, Waltz's concept case frames are data-base specific, and, along with the patterns, embody a semantic grammar for the restricted planes world: here no real problems of matching a set of constituent concepts are encountered, particularly with the simplifying assumptions Waltz makes, such as the lack of lexical ambiguity, and simplicity of the question forms. In my system the question of what the case links between the verb and its argument concepts are is asked only after a frame match has selected the verb meaning; in PLANES, in contrast, the aim is to fill in a predetermined set of slots.

Similar remarks can be made of the process of query analysis in TORUS [Mylopoulos76], where again noun (phrase) nodes are tested for a fit in a case frame for the verb. Thus no real parallels can be drawn between my system and current front end natural language analysers for data bases.

6.3. The System In Perspective.

6.3.1. Current State.

The system has been tested on a non-trivial set of syntactic constructs, the development being essentially a three step process: making sure that a new construct can be recognised by the grammar; making sure that semantic analysis can be incorporated in the process, or that higher level semantic routines can be made aware of the construct; and finally making sure that the generator can handle the intermediate representation of the construct. Thus the grammar grew gradually, and so did the program. Of course, one of the central questions throughout this research was how easy it is to extend an existing system in order to make it hospitable to a new construct.

Necessarily, this involved a study of how a proposed extension would affect the present coverage of language constructs, and how it would interact with the existing grammar framework. In most cases it turned out that the changes or extensions to the grammar that were needed were not many, and the real difficulty was interfacing the context of the new construct with the parsing environment in which the recognition has to be carried out, i.e. in which the construct has to be analysed. What this means is that the setting of the registers local to the parsing environment, as well as the communication of data (registers) between different stages in the processing (i.e. levels of computation) had to be planned carefully. I found that once the basic grammar framework was set up, extensions to it were relatively easy to make. The basic grammar with which this project started was very simple: an NP V NP nucleus with possible prepositional phrases or complements optionally appearing at the end. However, it was very useful in setting up the underlying structure of the analyser and establishing the basic form of the semantic routines.

Once the grammar started growing, and more constructs were added to it, some of the ideas concerning the organisation of the semantic routines, as well as the possible ways of incorporating semantic judgement in the grammar, had to be thought over and restructured; the process without doubt contributed substantially to the clarification of my basic ideas. After the grammar was extended to cover a quite wide range of language constructs, it was possible to make useful generalisations about the organisation of the semantic specialists and the position of semantic judgement in the overall process of sentence analysis. It was also possible to acquire a more structured view of the principles determining the application of the semantic judgement routines. For example it was about a year before the concept of a contextual verb frame, together with its organisation and the

principles for its use, were developed and crystallised, and the necessary mechanisms incorporated in the system implementation in a well-organised way. But it was then only a matter of weeks to extend the system to deal with some "-ing" phrases in subject and object position.

The development of the generator followed similar lines. The questions asked were: can this construct, or rather a dependency structure containing a representation of this construct, be handled by the existing generator framework? If not, what extensions are necessary to do so? Once the general strategy of a two stage generation process going through an intermediate environment network was adopted, and the skeleton structure and organisation of the routines generating nodes for clauses and nominal groups were relatively clear and stable, it was easy to extend the generator apparatus to handle nominal pre- and post-modifiers, relative clauses, "-ing", "to-", "that-" phrases, etc. The process of extending the generator, however, has two aspects: is the mechanism to handle a construct available in the implementation, and does the generator possess enough linguistic knowledge to express a dependency structure it is given? It is reasonable to claim now that as far as versatility in terms of semantic structures that can be expressed, as well as in terms of ways of saying things in English is concerned, the program is quite flexible. This was necessary in order to meet the requirement for "dynamic paraphrases" that was set out in chapter 5. However, since work on the generator started after the analyser had made substantial progress, it was felt that an effort to develop a flexible and versatile output program would be more justified than an effort to extend the generator vocabulary. As a result there is certain disparity between the sets of concepts that the analyser and the generator know about.

The dictionary used by the analyser contains about 400 entries; around 64 of these are verb definitions, and most of them have more than one meaning. The reader can get some idea about the content and organisation of a dictionary entry from Appendix (ii). The dictionary used by the generator is considerably smaller; as a result even though the analyser has been tested on a wide range of input sentences, not all of the resulting dependency structures can, at the time of writing, be expressed back into English. It is for this reason that the example sentences tend to have rather simple subjects and objects: very often it is John who does things, and Bill or Mary that things happen to. It is important to realise that this is so not because of simplicity of the analyser, but because of the currently limited generator vocabulary. The system can handle examples of real life texts, as can be seen from Appendix (iv).

The range and form of syntactic constructs acceptable can be seen from Appendix (iii).

6.3.2. Limitations Of The System.

Every natural language analysis system has limits on the range of linguistic phenomena it can handle; so does mine. These, however, fall into different classes as far as their position in the overall analysis process is concerned. At the lowest level there are the syntactic constructs the system cannot recognise (see Appendix (iii)). Extending the program to accept, for example, existential "there" or comparative and superlative adjective structures is not really important at this stage because it would not contribute to the development of the project as far as its primary objectives are concerned. The system does not analyse questions, but again, it is felt that no major reorganisation of the analysis process would be necessary, and they could easily be accommodated in the existing framework of the SBUILD semantic specialist (the grammar extensions to recognise questions would not be too difficult to formulate).

Then there are phenomena occupying an intermediate position. For example, the system in its present state cannot handle conjunctions. It was already noted (1.1) that these constitute one of the causes of structural ambiguity. Consequently, the system should offer some treatment for them. It is my feeling that it would be possible to design a separate semantic specialist for conjunctions, to be activated whenever necessary, which falls within the general framework of semantic pattern matching applied at well chosen points of the analysis. Unfortunately, the recognition (or more precisely the lack of such) of structures with conjunctions is a deficiency of the basic ATN model, and even though some suggestions to overcome it are made by Woods [Woods73], the problem requires more thought. A generalised treatment will probably require some sort of demon mechanism (see [Winograd72]) which will have to be interfaced to the overall control structure of the ATN interpreter. However, the problem was felt to be too large to tackle along with all the others being studied; a justification for ignoring conjunctions at this stage may be the fact that, as shown in (1.1), the other types of structural and lexical ambiguity provide enough material for setting up a (hopefully) general framework for language analysis.

One level up are problems connected with the more immediate objectives of this system. I do not claim to offer a complete solution to the multiple choice problem. Semantic pattern matching alone is no panacea, and clearly wider world knowledge is required to interpret

At some time of his life every man is a bachelor

properly. Examples like this are too easy to find and deserve more than superficial treatment which they can be given in one thesis subsection. The issues concerning the exact kind of knowledge needed to deal with them, and the principles and mechanisms for its utilisation, have been left out of this discussion; however it needs to be said here that no language analysis system can be complete without offering a solution to them.

A possible partial, if not complete, solution could be to provide the system with an inference mechanism through which alternative, semantically plausible readings would be fed; the general principle of preference semantics would then be applied to the outputs of these inference process(es). The lack of such a mechanism accounts for the program's ignorance when it comes to the problem of pronoun resolution in particular, and the problem of referential ambiguity in general. The reasons why no attempts have been made to deal with these problems were explained in (1.1) and I shall not repeat them here. The only point that should be made here is that in order to provide the higher level deductive component of a future system with all the information it will need, some aspects of language which the system currently ignores, because they are too far away from the general problem of multiple choice, will have to receive more careful treatment: no sophisticated inference module would tolerate a superficial treatment of qualification, tense, complex determiner structures, etc.

6.3.3. Future Developments.

Following the remarks already made in this section, it is clear that the system can be extended in more than one way. An immediate course of action is to "connect" the dictionaries used for analysis and generation, i.e. make the generator know as much as the analyser does (see 6.3.1). Of course this should not preclude the straightforward extension both of the dictionary and of the grammar, thus enlarging the scope of the system as a whole.

Then there are certain issues of theoretical interest in my approach to text analysis. The most important of these has its roots in the nature of the ATN model. It was already shown that a transition network based analysis of sentences proceeds as a sequence of embedded computations. The model formalism developed by Woods allows communication between levels (via the SENDR and LIFTR primitives). The facility is, however, used only for initialising registers (before commencing an embedded computation) or resetting them (before resuming an upper level one). No study has been made on how to use facts known about upper levels (say, which are true within the range of the embedding clause unit) when processing a lower level. At present it would be so easy to trip the system by saying

John claims that his pet elephant can shoot.

On the one hand this is a general theoretical problem encountered under different guises in many areas of AI research. On the other hand it is a practical problem for a language analysis system. Finally, it is connected in an important way to the problems of analysing metaphors at the level of embedded clauses. Currently, the semantic inconsistency of a hypothesised embedded clause unit triggers the rejection mechanism, and backup is initiated following the blocking of the syntactic path currently being followed. This means that a sentence like

Ships proudly ploughing the sea are a magnificent sight

will not be parsed at all. One solution to the problem might be a

different treatment of the notion of "failure" (in the case of meta-category mismatch - see 4.1). A structure might be not rejected altogether, but constructed and strongly 'depreferred' by assigning it a very low 'preference weight' (see 4.5). Such an approach will fit easily into the preference semantics framework, but the cost will be high: greatly increased syntactic nondeterminism (even though it will not be as bad as explicitly following all the syntactic recognition paths through the network grammar.

In connection with this, it would be interesting to see how much of the later ideas of Wilks, and especially "making preferences more active" [Wilks78] could be incorporated in my system.

Clearly all the issues outlined above are important, and require careful consideration.

Next come system developments that have to be carried out before a claim can be made for a relatively 'complete' natural language analysis system. The first of these is a mechanism for pronoun resolution, which as already noted requires some sort of inference mechanism. It is sometimes maintained (e.g. by Minsky and Charniak) that some initial "parsing" can effectively be decoupled from the application of the inference mechanism. However, as Wilks notes, this is not so because "many of the later inferences would actually have to be done already, in order to have achieved the initial parsing" [Wilks76c].

It is reasonable to believe that a mechanism conceptually similar to Wilks' common sense inference rules [Wilks75] could easily be accommodated in the proposed framework: thus the semantic patterns that guide his CSI application procedure are present, though not in the immediately obvious form of an ordered triple, in my dependency structure.

A more 'complete' system will most certainly have to deal with connected texts, rather than isolated sentences - something that my system does not do. This naturally introduces the problems of multi sentence analysis, i.e. computing the semantic consistency and coherence of a structure in relation to larger chunks of knowledge, and, in the context of resolution of word-sense ambiguities, the connected issue of disambiguation by association.

