Report No.

UNIVERSITY OF BRISTOL

DEPARTMENT OF ENGINEERING MATHEMATICS

THE COMPUTERISED UNDERSTANDING OF NATURAL LANGUAGE

by

P.C.Baldwin and S.W.B.Brown

Date: MARCH 1983

SUMMARY

Several aspects of Understanding Natural Language have been practised or studied in the Project, from Pattern Matching through Parsing to Translation to a computer language. The programming language used was MAC-LISP on a Honeywell computer with a MULTICS operating system. The problem of Understanding Natural Language has been approached from an Artificial Intelligence point of view whilst also encompassing much linguistic theory. A historical account of the work done in this field to date is also included, although due to the keen interest and research in this subject, this might soon be out of date.

We have written three programs CAREER, GENERAL PARSER, and QUESSIE in LISP which respectively, hold conversations with a user, parse English sentences, and convert English questions to FRIL (a computer language).

AKNOWLEDGEMENT

We wish to thank our Project Supervisor, Mr J. F. Baldwin, for his guidance.

+ CONTENTS +

INTRODUCTION

The aim of the Project was to write a program in LISP to understand Natural Language; and to write a second program to translate the Natural Language to FRIL (an Intelligent Knowledge Base language).

So, what is Natural Language? Natural Languages are those used by people in the course of their daily affairs, for example, English and French. Users of Natural Language can express a broad range of ideas to others through them.

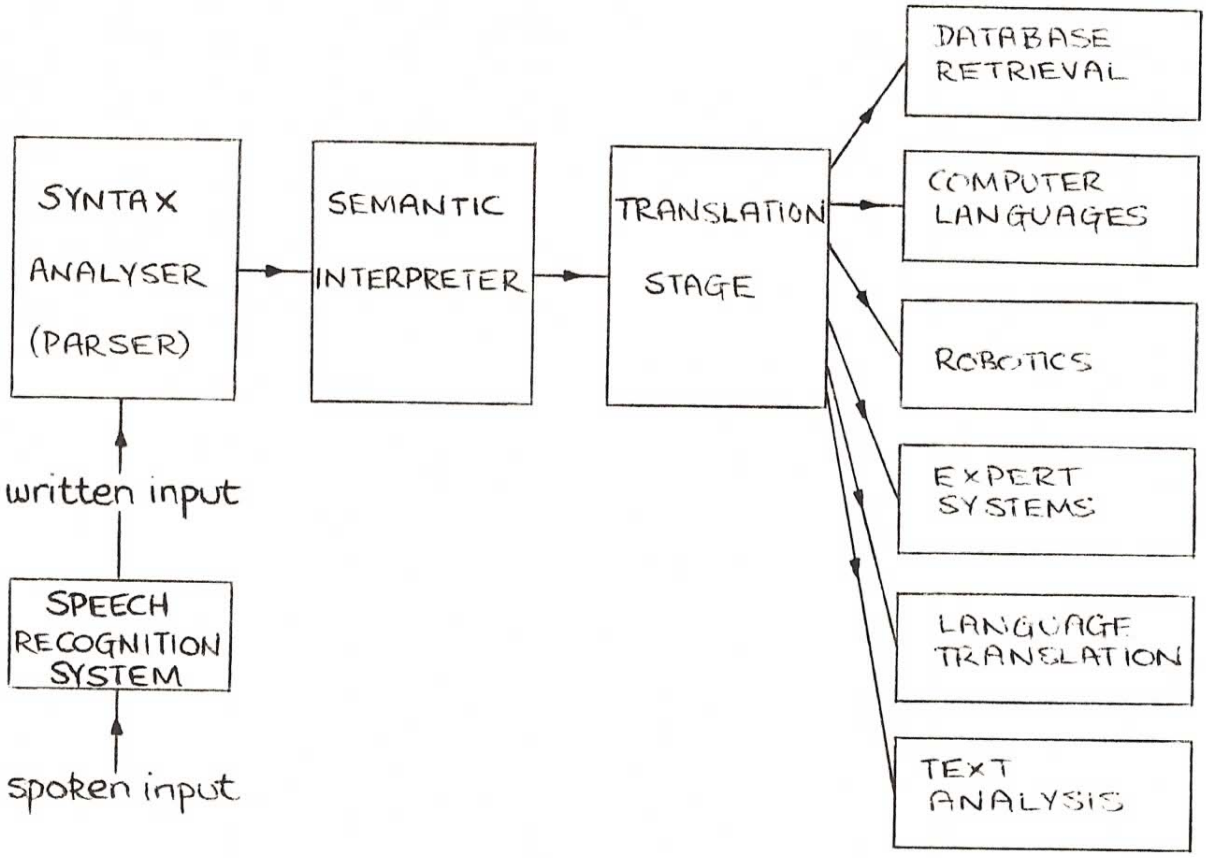


Fig.1 Simplified model of the Natural Language Processing System

The model shown in Fig1 is simplified by the distinction of processing stages which do in fact coincide to some extent. In our work we have taken a written input into a syntax analyser (known as a parser). A spoken input would have to be preprocessed by a Speech Recognition System and these are still in their infancy.

So the parser accepts a typed input and this is then analysed syntactically (into its constituent nouns and verbs etc). Once parsed, the Natural Language input in the reality of our system is sent straight to the translation stage.

The translation component converts the Natural Languge to a form suitable for any of the example uses. In our case this involves conversion to FRIL. For example:

"Who studies Maths" is converted to,
WHICH (X STUDY (X Y) AND WHERE (Y =MATHS)).

This FRIL query can then be directly fed into a LISP implementation of FRIL and the answer is given as a table of names of those people who study Maths.

Semantics is the study of meaning, so whereas an object is syntactically a common noun, semantically it is merely an image in our mind. As such it is difficult to analyse semantics logically, and we for our part, have not attempted to use any distinct semantic stage. Although there is some measure of semantics inherent in both the parser and the translation stage.

For our needs, syntax analysis has proved adequate, but no comprehensive process for understanding Natural Language would be complete without semantics. Indeed the failure to fully grasp semantics computationally is one of the current (and perhaps eternal) obstacles to pseudo-human processing of Natural Language.

There are, of course, many more application areas than those listed. Of those mentioned, language translation refers to translating between Natural Languages. This was one of the earliest research areas, particularly in the translation of Russian to English and vice-versa (needless to say this was heavily funded by the US government). Text Analysis involves the screening of, say newsprint, to locate any grammatical errors before going to press.

1. ARTIFICIAL INTELLIGENCE

Historical Background

A.I. in the field of computer science was pioneered by Alan Turing in 1950 when he proposed the question 'Can machines think ?'. His imitation game known as the Turing Test is now one of the definitive standards in the A.I. field. The term 'A.I.' and the language LISP (standing for List Processing) were both invented by John McCarthy.

In this thesis, reference is made to the 'intelligence' of machines which can evoke strong objections. It is not in our brief to philosophise on this point but it is worth noting that the words 'intelligence' and 'thinking' referred to computers should become increasingly widely accepted as the flexibility of people's thoughts broaden.

However one theorem suggests that: 'once some mental function is programmed, people soon cease to consider it as "real thinking". So the goal of A.I. is always that which has not yet been programmed'.

Progress in A.I. has been concentrated on the whole in the following areas:-

(1) game playing - e.g. chess
(2) theorem proving
(3) symbolic manipulation of mathematical expressions
(4) learning - e.g. Expert Systems
(5) creating original (or random) thoughts and art
(6) vision - pattern analysis
(7) hearing - speech recognition
(8) mechanical translation of natural language
(9) generating natural language
(10) understanding natural language.

The last four areas and indeed this thesis are all sited within the Natural Language field. It is widely accepted that this field will be at the forefront of computer science in the 1990's as 5th Generation computers are developed. The Japanese have already allocated $750m to their 5th Generation project. They intend using PROLOG as the basis of their A.I. applications. The Japanese have also claimed that they will, within a decade, be able to recognise and understand at least 90% of spoken input regardless of the speaker or speakers. This represents a huge advance on the present state of development and there is good reason to suggest that they are over-optimistic.

In Britain, the Alvey Report has urged the government to invest money in Information Technology. At present Marconi have produced a system which successfully recognises two-second word strings of spoken English. Logica's LOGOS system recognises up to 2000 different words in real time using parallel-processing; this won the computer trade's Technical Prize in 1982 and represents the best system on the market in this country at the moment. Certainly, in computer science, the problem of how to input and output information in a more 'user-friendly' fashion is at the forefront of research.

Early attempts at Natural Language processing were concerned with finding small sets of general rules for understanding language. This is contrasted with more recent programs (1970 to the present) which have been more concerned with finding a large number of specific rules to cover a given domain.

Another fundemental difference between the early programs and the later ones was the way in which computer technology had changed. In 1960 there was little support from the computer. Primary memory was typically a few thousand words. Today a user may enjoy a virtual address space of over a billion words with a five-fold increase in access speed as well as much greater primary memory. Computer languages were less sophisticated and in 1960 LISP was just being developed.

Computer Systems for Natural Language Understanding

Harvard Syntax Analyser (Kuno & Oettinger 1962)
BASEBALL (Green 1963)
ELIZA (Weizenbaum 1966)
PARRY (Colby 1974)
PHLIQA1 (Scha 1976)

EPISTLE (Miller,Heidorn,Jensen 1981)
SHRDLU (Winograd 1972)
LUNAR (Woods 1973)
GUS (Bobrow 1977)

Machine Translation

The first computational work on Natural Language was done in pursuit of machine translation, an effort heavily supported in the 1950's and early 1960's by the US government. The naive word-by-word dictionary look-up was supported by special rules for certain problems. However by the mid 1960's it was concluded that the high level of funding was not justified considering the limited achievement and prospect. An example of the limitations of this method is the translation from English to Russian of:

'The spirit is willing but the flesh is weak', which when translated back to English can read:

'The wine is agreeable, but the meat is spoiled'.

Theory Motivated Systems

One of the best known early projects was the Harvard Syntax Analyser which was a top-down, backtracking, context-free parser.

Question Answering Systems

From the mid 1960's to the mid 1970's, work was centred on systems that integrated syntax, semantics, and reasoning to carry out question answering tasks.

BASEBALL, for instance, answered questions on the month, day, place, teams and scores of each US League baseball game in one year.

Pattern Matching

This technique found most applications in systems where the computer poses the questions and the human user answers. This topic is discussed more fully later. The two best known examples of this technique are ELIZA and PARRY.

Database Retrieval Systems

In the last 5 years there has been much work on limited systems to handle complex syntax and deductions. PHLIQA1 and REQUEST both act as front-ends for commercial databases.

Text Analysis

This is a either a non-interactive application which involves the syntactic processing of bodies of text, or it can be an interactive technique like IBM's EPISTLE which aids letter-writers with spelling and grammar.

ATN Parsers

These are syntax analysing systems using Augmented Transition Networks - the basis for the GENERAL PARSER system described later. Previous work in this field includes Terry Winograd's SHRDLU.

SHRDLU was an integrated system to simulate a simple robot, allowing a person to give commands, state facts, and ask questions concerning a series of actions being displayed on a CRT screen. The PROGRAMMER parsing system was designed for use in SHRDLU; it was written in LISP and follows the ATN formalism fairly closely though not exactly.

LUNAR is a question answering system which uses an ATN parser. LUNAR responds to questions based on data about the mineral samples brought back from the moon, using a large database provided by NASA. This too was written in LISP. Parsing is done left-to-right, top-down using backtracking with ordered arcs (some versions use parallel-tracking).

So this has outlined the history of computerised Natural Language Understanding up to the present day. The work of Terry Winograd sticks out as particularly influential and his views on Natural Language have formed the foundation for our own work.

2. NATURAL LANGUAGE

Natural Language is the set of languages both spoken and written that are commonly used for human communication.

The word 'Natural' emphasises a contrast with artificial languages. Artificial languages are those designed to be highly expressive over a a limited range of ideas, such as programming languages which have been written so as to be analysed easily by computers.

In contrast, Natural Languages have an enormous range of expressive power. This communication process involves simultaneous related activities by producer and comprehender.

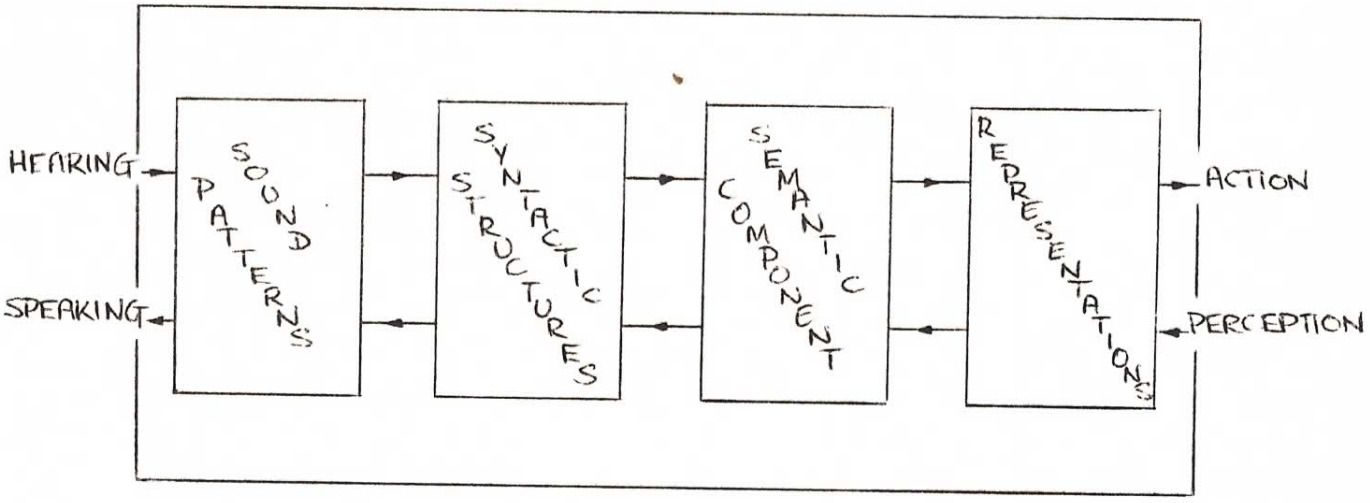


Fig.2 Model of processing done by the language user

For clarity, the various aspects of language analysis: syntax, semantics, discourse, and knowledge representation are considered separately but they are, in fact, interdependent.

VIEWING LANGUAGE AS A KNOWLEDGE BASED PROCEDURE

The study of the mental processes involved in language draws heavily on concepts developed in A.I. The unconsciousness with which we use language gives the illusion that language communication is a simple process not involving much knowledge.

When learning a new language, we are aware of needing to know more information such as meanings and grammar. And yet we are still unaware of the degree of underlying knowledge that is required for intelligent communication. Humans regard it as native ability.

We choose to view language as a 'communication process based on knowledge'. In any situation of language use, the producer and comprehender are processing information, making use of their knowledge of the language and of the topics of conversation. The task as linguists is to understand the organisation of these processes. Indeed, much of the understanding of the mental processes involved has come as a by-product of A.I. research into computerising them. The computer shares with the human mind the ability to manipulate symbols and carry out complex processes that include making decisions on the basis of stored knowledge.

SYNTAX

Syntax is the part of linguistics that deals with how the words of a language are arranged into phrases and how components like prefixes and and suffices are combined to make words.

Human language taken as a whole is infinite. We can produce sentences that we have never heard or spoken before and they can be understood by others for whom they are totally new. Our freedom to create novel utterances operates within a framework of grammar which puts strong constraints on the patterns that are used in the language. Certainly no human language is syntax-free.

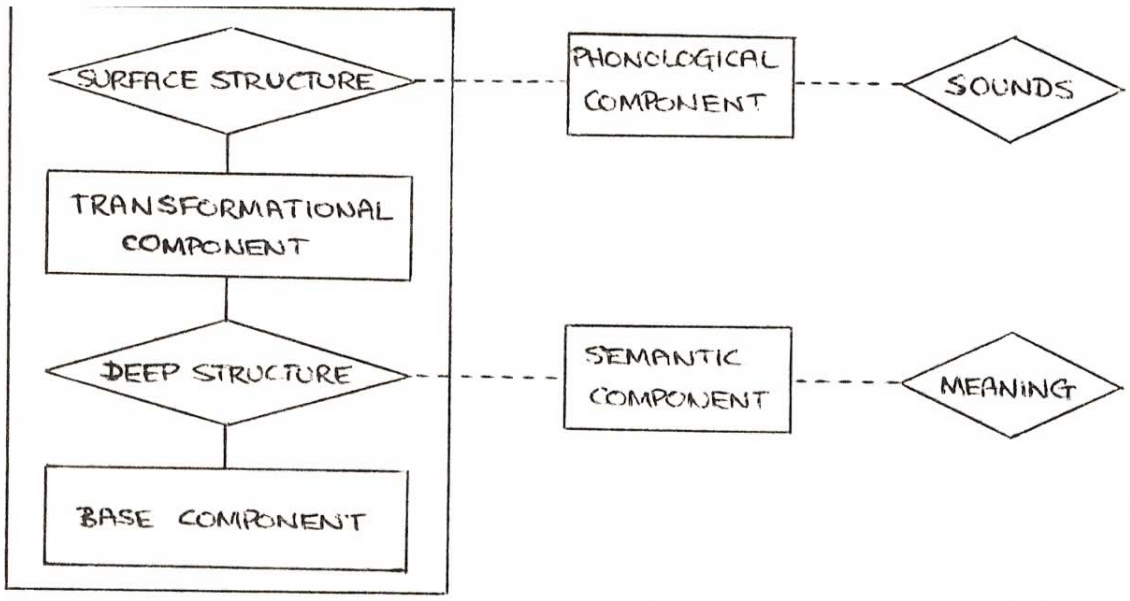


Fig.3 Syntactic Component in the Human Language Capacity

In fig.3 the so-called transformational model of language is illustrated. There are two levels of structure, deep structure and surface structure, which are related to one another by the grammar (transformational component). The base component creates the deep structure out of morphemes (units of meaning) using a context-free grammar. This can be meaningfully interpreted by the semantic component.

Whereas the surface structure is interpreted by the phonological component to produce sounds or, in reverse, sounds are interpreted by the phonological component to produce a surface structure.

TRADITIONAL WORD CLASS DEFINITIONS

Parts of Speech

(1) **Determiner (Article)**

(i) definite : the, this, that....
(ii) indefinite : a, an

(2) **Noun**

A noun represents a 'person, place, thing, or concept'. They can take endings to represent plural (cat --> cats) or possessive (cat --> cat's) forms.

(i) common nouns : all nouns other than proper
(ii) proper nouns : names of people, places
(iii) pronouns : sustitute for a noun. There are various types:-
(a) personal : (he, she, it ...)
(b) demonstrative : (this, that ...)
(c) possessive : (my, your ...)
(d) relative : (who, which, that ...) They can introduce subordinate clauses.

(3) **Adjective**

They modify nouns (the big cat) or act as complements (she seems sad).

(i) comparative : applies to two nouns (he is heavier than me)
(ii) superlative : applies to three or more nouns (he is the heaviest man)

(4) **Verb**

A verb signals the performance of an action, the occurrence of an event, or the presence

tense (present, future, imperfect, past, plu-perfect, conditional, present-participle, imperative). Most of these endings are regular, e.g. present-participle (-ing), past tense (-ed), past-participle (-en). Some verbs, especially the more common ones, have irregular forms. For example - ring, rang, rung.

(i) Transitivity

(a) transitive verbs require an object (I like ice-cream)
(b) bi-transitive verbs require two objects (Give him the book)
(c) intransitive verbs require no object (She is sitting)

(ii) Auxiliary Verbs (will, can, has, is ...)

These verbs precede a main verb in a phrase, as in 'He must have been working hard'.

(iii) Modal Verbs (can, may, must, should ...)

They form a subclass of auxiliary verbs. They confer a measure of doubt on the main verb.

(iv) Gerunds

These are verbs in present-participle form which are also used as nouns, as in 'Laughing makes you live longer'.

(v) Imperatives

These are commands, as in 'List all the students in the Eng Maths Dept.'

(5) **Adverbs**

Modify verbs. They have comparative and superlative forms like adjectives.

(6) **Prepositions**

They relate a noun to the rest of the sentence. They are a small closed class (in, or, until, by, at ...)

(7) **Conjunctives**

They connect major parts of a sentence, or link two or more nouns or adjectives or verbs etc. Their treatment is a major issue of syntax. The simplest conjunctions form a small closed class (and, but, or ...).

Clauses

(1) Main Clause : the backbone of the sentence. It often makes a simple sentence on its own.

(2) Subordinate Clause : often introduced by a relative pronoun. They add meaning to the main noun or verb.

(3) Conditional Clause : e.g. 'If you do not behave then I will be angry'
'When cats sit on mats they are happy'

Active/Passive

When the subject is performing the action, the verb is said to be in the 'active voice'; e.g. 'Jack built the house'.

When the subject is suffering the action, the verb is in the 'passive voice'; e.g. 'The house was built by Jack'.

Questions

Questions are introduced by questioners (who, what, which ...) or by the inversion of the auxiliary verb with the subject, as in 'Does Jack build houses?'.

These traditional word class definitions correspond to the surface structure of fig.3. They are not adequate for developing a comprehensive formal account of syntax, however they provide the terminology for the work on the GENERAL PARSER in Section 7.

SEMANTICS

The role of semantics in language analysis is to relate symbols to concepts. The effect of semantic processing should be to establish in the hearer the meaning of what is heard. In simple cases, such as highly restricted domains of discourse, semantic processing may not be very complex. In the normal use of language, however, semantics requires a great deal of computation. And, in fact, the semantics that human language users deal with effortlessly, significantly outstrip any computational analysis devised so far.