Finally, there is the question of using the system. Currently I can envisage two possible applications. The first one is implementing the originally conceived translation system for automatic translation from English to Bulgarian. Secondly, having in mind the current interest in providing natural language access to structured data bases (see 6.2), an attempt could be made to extend the system to serve as a general purpose front end for data base access i.e. as a portable natural language query analyser applicable to the different contents of queries addressed to distinct data bases.

6.4. Conclusions.

Even though development of the natural language processing system described in this thesis is still continuing, I believe that the project as a whole has achieved its initial objectives. The system is capable of handling texts of average complexity, while performing in a highly ambiguous environment, both syntactically and semantically. The dictionary currently used by the parser is not very big, but certainly is not small either, and expanding it is trivial. A complete processing cycle is performed, over a range of sentences, which starts with text analysis, with emphasis on the ever-present problem of multiple choice, goes through an intermediate unambiguous meaning representation of the input, and then expresses that in English, thus offering a natural way of validating the results of the analysis process. The system performs creditably over a range of sentences, and resolves word-sense and structural ambiguities in the input correctly and efficiently, as both performance statistics and analysis tracings show. The system performance does not become dramatically worse when real life texts have to be handled. Closer examination of the processes involved in the cycle described above demonstrates that the external manifestation of the program work is not merely a result of clever surface pattern matching, but reflects the systematic application of a set of principles derived within a general framework for text analysis in a highly ambiguous environment.

Evaluating the system in the light of the overall objectives of this project, it is possible to draw certain conclusions.

(1.) It is feasible to design a natural language analysis processor based on the judgemental aspect of semantics. Furthermore, any approach to language analysis has to acknowledge the fact that the problem of multiple choice needs a solution, and this has to lie within a general framework rather than rely on applying randomly designed and/or ad hoc rules.

(2.) In a highly ambiguous environment, an overall strategy of global, context-sensitive, semantic pattern matching is more effective than active parsing based on semantic expectations/ predictions. The strategy is embodied in the "semantic wait-and-see" principle which is the underlying and guiding principle for the design of the parser.

(3.) The utilisation of syntactic information during the analysis process is not only useful, but necessary for the computer 'understanding' of text. Furthermore, semantic judgement can be incorporated in the process of syntactic recognition, thus allowing the syntactic and semantic processes to be carried out virtually concurrently.

(4.) Such an approach is the basis for a considerable improvement in the performance of the parser: the close interface between

syntactic preprocessing and semantic valuation of a constituent, when organised properly, can be extremely efficient.

(5.) The dependency structure delivered as a result of the analysis is clearly unambiguous, sufficiently removed from the surface text, and informative enough to be handled conveniently by a separate and independent generation program.

(6.) The generator can be designed to be flexible, versatile, and well-structured so that minimal effort, and almost no changes in the program control structure and code, will be needed in order to adjust it to generate another language than English from the (largely) language independent semantic representation.

The framework defined by (1)-(6) above allows a systematic and general approach to text processing, possibly also applicable to tasks other than translation/ paraphrase.

Appendix (i)

The Program.

What follows is a brief specification, on a structural rather than on a functional description basis, of the implemented natural language analysis system. The program is written entirely in LISP, the language being an obvious choice for a number of reasons:

(a.) it is very convenient for symbol manipulation; it is still the most widely used language in the field of natural language processing, and is the only apprpriate language available on the Cambridge IBM 370/165;

(b.) the programmer is allowed to blur the distinction between code and data, which enables him to organise the control structure of the program in an extremely flexible and efficient way. In an environment where managing 15000 lines of source code is a perpetual task, this is an invaluable asset;

(c.) the ATN formalism, which is in the basis of this project, is defined in such a way that the grammar can be naturally represented as a LISP data structure (see Appendix (iii)); furthermore, the tests and actions on the arcs may be arbitrary functions in a functional specification language, which LISP is. If the ATN interpreter itself is written in LISP, it will have the simplest possible interface to the grammar, as well as a convenient mechanism (the EVAL concept) of transferring control directly to the arcs of the grammar (see b. above).

The system consists of two largely independent parts: the analyser and the generator. The only way of communication between the two is via the dependency structure, which is the output from the first, and the input to the second phase of the sentence processing cycle. The two programs are divided into a number of modules, each containing functions relevant to a particular aspect of the overall text analysis (or generation) process.

The Parser.

The following modules are invoked during the process of sentence analysis:

BCKTRACK
COMMON
COMPL
CONTROL
CTRANS
FUNCDEFS
GENERAL
GERUND
INIT
MAINLOOP
MODIFY
MORPH
NPBUILD
ORG
PICKADJ

PICKNOUN
PICKVERB
PPHANDLE
READDICT
SBUILD
SORTER
SORTPACK
SUMMARY
VBHANDLE.

The entry point to the program is in "MAINLOOP"* where the basic sentence processing cycle (ANALYSE) is defined, and after a call to INITIALISE, activated. INITIALISE is defined in "INIT", together with READSENTENCE (the program runs offline, non-interactively, due to the large computer resources required) and READGRAMMAR; READDICT, which is also called from INITIALISE, is defined in a separate module of the same name, containing functions which scan the master dictionary, construct a subdictionary of the words which will be used during the run, and set the property lists of the lexical items in question to the values of their various attributes (see Appendix (ii)). In the process of doing so various auxiliary functions may be called to perform morphology on the lexical item ("MORPH" is based on Winograd's treatment of the subject), or just to initialise properties and perform bookkeeping ("SORTER") and general sorting ("SORTPACK"). The true analysis process starts by executing the instruction (Parse S/) which activates a (PUSH S/...) and transfers control to the function 'Parse' in "CONTROL".

Below is a top-down organised description of the parser modules and the functions contained in them.

CONTROL: responsible, through the function 'Parse', for maintaining the major loop of the ATN interpreter: the arcs leaving a state are tested one by one and the first possible transition is carried out. In case a dead end is reached, or a POP arc encountered, the process of backtracking is activated.

GENERAL: the two most important functions here are FOLLOW which determines whether an arc currently being investigated can be followed. If so, PERFORM is invoked which starts executing the actions on the arc. In a not entirely obvious way, i.e. by calling EVAL on each action as specified in the grammar, this function passes control to

FUNCDEFS: this contains the definitions of the actions to be found in the grammar specification (such as SETR, SENDR, GETF, etc.). Note that due to the lack of distinction between program and data exhibited by LISP (see b. above), it is possible to have general LISP system functions on the arcs, as well as ATN specific ones. This allows for an extremely flexible and powerful grammar definition.

---

* the following convention is adopted: identifiers in upper case surrounded by double quotes are names of modules, otherwise they refer to function names.

BCKTRACK: contains the functions which perform the book-keeping work related to saving the current state of the analysis and later, when need be (see CONTROL), restoring it.

At certain points of the analysis process, semantic routines are called from the arcs of the grammar. These are either the sentence/ clause specialist, or the noun group specialist.

NPBUILD: contains the major functions which at one point or another participate in the process of constructing the semantic representation of the noun phrase. The pilot routine is NPBUILD, which, depending on context, might refer to (MAKE-TRACE-IN REL-CLAUSE) and a whole host of auxiliary functions, or transfer control to another module, "PICKADJ" (see below), to process adjectival premodifiers.

SBUILD: the master function SBUILD is defined here. It only sets up the basic skeleton semantic structure of the clause unit, and tests the context to determine which specialist structure building routine to activate. The module also contains the definitions of STRUCTURE-BASIC (see 4.5), and FINAL-TOUCHES-TO-THE-STRUCTURE (see below).

PICKADJ: this is where the sister function to STRUCTURE-BASIC is defined: STRUCTURE-WITH-PRED-ADJECTIVE. It also contains functions like ADJ-NOMINAL-MATCH and ADJ-MODS-MATCH, together with their auxiliaries, which are the procedural embodiment of the mechanism for adjective frames matching. ADJ-NOMINAl-MATCH can in addition be accessed from NPBUILD.

CTRANS: this defines another specialist structure-building function, COMPLEX-TRANSITIVE-STRUCTURE. The main function is just a switch; the actual work-horse routines are STRUCTURE-WITH-COMPL-OBJECT ("They made him president") and STRUCTURE-WITH-COMPL-ADJECTIVE. ("They made him happy").

Next come the modules which can be accessed from any of the structure building specialists above. These contain functions which embody the mechanisms for contextual verb frames application, templates and preplates matching, handling of postmodifying complement constructs, and so on.

PICKVERB: the module organises and supervises the process of contextual verb frames application. The master routine is DISAMBIGUATE-VERBS which is accessible from STRUCTURE-BASIC in "SBUILD". All the auxiliary functions for the process: FORCED-CHOICE-PREP, FORCED-CHOICE-COMPL, ANALYSE-CUES, etc. (see 4.5) are defined here as well; so are MATCH-TEMPLATE-REQUIREMENTS and MATCH-CASE-PREFERENCES. Another important function here which is accessible from all the semantic analysis routines is ITEM-BEST-PREFERRED-FROM which applies the selective principles of preference semantics theory.

PICKNOUN: this is the module which uses the information supplied by the dictionary and attempts disambiguation between noun meanings. The functions that do most of the work are MATCH-SUBJECTS-AGAINST, MATCH-OBJECTS-AGAINST and FILTER-NOMINALS. These are accessed

from all specialist structure building functions in "SBUILD", "PICKADJ" and "CTRANS" (see above).

PPHANDLE:this is another part of the general frames application mechanism: the parsing of optionally used prepositional phrases specified in the verb frames. This is accomplished by INITIAL-PREPLATE-TIES which makes heavy use of PP-MATCH for the actual matching process, and NOUN-DEFS-FITTING-IN-PP which carries out the disambiguation of modifying nominals ("hit Mary with the club"). This module also contains the definiton of the general preplate matching specialist MODIFY (see 4.5), to which INITIAL-PREPLATE-TIES is just a subsidiary. Thus the module is closely interfaced to

MODIFY: this is where all the functions relevant to the algorithm for preplate ties application are defined (4.5). This is done by COMBINE which takes a partial structure and a modifying prepositional phrase and tries to integrate them into one. In the process of doing so, the semantic tests are applied by COMPATIBLE which queries the compatibility between the top level item (potential head) and the currently analysed modifier, and MATCH which provides the interface between the general test application mechanism (as defined in the module) and the preplate stacks as supplied by the dictionary.