There is a measure of semantic interpretation inherent in parsing and also in the application of Natural Language, as in translation to FRIL. We have not explored the semantic element of the Natural Language problem. It would, needless to say, make the system more powerful in terms of resolving ambiguities.

3. THE DIFFICULTIES INVOLVED IN THE COMPREHENSIVE UNDERSTANDING OF NATURAL LANGUAGE

(1) Size of the problem

There are approximately 10^{20} different English sentences 20 words long. The size is such that it must be handled recursively to break the problem down into manageable sub-problems.

(2) Ambiguity

This is the most troublesome difficulty with Natural Language. It is common to all human languages and particularly so to the English language. Ambiguity comes in three main varieties.

(i) Ambiguity in word meaning.

This form of ambiguity can be further subdivided into ambiguity which can be resolved by (a) parsing (syntax analysis), and (b) semantic analysis, and (c) irresolvable.

Examples:-

(a) 'The students attended lectures'. "Lectures" can be a noun or a verb. However, only a noun is syntactically valid in that position in the sentence, so parsing the phrase will resolve which is the correct interpretation of the ambiguous word "lecture".

(b) 'Kick a ball'. The two meanings of "ball" are both nouns. This cannot be correctly interpreted by parsing, however the sense (i.e. the semantics) of the sentence can resolve the proper meaning.

(c) 'Swallow flies'. This phrase is equally valid both as a command to eat insects or as a statement that a bird is flying. Neither syntax or semantics can divine the intended meaning. To cope with this level of ambiguity, humans learn to ask the speaker to clarify their message. It would, of course, be possible to program the computer to do likewise.

(ii) Ambiguity in inference.

Example: 'The soldiers shot the prisoners and I saw several fall/reload.

Both "fall" and "reload" are verbs and yet "fall" refers to the prisoners and "reload" to the soldiers. Again semantics would have to be used to unravel this syntactic ambiguity. Thankfully as far as we are concerned, this type of ambiguity and the ambiguity shown by the example in (i)(b) form a smaller class than those typified by (i)(a). However they are still considerable problems.

(iii) Ambiguity in emphasis.

This is the most subtle form of ambiguity as it depends on the tonal inflection of the speaker. For instance, in the example 'The old men and women': are both the men and the women old? Or just the men? Indeed will computers ever be able to resolve this problem?

(3) Irregularities

The verb 'ring' is irregular (viz ring, rang, rung) and the adjective 'good' (good, better, best) are both irregular. These irregular forms must be individually defined in the dictionary.

(4) Commands, Questions, Passive Voice

These three constructions can each be recognised and successfully analysed by parsing.

(5) Idioms

E.g. 'The more the merrier'. This phrase defies normal syntax. Such idioms can be handled by listing each of the likely ones as patterns which can be matched against the input before the sentence is parsed.

4. PATTERN MATCHING

Intelligent recognition of normal English phrases was first made possible by pattern matching programs such as Joseph Weizenbaum's ELIZA (DOCTOR) and Kenneth Colby's PARRY. The former is supposed to simulate a psychiatrist using 'non-directive therapy', the latter to simulate the belief structure of a paranoid. Both programs rely on pseudo-intelligence created by matching stored word phrases to the input of a human user and replying with prefabricated responses. We have written our own program CAREER which uses exactly the same principles. It prompts its victims to talk about their interests.

The aim in writing CAREER was to achieve a convincing illusion of intelligence, particularly in the use of Natural Language. No attempt is made to understand either the syntax or semantics of the user's statements. As such, the program will never be able to offer thoughtful advice but it may succeed in prompting the user into a worthwhile discussion. The dialogue is fuelled by the user reading more into the machine's sentences than actually underlies them. Weizenbaum explained it as follows, "most men don't understand computers at all and cannot bring the same scepticism to bear on the conversation that they might to a professional magician on stage".

CAREER responses are made on the basis of matching templates containing key words to the user's input. For example, (+ I enjoy +) will match any sentence in which 'I enjoy' occurs. Each template can have several responses (which can be used sequentially to avoid repetition). The key words (or KEYS) have a priority attached to them so that whatever is considered to be the most important aspect of the user's input is responded to ahead of trivial features. So, for instance, swear words are searched for before 'I enjoy'.

Matching

The success of CAREER depends on an efficient MATCH routine. We based our MATCH on the Winston & Horn version which makes recursive calls to test each atom (word) in the input list.

Use of the symbol '+' allows strings of atoms to be inspected and ignored in the search for the KEY word. An amendment to the Winston & Horn program was necessary here, since if the KEY was first in the sentence, the MATCH would return 'nil' (i.e. a failure). This is rectified with a small insertion in the original program. The symbol '+R' before or after the KEY (word or words) in the template instructs MATCH to store the string of atoms before or after the KEY and to set the variable R equal to this string. Two new LISP functions, ATOMCAR and ATOMCDR, are defined to aid this process and they separate the '+' from the variable label.

The MATCH program is simple and has proved satisfactory for all the needs of CAREER. Some time lag is noticeable when long sentences are matched, as there are 25 calls to MATCH (regardless of recursive calls) if the input falls without a successful match to the bottom of the program. Certainly the time lag is not inconsistent with the pause for thought of a careers-advisor; and an instantaneous reply would positively detract from the illusion of machine intelligence.

```
(defun MATCH (p d)
   (cond ((and (null p) (null d)) t)
         ((or (null p) (null d)) nil)
         ((equal (car p) (car d))                          ;equality?
          (match (cdr p) (cdr d)))
         ((equal (car p) '+)                                ; + ignores atoms
          (cond ((match (cdr p) (cdr d)))
                ((equal (cadr p) (car d))
                 (cond ((null (cdr d)) t)                   ; KEY first?
                       (t (match (cddr p) (cdr d)))))
                ((match p (cdr d)))))
         ((equal (atomcar (car p)) '+)                      ; +atom puts atom equal
          (cond ((match (cdr p) (cdr d))                    ; to string of words
                 (set (atomcdr (car p)) (list (car d))) t)
                ((match p (cdr d))
                 (set (atomcdr (car p))
                 (cons (car d) (eval (atomcdr (car p))))) t)))))
```

1) CAREER : this function is the control program (TERPRI prints a blank line).

2) INTRO : asks for user's degree course and checks it against a list (called SUBJECTS) of common titles. The variable DEG is used to store the course title. If the user's input is not found in SUBJECTS then DEG takes the value 'your degree'. DEG is used later in qustions needing a retrospective element.

3) loop : heads the main iterative section of the program. The routine will loop until nil is returned and then the loop is escaped and control returns to PROG for the end of the program.

4) POINTER : takes values 1, 2, 3 or 4 and is incremented (mod 4) on each loop. POINTER allows a choice of 4 distinct responses to any one match. MOREP takes the value t or nil. This is used as a second pointer and it doubles the possible range of responses.

5) BADWORD : uses the same word-search as INTRO to look for disallowed words (listed in BADWORDP).

6) my --> your : this is included in the control program and it solves a common flaw of the early versions of CAREER, namely mistakes like this,
USER: I enjoy my work
CAREER: Why do you enjoy my work?
This problem is removed by changing all my's to your's at the outset.

7) YOU : the presence of 'you' in the user input is assumed to mean that the user is offering some opinion about the machine. So CAREER responds by putting the interview back on the simple question/answer track with a suitable question.

8) YES-NO : responds to 'yes', 'no', and 'not', and also a selection of uncertain answers 'maybe', 'sometimes', 'fairly'.

9) VERBS : responds to some common verbs. One of the responses uses the DEG variable defined in INTRO. The idea is to create an illusion that user input is being semantically analysed and stored.

10) I : picks up some of the verb phrases not caught by VERBS.

11) IT : reacts to the inexplicit 'it' by asking for more information.

12) MORE : acts as a safety net to all non-nil inputs not accepted by any of the specialist match routines. There is a range of 8 possible replies.

13) END : returns control to CAREER.

Example

This is an excerpt from a conversation between CAREER and a third-year Civil Engineer.

Q. hello, what course do you do?
A. civil engineering

Q. do you enjoy civil engineering?
A. yes

Q. for what reasons?
A. I like design

Q. oh that's good, why do you like design?
A. because it is creative

Q. why do you say it is creative?
A. bloody stupid question

... so insolent, you are the one who is unemployed!

1) In the above example, CAREER starts with a set question, 'hello, what course do you do?'. The answer is read and then WORD-SEARCH searches SUBJECTS for 'civil engineering'. It is found, so it returns true and the course is stored in DEG for later use (if necessary).

2) Then CAREER appends the prefabricated response 'do you enjoy' to 'civil engineering' and prints its response. The next input is 'yes'. This is matched with the template (+ yes +) and the POINTER chooses from a set of responses. In this case the POINTER is currently 2, so CAREER prints 'for what reasons?'.

3) The next input is 'I like design'. This is matched in VERBS with (+ like +L), and the variable L is set to 'design'. In this case there is only one choice of response and that is 'oh that's good, why do you like L'. This is then printed with the current value of L.

4) The reply from the user, 'because it is creative', is matched with (+ it +n). The POINTER is now 4, so the reply 'why do you say n' is chosen and expanded as 'why do you say it is creative'. Finally the input 'bloody' is picked up by BADWORD and a set reply is printed.

(A longer example of CAREER in action can be found in Appendix 2)

Limitations and Improvements

i) Early versions of CAREER had no POINTER system for multichoice responses. This is vital for a convincing conversation, and an obvious improvement to the current CAREER is to increase the number of choices.

ii) More templates and more specific templates will help the 'understanding' of the machine.

iii) Changing the priority of the templates by switching the order in the control program has the effect of softening or making the interview more aggressive. Since, if BADWORD is given a low priority, the machine will seem to tolerate a lot of abuse.

iv) More use of storing user input for later use will heighten the the illusion of 'thought'.

v) Each subroutine in CAREER could be given a separate pointer so as to reduce the possibility of the same response being reissued.

vi) The use of LISP property-values attached to particular questions under a variable, say the user's name, so as to create a personal history, might allow a constructive suggestion as to a career to be made.

Returning to the more celebrated pattern-matching program, ELIZA, it is interesting to note that, according to its creator Joseph Weizenbaum, some users found ELIZA's psychiatric therapy more to their liking than visiting a doctor. However the fact remains that such programs are offering only a very shallow level of understanding. This is amply demonstrated by the following conversation between the two computer programs ELIZA and PARRY. It is clear that each machine treats the other's words at face value and no more.

PARRY: I don't understand your motives.
ELIZA: Why don't you understand my motives?

PARRY: Shouldn't I be?
ELIZA: Why do you say that just now?

PARRY: That's just the way it is.
EIIZA: Suppose that's just the way it wasn't?

PARRY: So?
ELIZA: I am not sure I understand you fully.

PARRY: You harp on the same questions too much.
ELIZA: What does that similarity suggest to you?

PARRY: I don't get you.

It is clear that the dialogue never becomes a conversation of substance and indeed proved more and more circular. Once the imagination (or gullibility?) of the human is removed, the system collapses. However that does not make this technique trivial; on the contrary, it can be quite powerful, and producing canned responses to expected questions is not unknown amongst humans. However if the level of true understanding is to improve, then we must turn to some linguistic theory.

5. CHOMSKY'S HIERARCHY OF GRAMMAR

Evolution of Linguistic Science

Current linguistic science can trace its origins back to the Sanskrit grammarians of 2000 years ago. More recently, the science took tangible form in the post-Darwin 19th Century, as linguists attempted to generalise about all languages, and to relate different languages to each other.

This 'comparative philology' was superceded by 'structural linguistics which concentrated on a single language and analysed the language as a chemist might, searching for empirical rules. This approach was popularised by Leonard Bloomfield's book "Language" in 1933.

In the past 25 years a new theory has gained prominence, due to the work of Noam Chomsky, beginning with "Syntactic Structures" in 1957. He shifted the focus of analysis from observable behaviours to the intuitions of native speakers. He argued that structural analysis did not capture the essential creativity of human language. He also stressed that understanding language should be viewed as a 'knowledge-based' process.

The Chomsky Hierarchy

Chomsky observed that the size of the language space required recursive study if it was to be made sensible with a limited knowledge base. He then proposed a hierarchy of grammars which define various degrees of freedom for a language.

Name	Description	Examples	Model
0	recursively enumerable	all Natural Languages	Turing Machine
1	context sensitive	Natural Language subset	ATN
2	context free	computer languages	RTN
3	regular finite	highly restricted languages	TN

The four grammars are essentially a subset of the one higher, i.e.

Type3 < Type2 < Type1 < Type0

The grammars are restricted by only allowing certain production rules (governing word order).

Recursively Enumerable (Type0)

The most general type which requires that 'there exists a mechanical procedure by which each sentence of a given language can be enumerated in finite time'. This would need a so-called 'Turing Machine' to understand every detail. A Turing Machine can, by definition, solve any problem for which there is an algorithm. Such a machine does not exist as yet. Type0 grammar covers all Natural Language.

Context Sensitive (Type1)

It is called context sensitive as each word in a sentence is dependent on the words around it. The generality of Type0 is restricted by production rules (introduced in the next section). This is the grammar on which the GENERAL PARSER is based. It can be modelled for computational applications, and for clarity, by an Augmented Transition Network (ATN).

Context Free (Type2)

This group of grammars have a further restriction on the production rules. Most computer languages and their compilers are based on this grammar. They can be modelled by Recursive Transition Networks (RTN), which are a less powerful version of the ATN.

Context free grammar is a formalism that has widely used as a basis for computer systems which manipulate computer languages or a limited Natural Language domain. It is used because of its simplicity and clarity. It provides a simple way of setting up correspondences between the knowledge structures and the structures recognised in the sentence. When parsing an ambiguous Natural Language phrase, context free grammar based techniques require a large amount of either backtracking or parallel-processing. This is obviously inefficient in both time and memory. There has been a lot of effort put into optimising backtracking and parallel processsing algorithms. However, in our case, by using a context sensitive grammar, the need for these search schemes is largely removed.

Regular (Type3)

Here production rules are very restricted. As far as Natural Language is concerned regular grammars are a purely academic case. Regular grammars do however provide the foundation for the more complex grammars and they are modelled by Transition Networks (TN).

TN's, RTN's, and ATN's will be explained more fully later.

A Rigorous Definition of Language

The mathematics of set theory can be used as a metalanguage to define each type of grammar. The more intuitive Backus Normal Form is introduced in Section 7, but here the aim is for a more rigorous account.

Notation

(i) For any set X, X^* is the set of all finite strings over the alphabet X.
(ii) X^+ is identical to X except the empty set is excluded.
(iii) => is a binary relation, where 'w => x' is equivalent to 'w directly derives x'.

Definitions

(1) Terminals, T : a finite alphabet of symbols or groups of symbols, combined to form texts of the language.

(2) Non-Terminals, N : a finite set of variables which take values of groups of symbols. So a non-terminal represents a sub-phrase of a whole sentence. The sets T and N must be disjoint, i.e. $(N \cap T)^*$ is empty.

(3) Start Symbol, S : S is a member of N and is used for naming the whole sentence.

(4) Productions, P : a set of production rules which indicate which transformations are legal. Each production is a pair of strings, say Q and R, each of which can contain members of the sets T and/or N.
i.e. (Q,R) is equivalent to 'replacing Q by R is allowable'. It is often written 'Q --> R' where (a) Q can be empty, and (b) P must contain at least one non-terminal.

Given these four elements of a grammar G, we can define the language L(G) produced by G(T,N,S,P) as those terminal sentences s, generated by successive implications, starting with the start symbol S.

$$L(G) = \{s \in T^* / S \Rightarrow s\}$$

The nature of these implications vary between grammars within a language.

(a) Context Sensitive

Productions of the form QAR --> QBR where $P,Q \subset (N \cup T)^*$, $A \in N$, $B \subset (N \cup T)^+$

i.e. A is replaced by B in the context of Q and R.

(b) Context Free

Productions of the form A --> B where $A \in N$, $B \subset (N \cup T)^*$

(c) Regular

Productions of the form A --> xB or A --> B where $A,B \in N$, $x \in T$

6. TRANSITION NETWORKS

These are a linguistic subset of 'State Transition Diagrams' which is a branch of mathematics that serves as a basis for much of the theory of computation.

Transition Networks are a way of organising processes both for the production and the analysis of linguistic structures. A Transition Network consists of a set of **nodes** (or states), connected by **arcs**. Each arc represents a **transition** between two states. Each transition corresponds to a test on the next item of the input returning true. A **JUMP** arc allows an arc to be traversed even if the **TEST** fails.

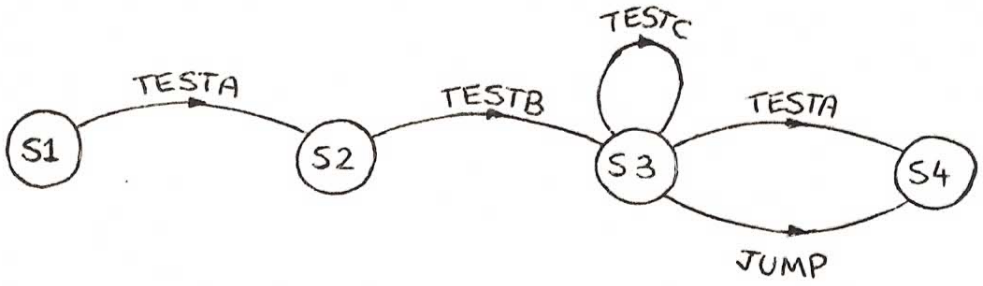


Fig.4 Example of a simple transition network

In the above example, TESTC provides a choice of arcs at node S3. The test is repeated until unsuccessful and TESTA returns true.

The power of Transition Networks lies not only in the simplicity of form, but in the emphasis of any repetitive nature of the input. In the case of Natural Language, algorithms that recognise sentences can be written using a Transition Network and a knowledge base consisting of a dictionary.

Recursive Transition Networks

An RTN will modularise the Transition Network and allow repeated calls to the same subnetwork. Each of these subnetworks can be **PUSH**ed to from the control network, and subnetworks can **PUSH** to each other building up recursive layers.

If a subnetwork is successfully traversed then it will **RETURN** to either the control network or whichever subnetwork it was pushed from.

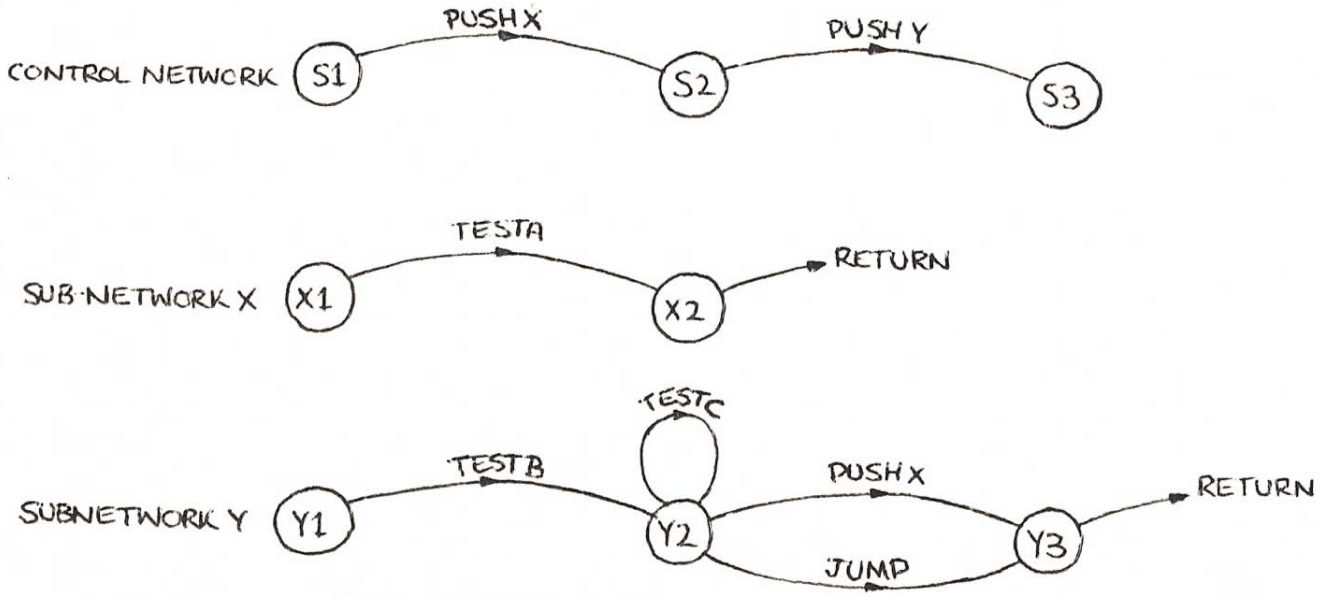


Fig.5 Example of a Recursive Transition Network

Augmented Transition Network

The augmentation of the RTN centre around the addition of **conditions** and **actions** associated with the arcs of a network. Conditions restrict the circumstances under which an arc is taken. Actions are used both to generate the structure (e.g. a tree structure) and to provide temporary information that can be used by conditions on other arcs.

Conditions and actions make use of **registers** associated with each node of a network. The contents of registers are called **features**.