The MODIFY specialist can be invoked from all specialised structure building functions (see above). Thus "PPHANDLE" is accessible to "SBUILD", "PICKADJ", "CTRANS" and "NPBUILD".

After the application of frames, templates and preplates has been completed, control is passed back to "SBUILD" by calling FINAL-TOUCHES-TO-THE-STRUCTURE. The function, apart from tidying up the semantic structure built so far, checks for any other postmodifying constituents: to- or that- complements, -ing phrases, etc. The module that gets activated then is

COMPL: this is where the mechanism for dealing with postmodification by clause constructs is defined and specialist semantic and linguistic knowledge relevant to the process embodied (see 3.10, 4.5). The work is done by CONVERT-COMPLEMENT-TO-CASE which decides on the proper functional relation between the currently available (semantic) structure and the postmodifying complement. After the decision has been made, INSERT-COMPLEMENT-CASE incorporates the complement in the structure.

Control is then passed back to SBUILD, where the semantic representation is given its final form and returned as the result of the SBUILD function. The ATN interpreter then resumes the network traversing loop.

The modules and functions described above make extensive use of some auxiliary functions which do most of the low level, behind the scenes, work.

VBHANDLE: This module contains definitions of functions which manipulate the semantic representations of the constituent slot-

fillers and integrate them into the gradually expanding semantic structure. The structure stats from the semantic formula of the main verb of the clause (see 3.8); it is organised, and consequently constructed, around it.

ORG: the functions in this module are in heavy use by all semantic specialists: they know the syntax and semantics of the semantic formulas as well as of the dependency structure. They know how to take a formula apart and extract specific semantic information required by the higher level, more general semantic tests. Names like FORMULA-OF-NPHEAD, LOCATE-CASE, GET-CASE-HEAD, etc. should be self-explanatory. The module also contains some functions subsidiary to keeping the ATN interpreter going: NEXTWORD, FIND-DICT-DEF, etc.

COMMON: the functions defined here are conceptually no different from the ones in "ORG": BELONGS tests for semantic class membership, GET-ITEM always returns the formula for the head-noun in any noun-phrase structure, LOCATEHEAD does the obvious thing suggested by its name. The reason why they are in a separate module is because the same functions are employed by the generator. Hence the name of the module.

Finally, there is the module "SUMMARY" which is a very useful and helpful debugging device. The main function in it is DISPLAY-PARSE-MAP, which is automatically called if the analysis process fails in some way. As a result of the call, a tree is printed out, representing in a conveniently formatted graphical form a complete history of the parsing process, with all its steps, transitions, and decision points clearly displayed. The program contains a switch whereby the user can call the function himself and trace the analysis process.

The Generator.

The generator functions are grouped in modules as follows:

AUX
COMMON
DEEPCASE
INITGEN
MASTER
MORPHGEN
NETFUNCS
RELATIVE
SELECT
SYNNET
SYNONYMS.

The process of paraphrase generation clearly requires some initialisation to take place, and the relevant functions are defined in "INITGEN": INITIALISE-LEXICON, INITIALISE-NOUN-PROPERTIES, INITIALISE-SYNTACTIC-ORDERING. The module also defines the function which effectively interfaces the analyser and the generator: it is called from ANALYSE in "MAINLOOP" (see above), and activates GENERATE:

```
(DE GENERATE-PARAPHRASES (DEP-STRUCTURES)
   (MAPC DEP-STRUCTURES (FUNCTION GENERATE)))
```

Thus control is passed over to

MASTER: the overall structure of the generator is defined by PARAPHRASE which reflects the two stages of the generation process as discussed in chapter5. This starts with calling BUILD-SYNTAX-NET-FOR with the dependency structure delivered by the analyser as its argument. The result is an environment network, later handed over to TRAVERSE-SYNTAX-NET which is directly responsible for generating the sentence. BUILD-SYNTAX-NET-FOR, also defined in the module, can be called recursively to construct a node in the environment network corresponding to an embedded clause in the dependency structure. A verb synonym is selected, and then handed over to USE (also defined here) which carries out all the essential manipulations resulting in the construction of the environment network. The process is one of successive analyses of the constituents dependent on the verb in the dependency structure, and is strongly guided by the syntactico-semantic environment specified by the selected synonym.

SELECT:
SYNONYMS: the major function responsible for the process of synonyms selection, SELECT-MATCHING-SYNONYM, together with the auxiliary ones it needs, are defined here. The interface between these two modules and "MASTER" is not very obvious because of the way a dictionary entry used by the generator is organised. Such an entry attempts to supply information both about a contextual environment required for the selection of a synonym, and about the way in which the components of this environment should be utilised

for the generation of a paraphrase. The general format of a dictionary entry is

```
(verb-sense
   ........
   (synonym
      (SYN-FN1 (CS-TEST1 @constituent-label1))
      (SYN-FN2 (CS-TEST2 @constituent-label2))
      ........)
   ........)
```

CS-TESTi are names of context sensitive tests, themselves defined in "SELECT". The tests' function is to evaluate the global context of the clause (top-level or embedded) as represented by the dependency structure. The functions in "SYNONYMS" can be divided into two classes. Some organise and provide easy access to the dictionary entry. Then there are those that analyse both the dictionary entry and the clause representation and dynamically construct a LISP data structure which represents a call to a specific context sensitive test with its arguments extracted from the dependency structure. This, via ANALYSE-CONTEXT and EXAMINE-CONSTITUENT, is handed over to the LISP EVAL mechanism, thus transferring control back to "SELECT". After all the context sensitive tests in all the synonyms suggested for the verb sense in the dictionary have been performed, SELECT-MATCHING-SYNONYM, which has been accumulating the results of the context evaluation against the requirements of each synonym, returns the result of (MOST-USEFUL-IN SYNONYMS-LIST).

After the synonym verb has been selected, the construction of the environment network begins. Control is back in "MASTER" where a call to (USE *synonym) is in progress. Apart from incorporating into the network information about the form, tense, aspect, negation of the sentence to be generated, the function contains a statement

```
(GENERATE-NODES-FOR *constituents) .
```

Thus control is passed, via GENERATE-NODE-FOR, to

DEEPCASE: this is the module which generates nodes in the network corresponding either to nominal groups linked to the verb in the dependency structure, or to states corresponding to predicate adjectives in the input (embedded clauses are handled by recursive calls to BUILD-SYNTAX-NET-FOR). Consequently, the two entry points to the module are NODE-FOR-NOUN-PHRASE and NODE-FOR-STATE. NODE-FOR-NOUN-PHRASE analyses all the information in the constituents, decides on how to output it ("the man in the park" or "the man who was in the park"), and sorts it into slots: 'possessors', 'adj-premodifiers', 'quantifiers', 'relatives', 'postnominal modifiers'; and then organises the process of node construction so that all this information is incorporated in the node. In case a relative has to be output, PROCESS-RELATIVE-CLAUSE is activated, and control transferred to

RELATIVE: the two important functions here are AUGMENT-NET-WITH-RELATIVE, which clearly involves a recursive call to BUILD-SYNTAX-NET, and RESHUFFLE-RELATIVE-NODE which does postprocessing on the newly created node (for the embedded relative clause) in order to get the data in the two clauses (top-level and embedded) in phase. For example, if it is the 'agent' in the embedded network that is relativised, its output should be suppressed, but the node for 'agent' cannot be deleted altogether because then the morphology on the vrb (see below) will run into problems. So 'agent' is replaced with 'dummy-agent'.

After a complete node for a verb argument - verb dependent constituent in the meaning representation - has been constructed, control is back in USE in "MASTER". This now activates another important function:

(ADD-TO *clause-node <node just constructed>)

which activates the packet of functions in

SYNNET: the module contains two classes of functions. At the lower level come the specialist routines that construct and manilpulate the environment network as a data structure with well defined fixed format: CREATE-SYNTAX-NODE, ADD-TO, etc. Then there are the definitions of the various functions specifying the syntactic environment of the selected synonyms. Referring back to the format of a dictionary entry (see above), SYN-FNi specifies in a compact way what will be the syntactic function of a constituent in the generated paraphrase, if that particular synonym is selected. A dynamically constructed call to such a function might yield something like

(MKOBJ @recipient),
(MKINGCOMPL by ing-compl @manner) etc.

Again a call to EVAL is initiated, via the function CONSULT (also defined here) and the information provided by the statements above is incorporated in the environment network: the node for the recipient in the dependency structure is tagged 'object' and attached to the clause node; The node for the @manner specifying embedded clause in the dependency structure is prefixed by the preposition "by", tagged with 'ing-clause' to denote that it will appear as a postverbal modifier, rather than in a subject position for example, and also added to the clause node. Finally, "SYNNET" defines the function SORTNODE which is called every time a node in the environment network is constructed. Its purpose is to sort the components in the node in an order specifying the order in which they should appear in the generated sentence.

After every constituent in the dependency structure has been analysed and a node for it constructed and linked to its governor, the whole environment network is complete. This terminates the call to BUILD-SYNTAX-NET-FOR in PARAPHRASE, and TRAVERSE-SYNTAX-NET then begins the second phase of the generation process.

NETFUNCS: Each of the nodes in the environment network is treated as

a collection of relation-concept pairs (5.4). Thus the node for a constituent is a small program which specifies how to phrase it. The environment network is a complete program for outputting the whole sentence. The module specifies all the relations that can appear in the network as functions whose effect is to augment the output string. The module also specifies coordinating and auxiliary functions: for example, all of 'agent', 'recipient', 'poss', 'object', etc. depend on a call to OUTPUT-NP. This in its own right uses SELECT-DETERMINER, RESTRICTIVELY-MODIFIED, NUMBER-OF, USE-PRONOUN-FOR, etc. Similarly, 'lexverb', which outputs the verb group, calls EXPANSION with arguments *verb, *tense, *form, *modality, *pncode (person number code), *negation, thus activating the module "MORPHGEN" which performs morphology on the verb. The entry point to "NETFUNCS" is TRAVERSE-SYNTAX-NET, which is effectively defined as

```
(DE TRAVERSE-SYNTAX-NET (*node)
   (COND
      ((NULL *node) NIL)
      (T
         (EVAL (CAR *node))
         (TRAVERSE-SYNTAX-NET (CDR *node)))))
```

Clearly this might be called recursively from functions such as 'thatcompl', 'ing-compl', etc.