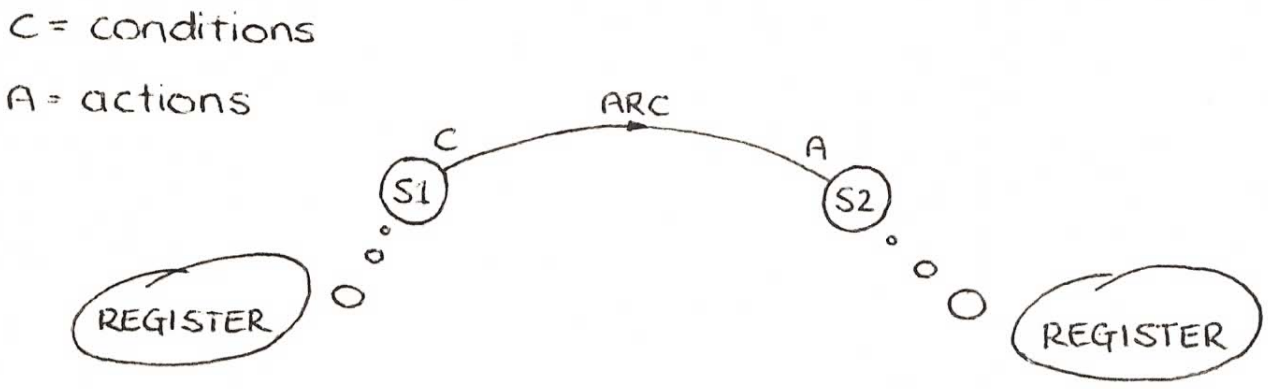


Fig.6 Augmentation of the network

A condition will consult the register of a node and will act on the features stored in that register. The power of the ATN lies in the ability of the conditions to be dependent not only on the special requirements of the current arc, but also on the entire structure that has been built up to that point. And it is the actions which create the features in the registers of each node.

7. GENERAL PARSER

A Parser determines whether a sentence conforms to the constraints of the syntax of a language, and, at the same time, builds up a representation of the syntactic structure. In designing this parser we have not tried to cater for full or definintive grammer, but have gone into the variety of problems that occur in language and solved the majority of them.

RECURSIVE TRANSITION-NODE NETWORK

Our final parser employs the 'top-down augmented transition matrix' method, this is a context sensitive method which makes implicit expectation of what will be found next in a phrase based on what has been found.

However, to start with, we tried to parse simple phrases using a transition-node network, as in figure 7.

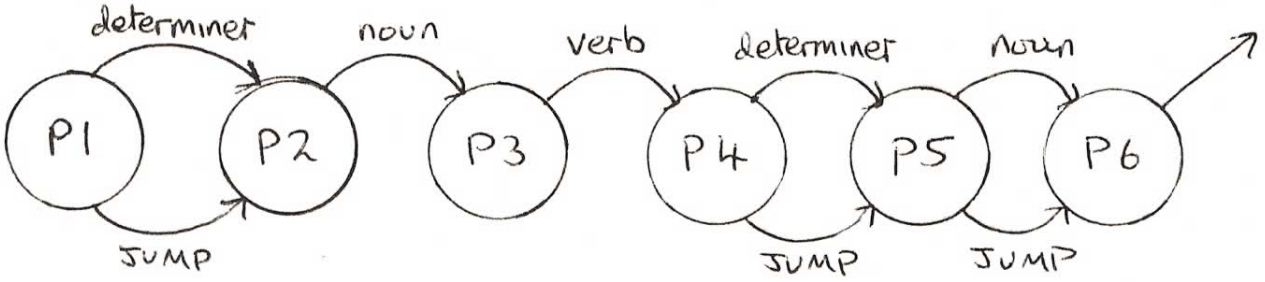


Fig. 7 Transition-node network to parse simple phrases

All the words which are intended to be used in the language are given features in the Dictionary, as explained in section 8, for example:

professor	-	singular noun
student	-	singular noun
lecture	-	singular noun
	or	root verb
attend	-	root verb
a	-	indefinite determiner
the	-	definite determiner
live	-	root verb intransitive

To illustrate how this transition-node network parses simple phrases, take the example:

the student attends lectures

The whole phrase enters at node P1 where the first word, 'the', is tested by the first arc for being a determiner. This is true so 'the' is taken off the phrase and the rest of the phrase 'student attends lectures' is passed onto the next node, P2. At P2 'student' is tested for being a noun, this is true so arc is traversed and 'attends lectures' is analysed at P3. Here 'attends' is tested for being a verb, this is true as it is a regular tensed form of the defined verb 'attend'. Because the generality of the regular endings of tensed verbs and plural nouns then these regular ending words don't have to be defined in the dictionary, this will be dealt with later on in section 8 in greater detail. At P4 'lectures' is first tested for being a determiner, this is not true, so the next arc is tried. This is a JUMP arc which is a means of tranversing between nodes without any words being taken off. So at node P5 'lectures' is tested for being a noun, it is a regular plural of 'lecture', so arc is taken to P6 and as there are no remaining words in the phrase then the parse is a success.

In the dictionary, 'lecture' is also defined as a verb so it will be tested according to the word order of the phrase. In the example:

the professor lectures the student

'lectures' is tested successfully for being a verb.

This network is quite sophisticated in that it will also accept phrases with intransitive verbs. These are verbs which cannot have nouns or determiners immediately after the verb, the following example will be a successfully parsed phrase:

the student lives

These simple phrases which the transition-node network parses can be split into smaller, separate phrases. This is illustrated with the Backus Normal form definition in fig. 8.

<PHRASE> ::= <NOUN-PHRASE> <VERB-PHRASE>

<NOUN-PHRASE> ::= <NOUN> | <DETERMINER> <NOUN>

<VERB-PHRASE> ::= <VERB> | <VERB> <NOUN-PHRASE>

<NOUN> ::= (professor student lecture ....)

<VERB> ::= (attend lecture .....)

<DETERMINER> ::= (a the .....)

Fig. 8 Backus Normal definition of a simple phrase

This definition says that:

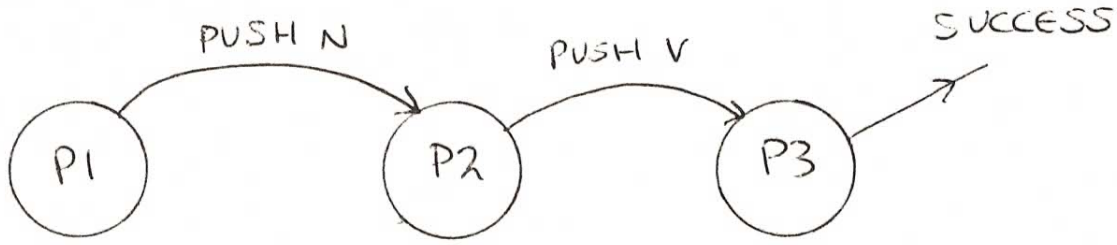
a PHRASE is a NOUN-PHRASE followed by a VERB-PHRASE

a NOUN-PHRASE is either a NOUN
or a DETERMINER followed by a NOUN

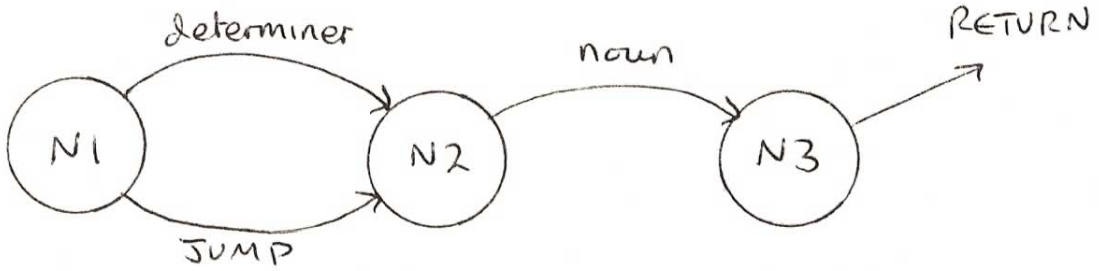
a VERB-PHRASE is either a VERB
or a VERB followed by a NOUN-PHRASE

So this phrase definition represents the transition-node network in figure 7. This in turn can be represented by a Recursive transition-node network in figure 9.

PHRASE, P



NOUN-PHRASE, N



VERB-PHRASE, V

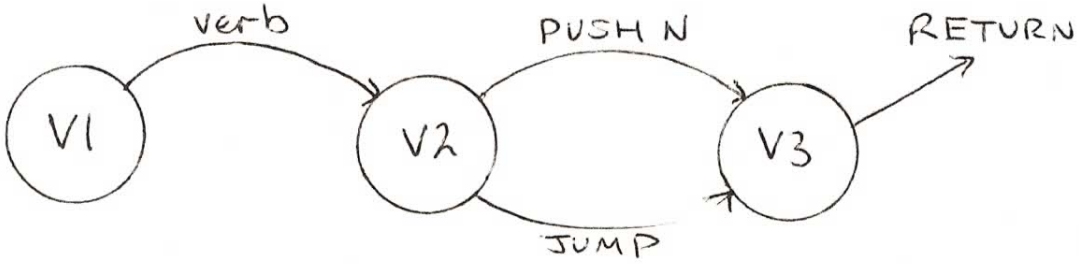


Fig. 9 Recursive transition-node network for a simple phrase

This RTN parses the same simple phrases, however it does it recursively by calling other networks from within networks. This can be seen in figure 9 where the verb-phrase calls the noun-phrase at node V2 (i.e. PUSH N). When a sub-network is successfully taken then a RETURN arc takes the rest of the phrase to be parsed back to the original network which PUSHed this successful sub-network, and hence the PUSH arc is traversed. However, if a sub-network fails to reach the RETURN arc then the original PUSH arc fails.

Using recursion enormously simplifies the design of a parser and makes building up a parser, for more complex grammer, much easier. The node-network system is a much better method than the Backus Normal form for representing a parser because the BNF has to define every combination of words that could occur in a phrase whereas the RTN caters for these choices with JUMP arcs and multiple arcs from the nodes.

AUGMENTED TRANSITION NETWORK (ATN)

An RTN can be augmented by placing conditions and actions on each arc that is taken. An ATN parse proceeds almost exactly like that for RTN grammers, except that the actions put contents into registers and conditions constrain the traversing of arcs.

When an arc is successfully taken then the associated action is carried out. This is a 'note-making' process which sets the registers to the feature of the arc just taken. In order for an arc to be taken its condition(s) must hold for the current state of parsing (along with the normal arc-applicability requirements, as for an RTN). A condition will consult the register features, stored by a previous action, and, depending on whether the condition is satisfied, the arc is taken.

The power of the ATN can be illustrated by this simple phrase example:

a students attends lectures

This will be successfully parsed by the RTN, even though 'a students' is not correct grammer. However with an ATN, when the determiner arc is taken, the determiner register is set to 'indefinite', as 'a' is an indefinite determiner. This means that only a singular noun can follow the determiner so a 'number' register is set to singular. At the next node 'students' is tested for being a noun, however the condition for traversing the arc is that 'students' has the same features as the previously set number register, i.e. is singular. This is not true, as 'students' is plural, so the arc fails and so the whole parse fails.

These registers also form relations which generate a syntactic-structure as the parsing proceeds.

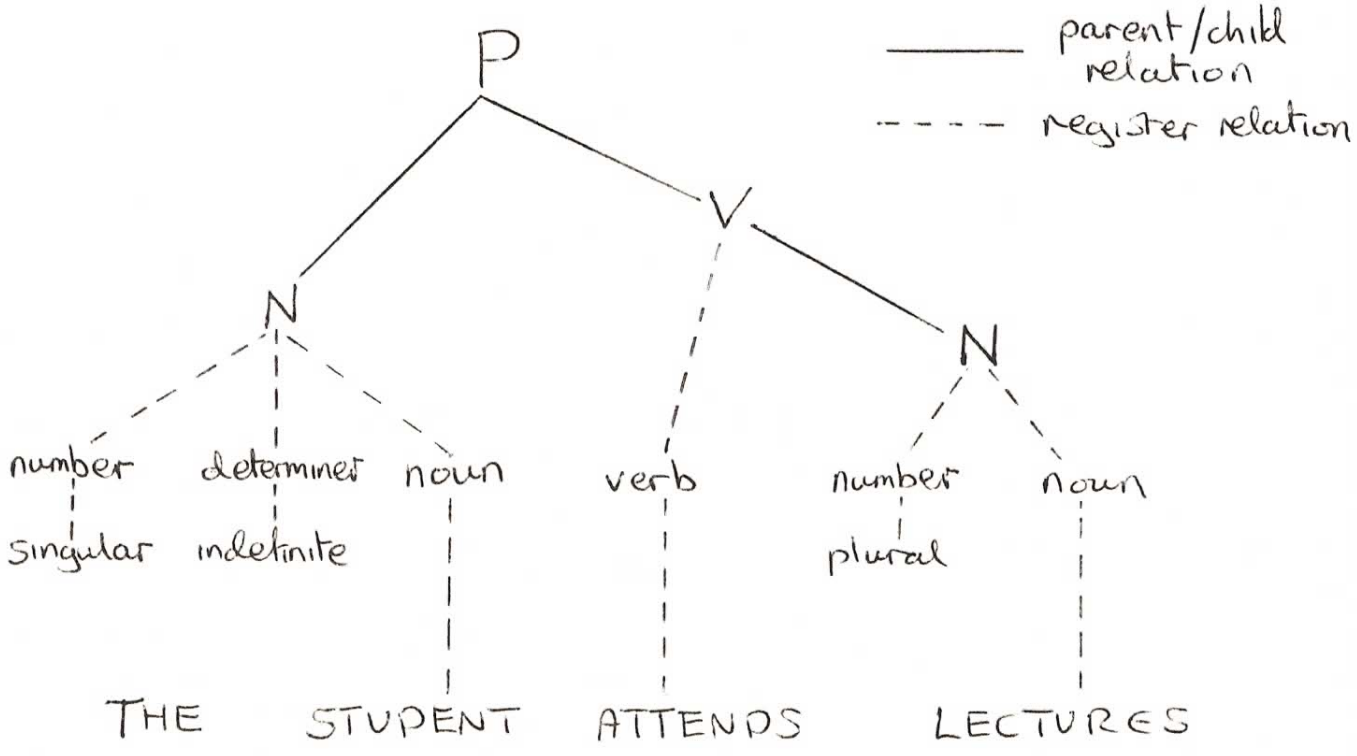


Fig. 10 The Syntactic-Structure of phrase **the student attends lectures**

The structure is built up as sub-networks and arcs are successfully taken. When a sub-network is successfully returned from, then a parent/child relation is formed between the sub-network and the calling network. When an arc (not a JUMP) is successfully traversed between nodes then a register relation stores the arc feature as a property of the sub-network.

The advantage of structure building is that it puts a phrase into a standard form, therefore more computing can then be done to process the structure.

Another way of representing the syntactic structure of a phrase is the sentence-diagram method that is often taught in schools, see figure 11.

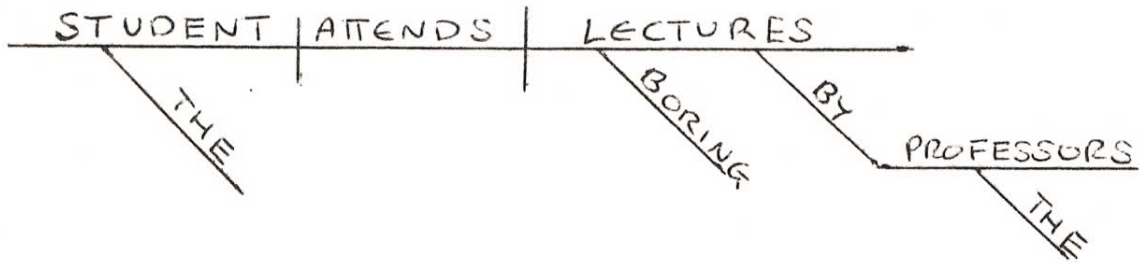


Fig. 11 Sentence-diagram for phrase **the student attends boring lectures by the professors**

However the previous syntactic structure, the Phonological structure of a phrase, has the advantage, over other syntactic structures, that structural units such as phrase, noun-phrase, noun, determiner, and so on are labeled. We will be using this phonological structure to represent phrases.

IMPROVING THE ATN

To cope with a wider variety of phrases, including those below, a much more comprehensive ATN has to be designed, see figure 12.

the students were in the lecture	(preposition)
the students were sitting happily in the lecture	(modal-verb adverb)
the three students were not in the lecture	(number negative)
if the students like Mathematics then they attend the lecture	(conditional-clause pronoun)
the professor is older than the students	(comparative)
the students who are lazy do not sit in the boring lectures	(subordinate clause)
Simon is the tallest	(proper-noun superlative)
attend the lectures	(imperative)

The imperative, or command, form is catered for in the phrase-network, P. At node P2 if the next phrase is not a noun-phrase, then a jump arc can be taken to P3 provided the next word is a root verb (or 'infinitive'). This is the test for an imperative phrase. An imperative flag is set when this JUMP arc is traversed, which is later made use of as a condition within the preposition network. This caters for imperative-phrases like:

come in sit down ...etc

In the verb-phrase, noun-phrase and adverb-phrase networks LOOP arcs occur, if one these arcs are successfully traversed then the remaining-words of the phrase are taken back to the same node and the next word is tested. Using this method, large verb, adjective, superlative, comparative or adverb sequences are parsed, for example:

would have been sitting	(verb)
big red juicy	(adjective)

In the verb-phrase, the last verb in a sequence of verbs is taken to be the main verb of the phrase. At node V5 in the verb-phrase network, the noun-phrase arc is the first test, however the condition for taking this arc is that the main verb is transitive. This prevents phrases like **I live a house** from being successfully parsed.

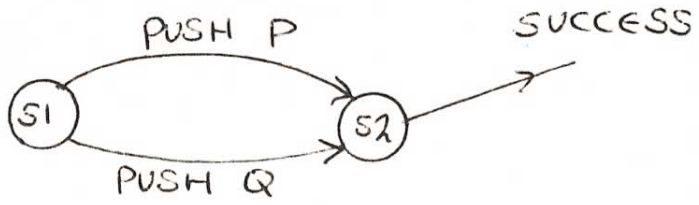
In noun-phrase network, the superlative, comparative or adjective test uses the same arc as this is easy to implement due to the regular endings of these words, this will be dealt with in greater detail later on in section 8.

-est	superlative of an adjective
-er	comparative of an adjective

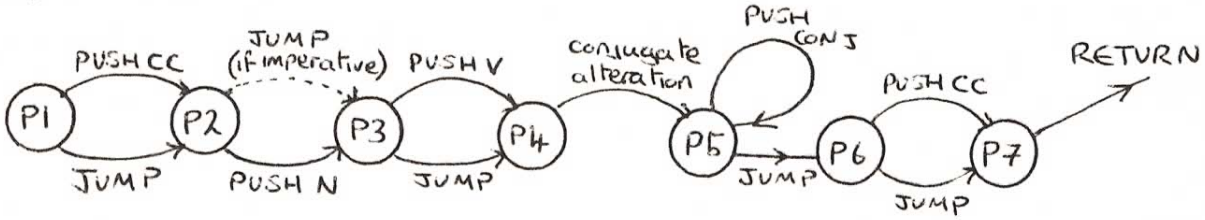
In the phrase: **the big car is the fastest**, 'the fastest' is a noun-phrase without a noun, this is catered for by including a JUMP arc at node N3 with the condition that a superlative, comparative or adjective has occured in the noun-phrase. A similar JUMP arc is in the adverb network.

An example of the syntactic structure produced by this improved ATN can be seen in figure 12.