Finally, there are the two modules "COMMON", which was discussed above (see "The Parser"), and "AUX", which clearly contains the definitions of some low level functions used by the more specialised generator routines.

Appendix (ii)

A Dictionary Sample.

This appendix shows a representative sample of the dictionary used for the experiments. Altogether the dictionary contains some 400 words, at least half of which have more than one sense.

The program keeps a list of the words that it will need for the current run, with all their relevant attributes - syntactic category, features, verb, noun or adjective definitions, etc. - on the property list of the root form of the surface lexical item.

The actual dictionary organisation of a verb contextual frame has already been shown in 4.5. The preferences for the semantic class membership of the subject (and surface object) of the verb are specified in its semantic formula. Optional modification by a prepositional phrase is indicated by a

(preps (PList F(sc) DLink))

construct, where 'PList' is a list of those prepositions that can introduce a candidate prepositional phrase for inspection, the head noun of the PP must satisfy the semantic test specified by the function F over a certain semantic class (most often this is a simple test for membership), and 'DLink' is the dependency link that will be established between the modifier and the verb in case the test succeeds. Similarly, obligatory postmodification is introduced by

(compulsory (PList F(sc) DLink))

The header 'cues' is used for two different purposes. It can specify constructs like

..... THAT <clause>, or
..... TO *do @act:

by stating

(cues THATCOMP) or (cues TOCOMP),

i.e. the lexical item, if used in that particular sense, can be considered as tagged with THATCOMP or TOCOMP feature. Or, alternatively, it suggests a surface syntactic pattern which may give clues to the particular meaning of the lexical item used. Thus (cues TRANS) vs. (cues INTRANS) may help in interpreting

John grew a beard, vs.

John grew.

Similarly, both (cues INDOBJ) and (cues COMPL-TRANS) specify a surface syntactic pattern '... V NP NP', and will be activated on encountering

John called Bill a taxi/ John called Bill a fool.

The dictionary also specifies frames for nouns (where applicable) and predicate adjectives.

Particled verbs are accessed through an indirection via the 'particles' mechanism. The grammar is suitably designed so that "...call off the strike" is not parsed as "... call (off the strike)".

```
(ADMIT
   (cat VERB)
   (features PASSIVE TRANS INTRANS THATCOMP)
   (vdef
      (admit1
         ((man subj)
            ((*hum obje)
               (((((((same *hum) poss) state) obje)
                     change)
                  cause)
               goal)
            want)))
         (compulsory ((TO IN) *pla location)))
      (admit2
         ((*ani subj) ((sign obje) (true tell)))
         (cues THATCOMP))
      (admit3 ((*ani subj) ((state obje) (true feel))))
      (admit4
         (((*ani subj)
            ((((same * ani) subj) ((act obje) do))
               (true tell)))
         (compulsory (TO *act @act)))
   ))


(ADVISE
   (cat VERB)
   (features TRANS INTRANS THATCOMP)
   (vdef
      (advise1
         ((*hum subj)
            ((*animar obje)
               (((((man subj) ((act obje) do)) cause)
                  goal)
               tell)))
         (cues TOCOMP TRANS))
      (advise2
         ((*hum subj)
            ((*hum obje) (((this sign) obje) tell)))
         (cues THATCOMP INTRANS))
      (advise3
         ((*hum subj)
            ((*hum obje)
               ((((((same *hum) subj) ((act obje) notdo))
                     cause)
                  goal)
               tell)))
         (compulsory
            (AGAINST *mar object))))
   ))


(AFRAID
   (cat ADJ)
   (features THATCOMP TOCOMP LOWSUBJ)
   (adjdef
      (afraid1
         ((*hum poss)
            (((*ent reason) (notplease feel)) kind))
         (preps
            (OF *ent reason)))
      (afraid2
         ((*hum poss) ((((*act obje) do) notwant) kind))
         (cues TOCOMP))
      (afraid3
         ((*hum poss) ((((*mar obje) (true be)) think) kind))
         (cues THATCOMP))
```

```
   ))


(ASK
   (cat VERB)
   (features
      TRANS PASSIVE INTRANS TOCOMP TRANSCOMP INDOBJ)
   (vdef
      (ask1
         ((man subj) ((*ani obje) ask))
         (preps (ABOUT *ent subj-matter))
         (cues INDOBJ))
      (ask2
         ((man subj)
            ((act obje)
               ((((man (please feel)) cause) goal) ask)))
         (cues TOCOMP))
      (ask3
         ((man subj) ((*ent obje) ((*hum from) want)))
         (compulsory (FOR *inan neutral)))
   ))


(CALL
   (cat VERB)
   (features TRANS PASSIVE INTRANS COMPL-TRANS)
   (particles OFF CALLOFF FORTH CALLFORTH OUT CALLOUT)
   (vdef
      (call1
         ((man subj)
            ((*animar obje)
               ((((much (hear sense)) cause) goal)
                                          tell))))
      (call3
         ((man subj)
            ((*hum obje)
               (((((act obje) do) cause) goal) ask)))
         (cues TOCOMP))
      (call2
         ((man subj)
            ((self obje) (((where point) to) move)))
         (preps
            (ON *hum recipient)
            (AT *pla location)))
      (call4
         ((*hum subj)
            ((*ani obje)
               (((same *ani) ((sign obje) have)) think)))
         (cues COMPL-TRANS))
      (call6
         ((*ent subj) ((*ent obje) want))
         (cues INTRANS)
         (compulsory (FOR *ent objective)))
      (call6
         ((*ent subj) ((act obje) want))
         (cues INTRANS)
         (compulsory (FOR act objective)))
      (call7
         ((man subj) ((*inan obje) want))
         (cues INDOBJ))
      (call7
         ((man subj) ((*inan obje) ((man for) want)))
         (compulsory (FOR man recipient)))
   ))


(CALLOFF
   (cat VERB)
   (features TRANS)
   (vdef
```

```
      (calloff
         ((*org subj) ((*ac obje) notdo)))))


(CALLOUT
   (cat VERB)
   (features TRANS INTRANS)
   (vdef
      (callout
         ((man subj)
            ((*mar obje)
               ((((much (hear sense)) cause) goal) tell)))
                                        )))


(CALLFORTH
   (cat VERB)
   (features TRANS)
   (vdef
      (callforth
         ((*pot subj) ((*ent obje) want)))))


(EFFORT
   (cat NOUN)
   (features TOCOMP)
   (ndef
      (effort
         (((*hum subj)
             (((((this evnt) hapn) cause) goal) do))
            (obje act)))))

(STOP
   (cat VERB)
   (features TRANS INTRANS TOCOMP)
   (vdef
      (stop1
         ((*org subj) ((*inan obje) (notmove cause)))
         (preps
            (FROM *act avoidance))
         (cues TRANS))
      (stop2
         ((*pot subj) notmove)
         (cues INTRANS)
         (preps
            ((IN AT ON) *pla destination)))
      (stop3
         ((*pot subj) ((act obje) notdo))
         (cues ING/PHRASE TRANS))
      (stop4
         ((*hum subj) ((((act obje) do) goal) notmove))
         (cues TOCOMP))))
```

Appendix (iii)

The Augmented Transition Network Grammar.

What follows is an annotated listing of the Augmented Transition Network Grammar used by this system at the time of writing. As will become clear from the examples in this appendix and in Appendix (iv), the grammar offers coverage of a substantial non-trivial subset of English language constructs; in any case large enough to provide a solid basis for testing the ideas presented in this thesis. Among the phenomena not accounted for are: complex determiner structures, quantification, existential "there", complex adjectival (comparative and superlative) structures, questions, conjunctions, etc. It has been demonstrated by Woods [Woods72] that these can be accommodated in the ATN framework and a syntactic recogniser which can handle these and other constructs has been tested in action. In the course of this research it was felt to be more important to concentrate on the essential objectives of this work, rather than to develop an extensive grammar of English - an effort which would not be justified at this stage because it would not necessarily contribute substantially to the understanding of the semantic processes taking place during sentence analysis.

The grammar starts (by looking for the possible sentence openers:

noun-phrases:

she asked questions with great interest;

"to-" infinitive nominal clauses (with or without a subject):

for a research student to work at night is normal,

to tell lies is wrong,

to call on Mary, John called a taxi;

"that-" complements:

that John ran away became known soon;

nominal "-ing" clauses:

telling lies is wrong;

"-ing" participle phrases (with or without a subject):

calling on Mary, John decided to ask for the book,

her aunt having left the room, I declared my passionate love for Celia;

extraposed subject complement - leading "it":

it is clear that Bill loves Janet,

it is important to tell the truth;

prepositional phrases:

in the morning he runs in the park.

```
(S/
   (PUSH
      COMPL/
      (OR (WRD TO) (AND (WRD TO NEXTWORD) (CAT NEG)))
      (SENDR fcflag 'T)
      (SENDR in-compl-subj T)
      (SETR type 'subj-compl)
      (SETR fronted-compl *)
      (TO S/))
   (JUMP
      S/NP
      (AND (CAT VERB) (GETR fronted-compl))
      (SETR subj (CADDR (GETR fronted-compl)))
      (SETR type 'subj-compl)
      (SETR fronted-compl NIL)
      (RPLACD (LOOKFOR 'type (GETR subj)) (LIST 'for-to))
```

```
    )
   (PUSH ING/PHRASE (CAT VERB)
      (SENDR subj '((someone (man))))
      (SENDR in-compl-subj T)
      (SETR compl *)
      (SETR fronted-ing T)
      (SETR type 'subj-compl)
      (TO S/DCL))
   (PUSH ING/PHRASE/SUBJ T
      (SETR after *)
      (TO S/DCL))
   (JUMP S/DCL T (SETR type 'dcl)))


(S/DCL
   (PUSH NP/ (NPSTART)
      (SENDR subjNP 'T)
      (SETR subj *)
      (COND
         ((OR
             (CONTAINS-ANYWHERE (GETR subj) 'IT)
             (CONTAINS-ANYWHERE (GETR subj) 'it))
            (SETR it-subject T)))
      (COND
         ((GETR fronted-ing)
            (SETR while (GETR compl))
            (SETR compl NIL)
            (RPLACD
               (LOOKFOR 'agent (GETR while))
               (GETR subj))))
      (COND
         ((GETR fronted-compl)
            (SETR compl (GETR fronted-compl))
            (SETR fronted-compl NIL)
            (RPLACD
               (LOOKFOR 'agent (GETR compl))
               (GETR subj))))
      (TO S/NP))
   (PUSH COMPL/ (WRD (FOR THAT))
      (SENDR in-compl-subj T)
      (SETR subj *)
      (SETR subj (CADDR (GETR subj)))
      (SETR type 'subj-compl)
      (TO S/NP))
   (WRD !, T (TO S/DCL))
   (PUSH PP/
      (AND
         (CAT PREP)
         (NOT (WRD OF))
         (NULLR fronted-ing)
         (NULLR fronted-compl)
         (NULLR after))
      (ADD mods *)
      (TO S/DCL))
   (JUMP S/NP (GETR fronted-ing)
      (SETR subj (CADDR (GETR compl)))
      (SETR compl NIL)))
```

The grammar then proceeds to identify the verb of the clause. The item picked up at S/NP may be a modal, or an auxiliary, hence the loop on VP/V: it recognises possible perfect-progressive-passive constructs. Alternatively, a different path through the network may be followed, on identifying an adjective after a COPULA verb.

```
(S/NP
   (CAT
      VERB
      T
      (COND
         ((MODAL) (SETR modal *))
         (T (SETR v *)))
      (COND
         ((AND (GETR whq) (NOR (MODAL) (WRD (HAVE BE))))
            (SETR subj (GETR whq))
            (SETR whq NIL)))
      (SETR tns (GETF TNS))
      (TO S/AUX))
   )


(S/AUX
   (CAT
      NEG
      (OR (GETR modal) (WRD (HAVE BE) v))
      (COND
         ((WRD DO modal) (SETR modal NIL)))
      (SETR neg 'neg)
      (TO S/AUX))
   (JUMP VP/V T))


(VP/V
   (CAT
      VERB
      T
      (COND
         ((GETF pastpart)
            (COND
               ((WRD BE v)
                  (SETR obj (GETR subj))
                  (SETR subj NIL)
                  (SETR agflag 'T)
                  (ADD aspect 'passive))
               ((AND (NULLR aspect) (WRD HAVE v))
                  (ADD aspect 'perfect))
               (T (ABORT))))
         ((GETF prespart)
            (COND
               ((WRD BE v) (ADD aspect 'progressive))
               (T
                  (SETR aborted-ing T)
                  (ABORT))))
         ((OR (NOT (GETF UNTENSED)) (GETR v)) (ABORT)))
      (SETR v *)
      (TO VP/V))
   (CAT ADJ (RFEAT COPULA v)
      (SETR adj/pred *)
      (TO VP/ADJ))
   (WRD LIKE T
      (SETR vp/like T)
      (TO VP/LIKE))
   (JUMP VP/HEAD T))
```

The predicate adjective network allows the adjective to be followed by various complement phrases:

John is easy <u>to please</u>,

John is eager to please,

it is important not to panic,

John is afraid that Bill loves Mary;

or prepositional phrases:

the crook is afraid of the policeman,

they were angry about the inflation;

or nothing:

Bill grew sad.

```
(VP/ADJ
   (PUSH COMPL/
      (AND
         (WRD THAT)
         (RFEAT THATCOMP adj/pred)
         (OR
            (GETR it-subject)
            (IS-A-HUMAN (CAR (GETR subj)))))
      (SETR adj-compl *)
      (COND
         ((GETR it-subject)
            (SETR
               subj
               (LIST
                  (COND
                     ((LOOKFOR 'clause (GETR adj-compl)))
                     ((LOOKFOR 'S (GETR adj-compl)))))))
         (T
            (SETR compl *)))
      (TO S/POP/S))
   (PUSH FOR/NP
      (AND
         (OR
            (WRD TO)
            (AND (CAT NEG) (WRD TO NEXTWORD)))
         (NOT
            (CONTAINS-ANYWHERE (GETR subj) 'it))
         (NOT
            (CONTAINS-ANYWHERE (GETR subj) 'IT))
         (RFEAT TOCOMP adj/pred))
      (*
         (COND
            ((RFEAT LOWOBJ adj/pred)
               (SENDR in-adj-compl T)
               (SENDR obj (GETR subj))
               (SENDR subj
                  (QUOTE ((someone (man))))))
            ((OR
                (RFEAT LOWSUBJ  adj/pred)
                (NOT (RFEAT LOWOBJ adj/pred)))
               (SENDR in-adj-compl T)
               (SENDR subj (GETR subj))
               (SENDR might-need-an-object T))))
      (SETR adj-compl *)
      (COND
         ((RFEAT LOWOBJ adj/pred)
            (SETR
               subj
               (LIST
                  (COND
```

```
                    ((LOOKFOR 'clause (GETR adj-compl)))
                    ((LOOKFOR 'S (GETR adj-compl)))))))
        (T
           (SETR compl *)))
     (TO S/POP/S))
  (PUSH FOR/NP
     (AND
        (WRD TO)
        (GETR it-subject))
     (SENDR subj (QUOTE ((someone (man)))))
     (SETR subj *)
     (TO S/POP/S))
  (PUSH PP/ (CAT PREP)
     (SENDR subj (GETR subj))
     (ADD mods *)
     (TO VP/ADJ))
  (JUMP S/POP/S T))


(VP/LIKE
  (PUSH NP/ T
     (COND
        ((NULLR *) (ABORT)))
     (SETR similarity *)
     (TO VP/PRED/NOM)))


(VP/PRED/NOM
  (PUSH PP/ (CAT PREP)
     (ADD mods *)
     (TO VP/PRED/NOM))
  (JUMP S/POP/S T))
```

At VP/HEAD, apart from picking up the direct object, the grammar tries to recognise some of the possible postmodifiers:

particles:

the workers called off the strike at the last moment;

sentencial complements:

he wants to study in the university,

he hopes for his son to study in the university,

he expects that he will study in the university;

Also, when parsing relative clauses (see below), this is where the direct object is picked up:

I heard the story that John told Mary.

```
(VP/HEAD
  (CAT PREP (SETQ temp (VPARTICLE v))
     (SETR v temp)
     (SETR *particle (CURRENTWORD))
     (TO VP/HEAD))
  (WRD BY (NULLR subj) (JUMP S/POP/S))
  (PUSH
```

```
   COMPL/
   (AND
      (OR
         (WRD (FOR TO THAT))
         (AND (CAT NEG) (WRD TO NEXTWORD)))
      (NULLR obj)
      (NULLR in-rel-clause))
   (*
      (COND
         ((OR (WRD TO) (CAT NEG))
            (SENDR subj (GETR subj)))))
   (SETR compl *)
   (TO S/POP/S))
(PUSH
   NP/
   (AND
      (RFEAT INDOBJ v)
      (GETR in-rel-clause)
      (GETR deleted-NP)
      (NPSTART))
   (COND
      ((IS-A-HUMAN (CAR (GETR deleted-NP)))
         (SETR obj *)
         (ADD
            mods
            (BUILDQ (PP (prep TO) +) deleted-NP)))
      (T
         (SETR obj (GETR deleted-NP))
         (ADD mods (BUILDQ (PP (prep TO) *)))))
   (SETR deleted-NP NIL)
   (SETR indobjflag 'T)
   (JUMP VP/NP))
(TST
   HELD-REL-NOMINAL
   (AND (GETR in-rel-clause) (GETR deleted-NP))
   (SETR obj (GETR deleted-NP))
   (SETR deleted-NP NIL)
   (SETR transflag 'T)
   (JUMP VP/NP))
(PUSH
   NP/
   (AND
      (RFEAT TRANS v)
      (OR
         (NULLR in-rel-clause)
         (AND
            (GETR in-rel-clause)
            (NULLR deleted-NP)))
      (NPSTART))
   (SENDR subj (GETR subj))
   (SENDR v (GETR v))
   (SETR obj *)
   (COND
      ((AND
          (GETR in-adj-compl)
          (GETR might-need-an-object))
         (SETR might-need-an-object NIL)))
   (SETR transflag 'T)
   (TO VP/NP))
(PUSH PP/
   (AND (CAT PREP) (WRD TO))
   (SENDR subj (GETR subj))
   (SETR odd-PP T)
   (ADD mods *)
   (TO VP/HEAD))
(TST ADJ-COMPL-NEEDS-AN-OBJECT
   (AND
      (GETR in-adj-compl)
      (NULLR obj)
      (GETR might-need-an-object)
      (RFEAT TRANS v))
   (SETR obj (QUOTE ((someone (man)))))
   (SETR might-need-an-object NIL)
```

```
      (JUMP S/POP/S))
   (JUMP VP/NP
      (AND
         (NULLR deleted-NP)
         (OR (RFEAT INTRANS v) (GETR obj)))))
```

At VP/NP the object of the verb has been identified. Here possible complex-transitive or ditransitive structures are recognised:

the news made him happy,

they made him a president,

Mary made Bill a cake.

The object can be followed by a complement:

Bill expected Mary to go home,

Bill told Mary that she must go home.