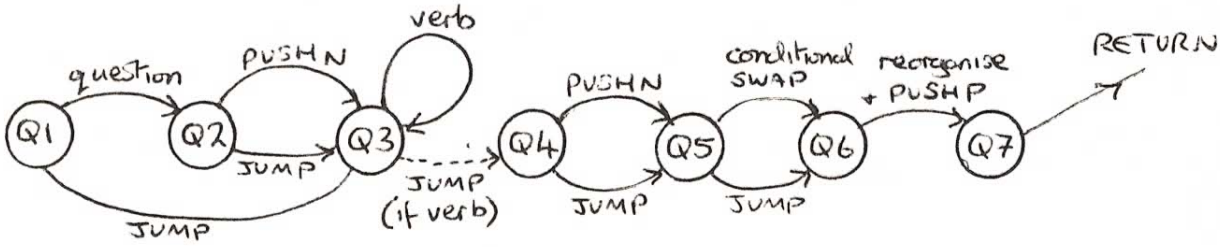
SENTENCE, S



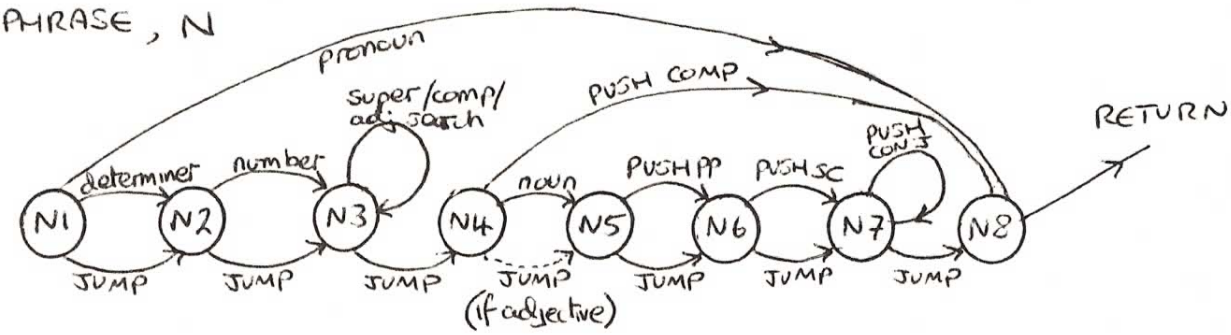
PHRASE, P



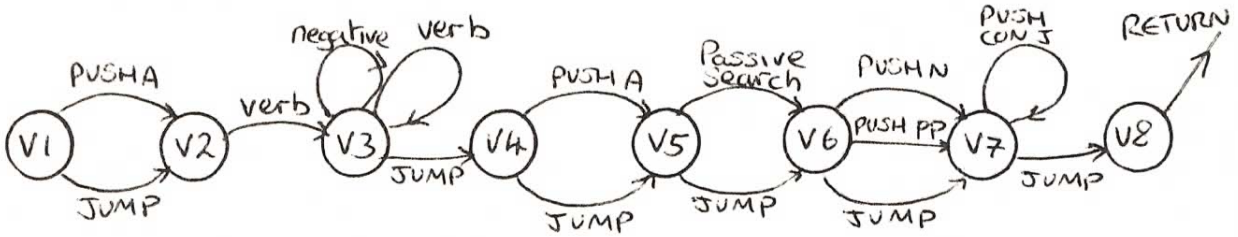
QUESTION, Q



NOUN-PHRASE, N



VERB-PHRASE, V



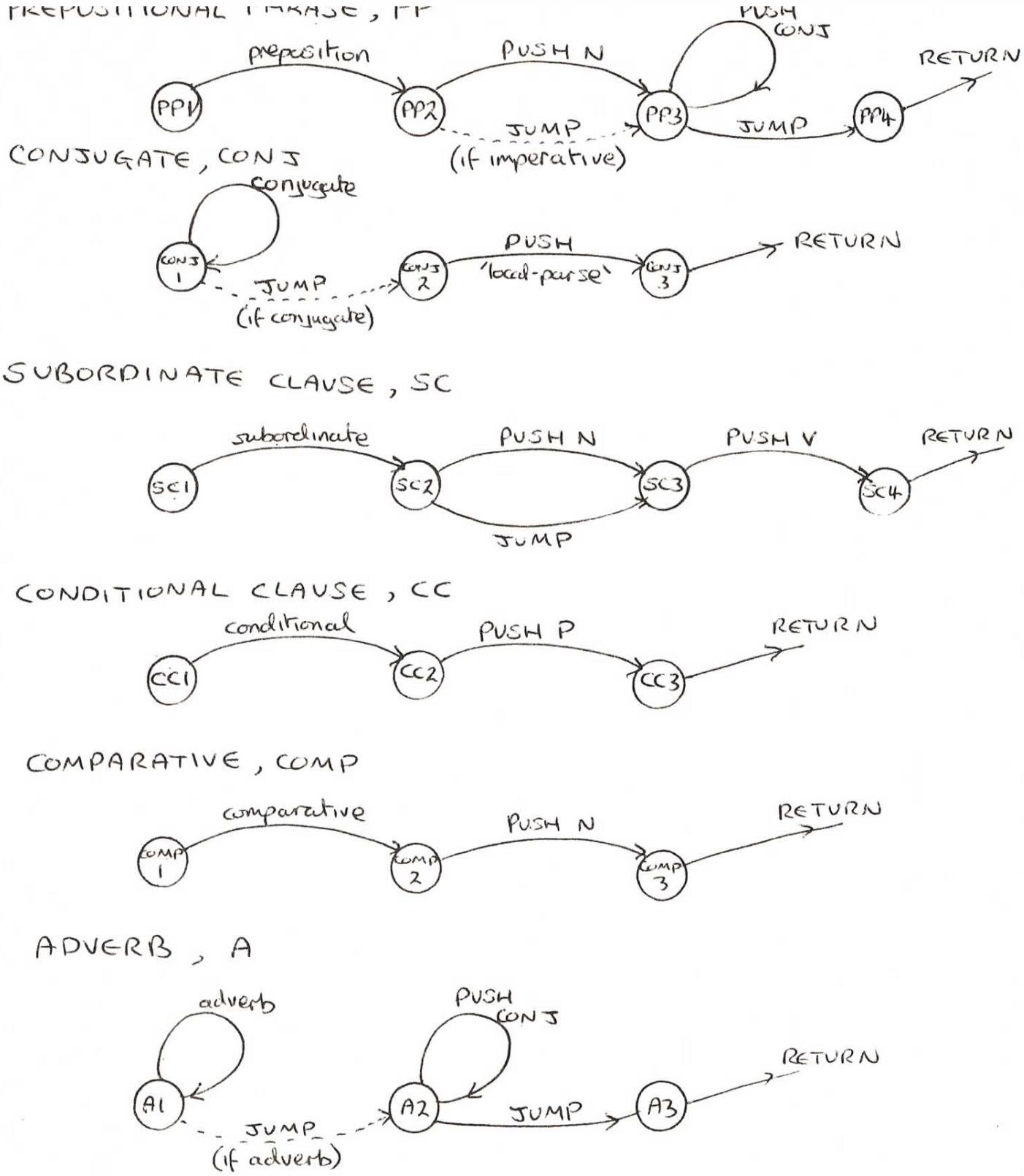


Fig. 12 Comprehensive ATN

To further improve this existing ATN, the following grammer is included:

a)	**Passive voice**	so far only dealt with active voice
b)	**Conjunctions**	or phrase-joiner
c)	**Question**	or Interrogative mood so far dealt with Declarative & Imperative moods

Passive Voice

Phrases like :-

the students attended the lectures

the lectures were attended by the students

have essentially the same semantics but have a different noun-phrase focus. The first phrase is in the active voice and the latter is in the passive voice. A passive voice essentially means that the order of the object and subject noun-phrases of an active phrase have been switched.

The method we used for identifying a passive phrase is :

(i) If the main verb is transitive and in past-participle tense and the next word is a preposition (usually 'by').

or (ii) If the main verb is intransitive and in past-participle tense and the next **two** words are prepositions (the last one is usually 'by').

Most participles of verbs have regular endings:

-ed or **-en**	for past-participles
-ing	for present-participles

So these participles are not defined as such in the dictionary, however irregular participles: **sat, gave, ate ... etc** have to be defined in the dictionary. The subject of regular endings will be dealt with in greater detail later on in section 8.

These passive voice conditions are implemented by including a new node in the verb-phrase network after V5, once the conditions are satisfied then this is stored as a 'voice' register relation in the verb-phrase. Once passive voice has been identified the phrase is parsed as usual, however no attempt has been made to alter the syntactic structure formed by a passive voice phrase.

Conjunctions

One of the problems with any top-down method, such as the one we have used, is the problem of conjunctions, for example :

he likes Engineering not Mathematics	(noun-phrase)
he came and sat in the lecture	(verb-phrase)
the students and professors but not the cleaners attended the lecture	(multiple noun-phrase)
the student was in the library or in a lecture	(preposition)
he sat proudly and happily in the car	(adverb)
Peter was in the lecture but Simon was in the pub	(phrase)
the car was big red and fast	(adjectives)

These phrases could produce a considerable amount of back-tracking and re-parsing, each time giving the same structure and being rejected for a larger structure. However this inefficiency can be overcome by keeping the struture of a successfully parsed phrase and then forming a parent/child relation with the conjunction-phrase. This is done using a conjuction sub-network and including an extra node in the phrase sub-networks, as seen in figure 13. The PUSH 'local-parse' between nodes CONJ2 & CONJ3 means PUSH the sub-network from which the conjunction network was called.

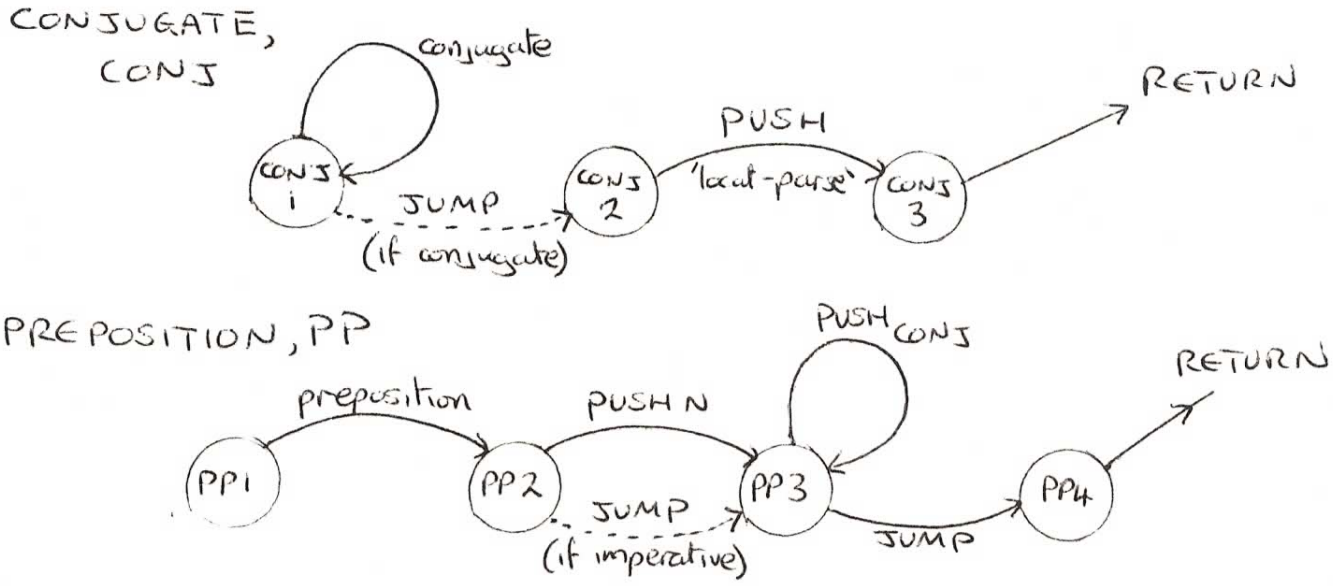


Fig. 13 Conjunction and complete preposition sub-networks

These additional PUSH CONJ arcs are also included in the phrase, noun-phrase, verb-phrase, and adverb sub-networks.

For example:

the student was in the library or in a lecture

The conjunction sub-network in this phrase is called whilst in the preposition sub-network, so 'local-parse' is set to preposition and the rest of the phrase is tested for being a preposition-phrase. If this is true, as in this example, then the parent/child relations are re-organised so that the two preposition-phrase structures are joined as children of the conjunction, which is, in turn, set to the child of the verb-phrase which originally called PP. This can be seen in figure 14. If the next phrase is not another PP, then PUSH 'local-parse' arc will fail, causing the CONJ sub-network to fail and hence a parent/child relation will be formed between the verb-phrase and the already parsed preposition-phrase.