Prepositional phrases acting as verb or object postmodifiers are recognised here as well:

I saw the man in the park with the telescope.

```
(VP/NP
   (CAT ADJ (RFEAT ADJCOMP v)
      (SETR adj/comp *)
      (TO S/POP/S))
   (PUSH COMPL/
      (AND (WRD THAT) (GETR it-subject))
      (SETR compl *)
      (COND
         ((AND (GETR agflag) (GETR it-subject))
            (SETR obj NIL)
            (SETR subj '((someone (man)))))
         (T
            (PRINTC "*** apparently an unexpected case:")
            (MAPC *REGLIST (FUNCTION SUPERPRINT))))
      (TO S/POP/S))
   (PUSH
      COMPL/
      (AND
         (WRD THAT)
         (OR
            (AND (RFEAT INDOBJ v) (GETR obj))
            (AND (NULLR obj) (GETR odd-PP))))
      (SETR compl *)
      (COND
         ((GETR obj)
            (ADD mods (BUILDQ (PP (prep TO) +) obj))
            (SETR obj NIL)))
      (COND
         ((GETR odd-PP) (SETR odd-PP NIL)))
      (SETR indobjflag 'T)
      (TO S/POP/S))
   (PUSH
      FOR/NP
      (AND
         (OR
```

```
            (WRD TO)
            (CHECKF VERB UNTENSED)
            (AND (WRD TO NEXTWORD) (CAT NEG)))
         (OR (GETR obj) (GETR mods)))
      (SENDR topsubj (GETR subj))
      (SENDR topobj (GETR obj))
      (SENDR intocompl 'T)
      (SETR compl *)
      (COND
         ((AND
             (RFEAT INDOBJ v)
             (GETR obj))
            (ADD mods (BUILDQ (PP (prep TO) +) obj))
            (SETR obj NIL))
         ((GETR mods)
            (SETR goal (GETR compl))
            (SETR compl NIL)))
      (COND
         ((RFEAT DELTOPOBJ v)
            (SETR obj NIL)))
      (TO S/POP/S))
   (PUSH
      NP/
      (AND
         (NULLR in-rel-clause)
         (NPSTART)
         (OR (GETR obj) (GETR odd-PP))
         (OR (RFEAT INDOBJ v) (RFEAT COMPL-TRANS v)))
      (SETR result *)
      (COND
         ((GETR odd-PP)
            (SETR obj *)
            (SETR odd-PP NIL))
         ((OR
             (RFEAT INDOBJ v)
             (AND
                (RFEAT COMPL-TRANS v)
                (SINGLE (GET-NOUN-DEFS (GETR result)))
                (NOT (IS-A-HUMAN (CAR (GETR result))))))
            (ADD mods (BUILDQ (PP (prep TO) +) obj))
            (SETR indobjflag 'T)
            (SETR result NIL)
            (SETR obj *))
         (T
            (SETR compl/trans/flag 'T)
            (SETR obj2 *)))
      (TO S/POP/S))
   (PUSH PP/
      (AND
         (CAT PREP)
         (NOT (WRD OF))
         (OR
            (GETR *particle)
            (AND (NULLR *particle) (NOT (VPARTICLE v)))))
      (SENDR subj (GETR subj))
      (ADD mods *)
      (TO VP/NP))
   (JUMP S/POP/S T))
```

S/POP/S is the final state of the sentence/ clause network. Jump to S/POP/REL if a relative clause has been processed, where a pointer to the deleted nominal (the head of the embedding noun phrase) is incorporated in the final structure - it will be used later by NPBUILD. Similarly, jump to S/POP/TOCOMP if a to-complement has been recognised, and in the process of doing so set up

the deleted subject of the embedded clause. S/POP/S itself provides directives for finding the subject of a passivised sentence:

John was beaten by Bill,

or nominal "-ing" clauses as direct objects or postmodifiers:

John loves running in the park,

John killed Bill firing a gun.

Finally, a semantic representation for the recognised clause is constructed (by SBUILD) and popped.

```
(S/POP/S
   (JUMP S/POP/REL (GETR in-rel-clause))
   (JUMP
      S/POP/TOCOMP
      (AND (GETR intocompl) (NULLR odd-PP) (NULLR subj))
      (COND
         ((AND
             (GETR topobj)
             (IS-A-HUMAN (CAR (GETR topobj))))
            (SETR type 'for-to)
            (SETR subj (GETR topobj)))
         (T
            (SETR type 'in-order-to)
            (SETR subj (GETR topsubj)))))
   (WRD BY (NULLR subj) (TO VP/AGT))
   (JUMP S/POP/S (NULLR subj)
      (SETR subj '((someone (man)))))
   (PUSH ING/PHRASE
      (AND
         (CAT VERB)
         (NOT (EQ 'BE (GETR v)))
         (NULLR aborted-ing)
         (NOR (RFEAT INDOBJ v) (RFEAT COMPL-TRANS v)))
      (SENDR subj (GETR subj))
      (SETR compl *)
      (TO S/POP/S))
   (PUSH PP/ (AND (CAT PREP) (GETR indobjflag))
      (SENDR subj (GETR subj))
      (ADD mods *)
      (TO S/POP/S))
   (JUMP S/POP/S
      (AND
         (NULLR s-pop-val)
         (NULLR s-built)
         (GETR subj)
         (OR
            (NOT (CAT VERB))
            (AND (CAT VERB) (GETR in-compl-subj)))
         (NULLR odd-PP)
         (NULLR in-rel-clause)
         (NULLR might-need-an-object)
         (NULLR intocompl))
      (SETR s-built T)
      (SETR s-pop-val (SBUILD)))
   (POP (GETR s-pop-val)
         (GETR s-pop-val)))
```

```
(S/POP/REL
   (POP
      (COND
         ((GETR poss-rel)
            (BUILDQ
               (@@poss (@@agent . +)  )
               governor
               (BUILDQ (rel (path +)  ) path (SBUILD))))
         (T (BUILDQ (rel (path +)  ) path (SBUILD))))
      T))


(S/POP/TOCOMP
   (JUMP S/POP/TOCOMP
      (AND
         (NULLR s-tocomp-val)
         (NULLR tocomp-built))
      (SETR tocomp-built T)
      (SETR s-tocomp-val (SBUILD)))
   (POP (GETR s-tocomp-val) (GETR s-tocomp-val)))


(VP/AGT
   (PUSH NP/ T
      (COND
         ((NULLR *) (ABORT)))
      (SETR subj *)
      (TO VP/NP)))
```

The noun phrase network recognises constructs of the following classes:

pronouns;

proper names;

the basic noun group:

(det) (quant) (adj)* noun:

the five big green monsters;

nouns preceded by possessors:

John's crook,

the lazy dog's wagging tail;

"of-" genitives:

the wagging tail of the lazy dog;

nouns premodified by present participles:

kissing aunts;

nouns postmodified by complements:

an opportunity to study in the university,

the fact that kissing aunts can be boring;

relative clauses (via the REL/ or ING/RELATIVE networks):

the man who fell to earth,

the friend with whom I saw the film,

the friend whose book I am reading,

the man teaching Mary;

postmodifying prepositional phrases in noun phrases in subject position:

the man in the street...

```
(NP/
   (CAT DET T
      (SETR det (BUILDQ (det *)))
      (TO NP/DET))
   (CAT QUANT T
      (SETR quant *)
      (SETR quant (FORMULA (GETR quant) 'NOUN))
      (TO NP/DET))
   (CAT PRO T
      (SETR n (BUILDQ (pro *)))
      (SETR inhibit-rel T)
      (SETR nu (GETF number))
      (TO NP/HEAD))
   (JUMP NP/DET T))


(NP/DET
   (CAT NOUN T (SETR n (BUILDQ (n *)))
      (SETR nu (GETF number)) (TO NP/HEAD))
   (CAT ADJ T
      (ADD adjmods *)
      (TO NP/DET))
   (CAT VERB (GETF prespart)
      (SETR v-ing *)
      (TO NP/DET))
   (CAT POSS T
      (SETR possessor *)
      (SETR possessor (FORMULA (GETR possessor) 'POSS))
      (TO NP/DET))
   (CAT NPR (NULLR det)
      (SETR n (BUILDQ (npr *)))
      (SETR inhibit-rel T)
      (SETR nu 'SG) (TO NP/HEAD))
   (JUMP NP/HEAD T))


(NP/HEAD
   (CAT POSS (GETR n)
      (SETR possessor (NPBUILD))
      (SETR inhibit-rel NIL)
      (TO NP/DET))
   (PUSH
      FOR/NP
      (AND
         (OR (WRD TO) (AND (WRD TO NEXTWORD) (CAT NEG)))
         (RFEAT TOCOMP n))
      (SENDR subj (GETR subj))
      (SENDR intocompl 'T)
```

```
   (SETR ncompl *)
   (TO NP/POP))
(PUSH COMPL/
   (AND (WRD THAT) (RFEAT THATCOMP n))
   (SETR ncompl *)
   (TO NP/POP))
(PUSH
   REL/
   (AND
      (GETR n)
      (NULLR inhibit-rel)
      (OR
         (AND
            (WRD (WHO WHOM WHOSE))
            (SETR *animates (HUMANS-IN (GETR n))))
         (AND
            (WRD WHICH)
            (SETR *inanimates (NON-HUMANS-IN (GETR n))))
         (OR
            (WRD THAT)
            (AND
               (WRD (WHICH WHOM WHOSE) NEXTWORD)
               (CAT PREP)))))
   (*
      (COND
         ((GETR *animates)
            (SENDR type 'relative)
            (SENDR in-rel-clause 'T)
            (SENDR
               rel-nominal
               (CUT-DOWN (NPBUILD) (GETR *animates))))
         ((GETR *inanimates)
            (SENDR type 'relative)
            (SENDR in-rel-clause 'T)
            (SENDR
               rel-nominal
               (CUT-DOWN (NPBUILD) (GETR *inanimates))))
         (T
            (SENDR type 'relative)
            (SENDR in-rel-clause 'T)
            (SENDR rel-nominal (NPBUILD)))))
   (SETR rel-clause *)
   (TO NP/POP))
(PUSH ING/RELATIVE (CAT VERB)
   (SENDR type 'relative)
   (SENDR in-rel-clause T)
   (SENDR subj (NPBUILD))
   (SETR rel-clause *)
   (TO NP/POP))
(PUSH PP/ (WRD OF)
   (SETR possessor *)
   (SETR possessor (CADDR (GETR possessor)))
   (TO NP/HEAD))
(PUSH PP/
   (AND
      (CAT PREP)
      (NOT (WRD OF))
      (GETR subjNP)
      (NULLR inPP))
   (ADD nmods *)
   (TO NP/HEAD))
(JUMP NP/POP (GETR n)))
```

Finally, the subnetworks for recognising relative clauses (all the states prefixed with REL/), "-ing" phrases (ING/) and complements (COMPL/ and FOR/) allow parsing all of the constructs used in the examples above.

```
(ING/RELATIVE
   (CAT VERB (GETF prespart))
      (SETR v *)
      (SETR path (GETR subj))
      (TO VP/V)))


(ING/PHRASE/SUBJ
   (PUSH NP/ (NPSTART)
      (SENDR subjNP T)
      (SETR subj *)
      (TO ING/PHRASE/AUX)))


(ING/PHRASE/AUX
   (WRD HAVING T
      (TO ING/PHRASE/VERB)))


(ING/PHRASE/VERB
   (CAT VERB (GETF pastpart)
      (SETR vger *)
      (SETR tns 'past)
      (TO ING/PHRASE/OBJ)))


(ING/PHRASE
   (CAT VERB (GETF prespart) (SETR vger *)
      (TO ING/PHRASE/OBJ)))


(ING/PHRASE/OBJ
   (PUSH
      VP/HEAD
      T
      (SENDR
         subj
         (COND
            ((GETR subj))
            (T '((someone (man))))))
      (SENDR v (GETR vger))
      (SENDR in-compl-subj (GETR in-compl-subj))
      (SETR ing/pop *)
      (TO ING/PHRASE/POP)))


(ING/PHRASE/POP
   (POP (BUILDQ (compl ING/TYPE +) ing/pop) T))


(NP/POP
   (JUMP NP/POP
      (AND (NULLR np-pop-val) (NULLR np-built))
      (SETR np-built T)
      (SETR np-pop-val (NPBUILD)))
   (POP (GETR np-pop-val) (GETR np-pop-val)))


(COMPL/
   (WRD FOR T (TO FOR/FOR))
   (WRD THAT T (SETR ntype 'THAT) (TO COMPL/NTYPE))
   (WRD THAN T (SETR ntype 'THAN) (TO COMPL/NTYPE))
```

```
   (JUMP
      FOR/NP
      T
      (COND
         ((NULLR subj)
            (SETR subj (QUOTE ((someone (man)))))))))


(COMPL/NTYPE
   (PUSH S/ T
      (SENDR in-compl-subj (GETR in-compl-subj))
      (SETR s *)
      (TO COMPL/S)))


(FOR/FOR
   (PUSH NP/ T
      (SETR subj *)
      (TO FOR/NP)))


(FOR/NP
   (WRD TO T (TO FOR/TO))
   (CAT NEG T (SETR neg 'neg) (TO FOR/NP))
   (JUMP FOR/TO (CHECKF VERB UNTENSED)))


(FOR/TO
   (PUSH
      S/NP
      (CHECKF VERB UNTENSED)
      (SENDR topsubj (GETR topsubj))
      (SENDR topobj (GETR topobj))
      (SENDR intocompl (GETR intocompl))
      (SENDR in-adj-compl (GETR in-adj-compl))
      (SENDR
         might-need-an-object
         (GETR might-need-an-object))
      (SENDR fcflag (GETR fcflag))
      (SENDR in-compl-subj (GETR in-compl-subj))
      (SENDR subj (GETR subj))
      (SENDR obj (GETR obj))
      (SENDR neg (GETR neg))
      (SENDR tns (GETR tns))
      (COND
         ((GETR fcflag) (SENDR type 'in-order-to)))
      (SETR ntype 'NOM)
      (SETR s *)
      (TO COMPL/S)))


(COMPL/S (POP (BUILDQ (compl + +) ntype s) T))


(PP/
   (WRD HOME T
      (SETR prep '(prep TO))
      (SETR prepnp '((home (((man obje) wrap) spread))))
      (TO PP/POP))
   (CAT PREP T (SETR prep (BUILDQ (prep *)))
      (TO PP/PREP)))


(PP/PREP
   (PUSH NP/ T
      (SENDR subj (GETR subj))
```

```
      (SENDR inPP 'T)
      (SETR prepnp *)
      (TO PP/POP))
   (PUSH ING/PHRASE (CAT VERB)
      (SENDR subj (GETR subj))
      (SENDR inPP 'T)
      (SETR tempres *)
      (SETR prepnp (CADDR (GETR tempres)))
      (SETR tempres NIL)
      (TO PP/POP)))


(PP/POP
   (POP
      (BUILDQ (@   + +) (GENSYM1 'PP) prep prepnp)
      T))


(REL/
   (MEM (WHICH THAT WHO) T (TO REL/WH))
   (WRD WHOM T (SETR deleted-NP (GETR rel-nominal))
      (SETR path (GETR rel-nominal))
      (SETR rel-nominal NIL)
      (TO REL/WH))
   (WRD WHOSE T (TO REL/POSS))
   (CAT PREP T (SETR prep *) (TO REL/PREP)))


(REL/WH
   (PUSH PP/ (CAT PREP) (ADD mods *) (TO REL/WH))
   (PUSH
      NP/
      (AND (NOT (CAT VERB)) (NPSTART))
      (COND
         ((GETR rel-nominal)
            (SETR deleted-NP (GETR rel-nominal))
            (COND
               ((NULLR poss-rel)
                  (SETR path (GETR rel-nominal))))
            (SETR rel-nominal NIL)))
      (SETR subj *)
      (TO S/NP))
   (JUMP
      S/NP
      (AND (GETR rel-nominal) (CAT VERB))
      (SETR subj (GETR rel-nominal))
      (COND
         ((NULLR poss-rel)
            (SETR path (GETR rel-nominal))))
      (SETR rel-nominal NIL)))


(REL/PREP
   (MEM
      (WHICH WHOM)
      T
      (ADD
         mods
         (BUILDQ (PP (prep +) +) prep rel-nominal))
      (SETR path (GETR rel-nominal))
      (SETR rel-nominal NIL)
      (TO REL/WH))
   (WRD WHOSE T (TO REL/PREP/POSS)))


(REL/POSS
   (PUSH NP/ T
      (SETR poss-rel 'T)
```

```
      (SETR governor (GETR rel-nominal))
      (SETR rel-nominal *)
      (SETR path (GETR rel-nominal))
      (TO REL/WH)))


(REL/PREP/POSS
   (PUSH NP/ T
      (SETR poss-rel 'T)
      (SETR governor (GETR rel-nominal))
      (ADD mods (BUILDQ (PP (prep +) *) prep))
      (SETR path (GETR rel-nominal))
      (SETR rel-nominal NIL)
      (TO REL/WH)))
```

Appendix (iv)

Examples.

JOHN ASKED A QUESTION.
John questioned.

JOHN ASKED MARY A QUESTION.
John questioned Mary.

JOHN ASKED MARY A QUESTION ABOUT THE BOOK.
John questioned Mary about the book.

JOHN ASKED ABOUT THE BOOK.
John inquired about the book.

JOHN ASKED FOR THE BOOK.
John requested the book.

JOHN ASKED MARY A FAVOUR.
John begged Mary for a favour.

JOHN ASKED A FAVOUR OF MARY.
John begged Mary for a favour.

JOHN ASKED MARY TO GO HOME.
John begged Mary to go home.

JOHN ASKED TO GO HOME.
John desired to go home.

JOHN LOVES CHESS.
John enjoys chess.

JOHN LOVES TO BEAT BILL AT CHESS.
John enjoys defeating Bill at chess.

JOHN LOVES MARY.
John is in love with Mary.

JOHN LOVES RUNNING.
John likes to run.

JOHN LOVES RUNNING IN THE PARK.
John likes to run in the park.

JOHN ADMITTED THE TRUTH TO BILL.
John told Bill the truth.

JOHN ADMITTED DEFEAT.
John accepted defeat.

JOHN WAS ADMITTED TO THE HOUSE.
Someone let John into the house.

JOHN ADMITTED TO BILL THAT HE LOVES MARY.
John confessed to Bill that he is in love with Mary.

THE POLICEMAN WHO INTERROGATED THE CROOK ASKED MARY ABOUT THE BOOK.
The policeman who questioned the artful dodger questioned Mary about the book.

THE CROOK WHOM THE POLICEMAN INTERROGATED ADMITTED THE TRUTH.
The artful dodger whom the policeman questioned told the truth.

JOHN IS ANGRY.
John is cross.

JOHN IS ANGRY WITH HIS SON.
John is annoyed with his son.

JOHN IS ANGRY AT THE DECISION THAT BILL MADE.
The decision which Bill made makes John flush with anger.

JOHN IS ANGRY THAT BILL KILLED MARY'S MONKEY.
John is annoyed with Bill killing Mary's monkey.

JOHN IS ANGRY THAT BILL KILLED HIS MONKEY.
John is annoyed with Bill killing someone's monkey.

JOHN IS ANGRY THAT BILL'S SISTER'S MONKEY ATE THE BANANAS.
John is annoyed with the monkey of Bill's sister devouring the bananas.

BILL'S SISTER'S BOOK IS GREEN.
The colour of the book of Bill's sister is green.

THE CROOK IS GREEN.
The colour of the long thing is green.
The artful dodger is a novice.

THE BOY IS GREEN.
The boy is a novice.

MARY IS GREEN WITH ENVY.
Mary is envious.

THE BOY GREW.
The boy became bigger.

THE BOY GREW FLOWERS IN THE GARDEN.
The boy cultivated flowers in the garden.

THE BOY GREW SAD BECAUSE OF MARY.
Mary made the boy feel depressed.

JOHN IS EASY.
John feels relaxed.

JOHN IS EASY TO PLEASE.
It is not difficult for someone to make John happy.

JOHN IS EAGER.
John is impatient.

JOHN IS EAGER TO PLEASE.
John wants very much to make someone happy.

JOHN'S BIG FRIEND WHO HURT MARY IS AFRAID OF THE POLICEMAN.
The big friend of John who injured Mary fears the policeman.

JOHN'S BIG FRIEND WHO HURT MARY IS AFRAID OF BILL WHO LOVES HER.
The big friend of John who injured Mary fears Bill who is in love with her.

JOHN IS AFRAID TO TELL THE GIRL THAT HE LOVES THE TRUTH.
John doesn't want to tell the girl that he admires the truth.

John doesn't want to admit the truth to the girl with whom he is in love.

JOHN IS AFRAID THAT BILL LOVES MARY.
John suspects that Bill is in love with Mary.

JOHN IS AFRAID OF CALLING ON MARY.
John doesn't want to visit Mary.

JOHN CALLED MARY.
John shouted at Mary.

JOHN CALLED MARY TO FLY TO PARIS.
John summoned Mary to fly to Paris.

JOHN CALLED ON MARY TO ASK FOR THE BOOK.
John visited Mary in order to request the book.

JOHN CALLED AT THE CLUB.
John visited the meeting place.

JOHN CALLED BILL A FOOL.
John thought that Bill is a fool.

JOHN CALLED BILL A MONKEY.
John thought that Bill is a monkey.

JOHN CALLED BILL A TAXI.
John ordered a taxi for Bill.

JOHN DECIDED TO STOP CALLING ON MARY.
John made a decision to cease to visit Mary.

JOHN STOPPED TO CALL ON MARY.
John stopped in order to visit Mary.

JOHN STOPPED CALLING ON MARY.
John ceased to visit Mary.

TO CALL ON MARY, JOHN CALLED A TAXI.
John ordered a taxi in order to visit Mary.

THAT JOHN CALLED ON MARY WAS BAD.
It was not very good that John visited Mary.

JOHN HAVING ASKED FOR THE BOOK, I DECIDED TO CALL ON MARY.
After John requested the book, I made a decision to visit Mary.

CALLING ON MARY, JOHN DECIDED TO ASK FOR THE BOOK.
John made a decision to request the book while John was visiting Mary.

JOHN KILLED MARY, STRIKING HER WITH THE CLUB.
John hit Mary with the missile weapon and killed her.

KISSING AUNTS CAN BE BORING.
The aunts who kiss someone can be dull.

(The act of) kissing the aunts can be not particularly exciting.

SHOOTING ELEPHANTS CAN BE DANGEROUS.
(The act of) shooting the elephants can be hazardous.

THE WORKERS CALLED OFF THE STRIKE AT THE LAST MOMENT.
The workers cancelled the strike at the last moment.

THE CIRCUMSTANCES CALL FOR AN APOLOGY.
The circumstances demand an apology.

IBM CALLED OFF THEIR PROJECT FOR A NEW BIG COMPUTER.
IBM cancelled their project for a big new computer.

THE POLICEMAN INTERROGATED THE GREEN CROOK.
The policeman questioned the novice artful dodger.

IT WAS RUMOURED THAT THE PROPOSALS CALLED FORTH HOSTILE CRITICISM.
Someone spread rumours that the proposals provoked unfriendly criticism.

THE PORTERS WERE CALLING OUT THE NAMES OF THE STATIONS AT WHICH THE TRAIN STOPS.
The porters were shouting loudly the names of the stations where the train stops.

ONE BIG NEWSPAPER CALLED FOR THE PUNISHMENT OF THE WORKERS WHO WERE RESPONSIBLE FOR THE STRIKE.
One big newspaper demanded punishment of the workers who caused the strike.

I SAW THE MAN IN THE PARK WITH THE TELESCOPE.
With the telescope, and in the park, I saw the man.

In the park which had the telescope, I saw the man.
With the telescope, I saw the man who was in the park.
I saw the man who had the telescope, and who was in the park.
I saw the man who was in the park which had the telescope.

Bibliography.

Abbreviations:

IJCAI3:
Advanced Papers for the
Third International Joint Conference On Artificial Intelligence,
Stanford, USA, 1973.

TINLAP1:
SCHANK,R. and NASH-WEBBER,B. (eds.)
Theoretical Issues In Natural Language Processing,
MIT, Cambridge, Mass., 1975.

Aho72
AHO,A. and ULLMAN,J.
The Theory Of Parsing, Translation And Compiling,
Prentice Hall, Englewood Cliffs, N.J., 1972.

Bach68
BACH,E. and HARMS,R. (eds.)
Universals In Linguistic Theory,
Holt, Rhinehart and Winston, New York, 1968.

Cater79
CATER, A.W.S.
An Analysis And Inference System For English,
Forthcoming Ph.D.Thesis,
Computer Laboratory, Cambridge University, 1979.

Charniak76
CHARNIAK,E. and WILKS,Y.
Computational Semantics,
North-Holland, Amsterdam, 1976.

Chomsky57
CHOMSKY,N.
Syntactic Structures,
Mouton and Co., The Hague, 1957.

Chomsky65
CHOMSKY,N.
Aspects Of The Theory Of Syntax,
MIT Press, Cambridge, Mass., 1965.

Colby73
COLBY,K. and SCHANK,R. (eds.)
Computer Models Of Thought And Language,
Freeman and Co., San Francisco, 1973.

Fillmore68
FILLMORE,C.
The Case For Case,
in [Bach68].

Goldman75
GOLDMAN,N.
Conceptual Generation,
in [Schank75].

Harris77
HARRIS,L.
User Oriented Data Base Query With The ROBOT Natural Language Query System,
International Journal of Man-Machine Studies, vol.9, pp 697-713, 1977.

Hayes77
HAYES,P.
Some Association Based Techniques For Lexical Disambiguation By Machine,
Ph.D. Thesis, University of Rochester, New York, 1977.

Hendrix78
HENDRIX,G. et al.
Developing A Natural Language Interface To Complex Data,
ACM Transactions on Data Base Systems, vol.3, pp 105-147, 1978.

Herskovitz73
HERSKOVITZ,A.
The Generation Of French From A Semantic Representation,
Computer Science Department, Memo AIM-212, Stanford University, 1973.

Kaplan72
KAPLAN,R.
Augmented Transition Networks As Psychological Models Of Sentence Comprehension,
Artificial Intelligence, vol.3, pp 77-100, 1972.

Katz64
KATZ,J. and FODOR,J.
The Structure Of A Semantic Theory,
in "The Structure Of Language", J.Fodor and J.Katz (eds.)
Prentice Hall, Englewood Cliffs, N.J., 1964.

Kay73
KAY,M.
The MIND System,
in [Rustin73].

Klein65
KLEIN,S.
Automatic Paraphrasing In Sentence Format,
Mechanical Translation, vol.8, p.68, 1965.

Kuno62
KUNO,C. and OETTINGER,A.
Multiple Path Syntactic Analyser,
in "Information Processing", C.M.Popplewaite (ed.)
North-Holland, Amsterdam, 1962.

Lakoff72
LAKOFF,G.
On Generative Semantics,
in "Semantics", O.Steinberg and L.Jakobovits (eds.)
Cambridge University Press, Cambridge, Mass., 1972.

Lyons68
LYONS,J.
Introduction To Theoretical Linguistics,
Cambridge University Press, Cambridge, 1968.

Lyons77
LYONS,J.
Semantics,
Cambridge University Press, Cambridge, 1977.

McCarthy62
McCARTHY et al.
LISP 1.5 Programmer's Manual,
MIT Press, Cambridge, Mass., 1962.

McCawley68
McCAWLEY,J.
The Role Of Semantics In A Grammar,
in [Bach68].

Marcus75
MARCUS,M.
Diagnosis As A Notion Of A Grammar
in TINLAP1, 1975.

Minsky75
MINSKY,M.
A Framework For Representing Knowledge,
in "The Psychology of Computer Vision", P.Winston (ed.),
McGraw Hill, New York, 1975.

Mylopoulos76
MYLOPOULOS,J. et al.
TORUS: A Step Towards Bridging The Gap Between Data Bases And
The Casual User,
Information Systems, vol.2, pp 49-64, 1976.

Quirk72
QUIRK,J. et al.
Grammar Of Contemporary English,
Longman, London, 1972.

Rieger75
RIEGER,C.
Conceptual Memory And Inference,
in [Schank75].

Riesbeck74
RIESBECK,C.
Computational Understanding: Analysis Of Sentences And Context,
Ph.D.Thesis, Computer Science Department, Stanford University,
1974.

Ritchie77
RITCHIE,G.
Computer Modelling Of English Grammar,
Ph.D. Thesis, Edinburgh University Press, 1977.

Ritchie78
RITCHIE,G.
Augmented Transition Network Grammars And Semantic Processing,
Report No.CSR-20-78, Department of Computer Science, University
of Edinburgh, 1978

Rustin73
RUSTIN,R. (ed.)
Natural Language Processing,
Algorithmic Press, New York, 1973.

Sager73
SAGER,N.
The String Parser For Scientific Literature,
in [Rustin73].

Schank73
SCHANK,R.
Identification Of Conceptualisations Underlying Natural
Language,
in [Colby73].

Schank75
SCHANK,R. et al.
Conceptual Information Processing,
North-Holland, Amsterdam, 1975.

Simmons72
SIMMONS,R. and SLOCUM,J.
Generating English Discourse From Semantic Networks,
Communications of the ACM, vol.15, pp 891-905, 1972.

Simmons73
SIMMONS,R.
Semantic Networks: Their Computation And Use For Understanding
English Sentences,
in [Colby73].

Sparck Jones65
SPARCK JONES,K.
Semantic Classification,
Cambridge Natural Language Research Unit, Cambridge, 1965.

Waltz78
WALTZ,D.
An English Language Question Answering System For A Large
Relational Database,
Communications of the ACM, vol.21, pp 526-539, 1978.

Wilks68
WILKS,Y.
Computable Semantic Derivations,
Systems Development Corporation, Santa Monica, California,
1968.

Wilks72
WILKS,Y.
Grammar, Meaning And The Machine Analysis Of Language,
Routledge, London, 1972.

Wilks72b
WILKS,Y.
Understanding Without Proofs,
in IJCAI3, 1972.

Wilks73
WILKS,Y.
Natural Language Inference,
Computer Science Department, Memo AIM-211, Stanford University,
1973.

Wilks73b
WILKS,Y.
An Artificial Intelligence Approach To Machine Translation,
in [Colby73].

Wilks75
WILKS,Y.
A Preferential, Pattern-Seeking Semantics For natural Language
Inference,
Artificial Intelligence, vol.6, pp 53-74, 1975.

Wilks75b
WILKS,Y.
An Intelligent Analyser And Understander Of English,
Communications of the ACM, vol.18, pp 264-274, 1975.

Wilks76
WILKS,Y.
Processing Case,
American Journal of Computational Linguistics, Microfiche 56,
1976.

Wilks76b
WILKS,Y.
Parsing English, I and II,
in [Charniak76].

Wilks76c
WILKS,Y.
Natural Language Understanding Systems Within The AI Paradigm: A Survey And Some Comparisons,
American Journal Of Computational Linguistics, Microfiche 40, 1976.

Wilks77
WILKS,Y.
Good And Bad Arguments About Semantic Primitives,
Communication and Cognition, vol.10, pp 181-221, 1977.

Wilks78
WILKS,Y.
Making Preferences More Active,
Artificial Intelligence, vol.11, pp 197-223, 1978.

Winograd72
WINOGRAD,T.
Understanding Natural Language,
Edinburgh University Press, 1972.

Wood67
WOOD,F.
English Prepositional Idioms,
Macmillan Press, London, 1967.

Woods70
WOODS,W.
Transition Network Grammars For Natural Language Analysis,
Communications of the ACM, vol.13, pp 591-606, 1970.

Woods72
WOODS,W.
The Lunar Sciences Natural Language Information System,
Final Report, Bolt, Beranek and Newman, Cambridge, Mass., 1972.

Woods73
WOODS,W.
An Experimental Parsing System For Transition Network Grammars,
in [Rustin73].

Woods75
WOODS,W.
Syntax, Semantics And Speech,
AI Report 27, Bolt, beranek and Newman, Cambridge, Mass., 1975.

Woods76
WOODS,W.
Speech Understanding Systems,
Final Report, vol.4, "Syntax and Semantics",
Bolt, Beranek and Newmanm, Cambridge, Mass., 1976.