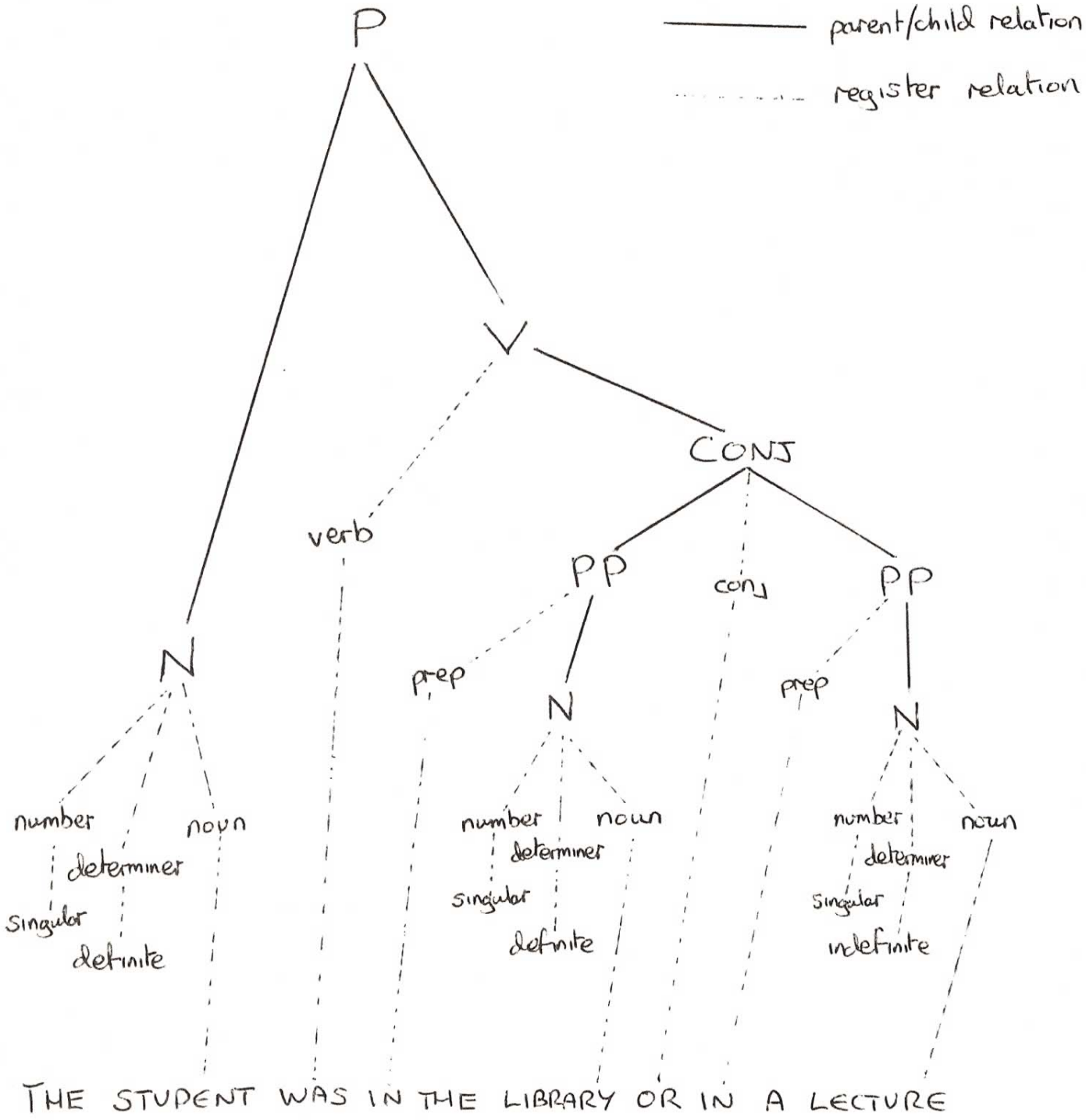


Fig. 14 Syntactic Structure of Conjunction
of **the student was in the library or a lecture**

A problem that can occur is that when the parser finds a conjunction, sometimes the wrong phrases are joined causing an unsuccessful parse. This can be illustrated by the following examples:

cats like milk and mice
cats like milk and mice like cheese

The first phrase will be correctly parsed using the existing conjunction sub-network, however the second phrase will parse exactly the same way leaving 'like cheese' as the remaining-words, hence will be an unsuccessful parse. A 'false' syntactic structure will also have been generated where the wrong phrases have been joined. To solve this problem we used a hold register. This register merely holds the contents of remaining-words when a conjunction network is called from within a noun-phrase network, in this example hold will contain 'and mice like cheese'. Another node is included in the PHRASE network which has associated conditions testing whether the hold register has been used and that the next word in the remaining-words is a verb. In this example this is true, so then the remaining-words are set to the contents of the hold register and the existing 'false' syntactic structure is re-organised. The parsing then proceeds in the PHRASE network and the conjunction sub-network is PUSHed again, however this time the 'local-parse' is a phrase, thus causing a successful parse and the correct are 'joined' in the syntactic structure parent/child relations.

If three or more phrases are joined in a phrase then a conjunctive structure as in figure 15 will be formed.

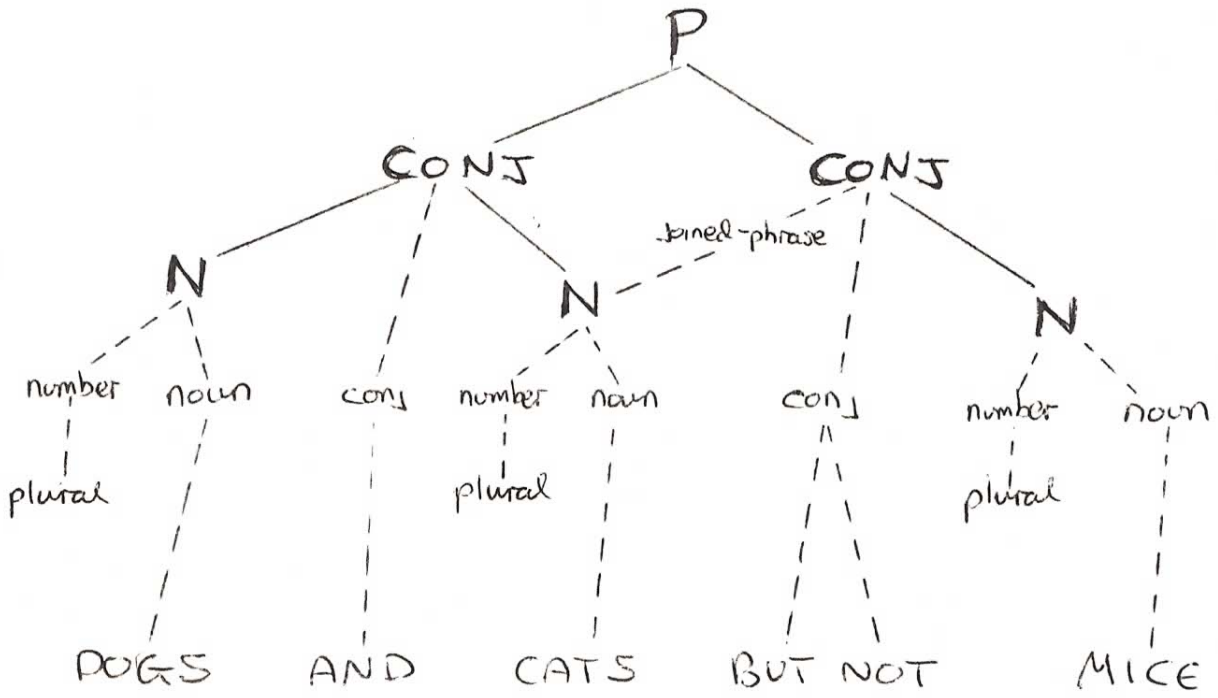


Fig. 15 Structure for multiple conjunction example **dogs and cats but not mice**

For the case of conjunctions in adjectives, we decided to deal with this within the noun-phrase, so all the adjectives are found and stored as a register relation of the noun-phrase. This is a case of joining words not phrases, so it is not worth using recursion as a search of just one word ahead is required.

Questions

The ATN in figure 16 will successfully parse the following questions :-

does Simon like lectures ?
who does Simon like ?
who likes Simon ?
which car is the fastest ?
who was lectured by the professor ?

So far the Moods of the sentences we have delt with are declarative and imperative, and instead of parsing 'questions' we will deal with the Interrogative Mood. These terms are used rather than the more obvious 'question', 'statement' and 'command' to indicate that the parser caters for the form of the sentence, not its use. This can be illustrated by the sentence **you are a student ?** spoken with a rising intonation is a question, although its form is declarative, while **am I a student !** is a statement whose form is interrogative.

PARSE-QUESTION, **Q**

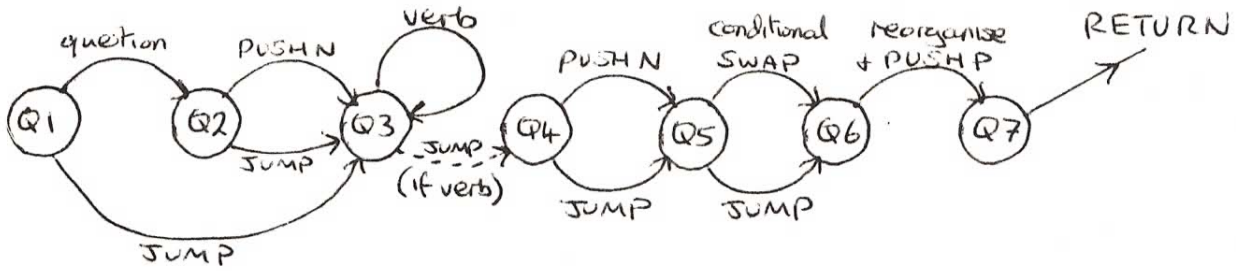


Fig. 16 Parse-question sub-network

Instead of designing a whole new ATN to cater for questions, the form of sentence is recognised as interrogative and then the sentence is re-organised into the declarative form and then parsed as such.

At node Q1 of the parse-question network the first word of the sentence is tested for being a questioner, for example **which, where, what, who, why, how ..etc.** if this is false then the question is tested for being a 'verb inversion' type as in:

does Simon like lectures ?

If the test arc at any of nodes Q2 Q3 or Q4 is successful then the words which were taken off due to the test are stored.

The successfully traversed noun-phrase between Q2 and Q3 is stored as the 'object', the noun-phrase between Q4 and Q5 as the 'subject' and the verb as the 'qverb' (say).

At Q5 the object and subject noun-phrases are swapped, if the question-word is a **wh-** type and there are no remaining-words left in the sentence, as in the example:

who likes Simon ?

At Q6, the sentence is then re-organised by appending:

subject + qverb + remaining-words of phrase + object

This sentence is now in the declarative form, and is parsed using the PHRASE ATN. If the Q6 --> Q7 arc succeeds then the 'question' is parsed successfully.

The subject and object noun-phrases have a default value 'dummy-qnoun' (say), which is defined in the dictionary as a noun, so that if a question has no subject or object noun-phrases then the sentence after re-organising will still be in a phrase form. For example the above example will be re-organised into:

dummy-qnoun likes Simon

which will be successfully parsed by the PHRASE ATN.

This technique of finding the subject and object of a question is used later on in section 9 on the interpreting of natural language for use as a FRIL interface.

COMPREHENSIVE SENTENCE PARESR

Now, with the interrogative, imperative and declarative moods catered for, a new network Parse-sentence is created which attempts to parse any given sentence, see figure 12.

An example of the syntactic structure generated by this ATN can be seen in figure 17.

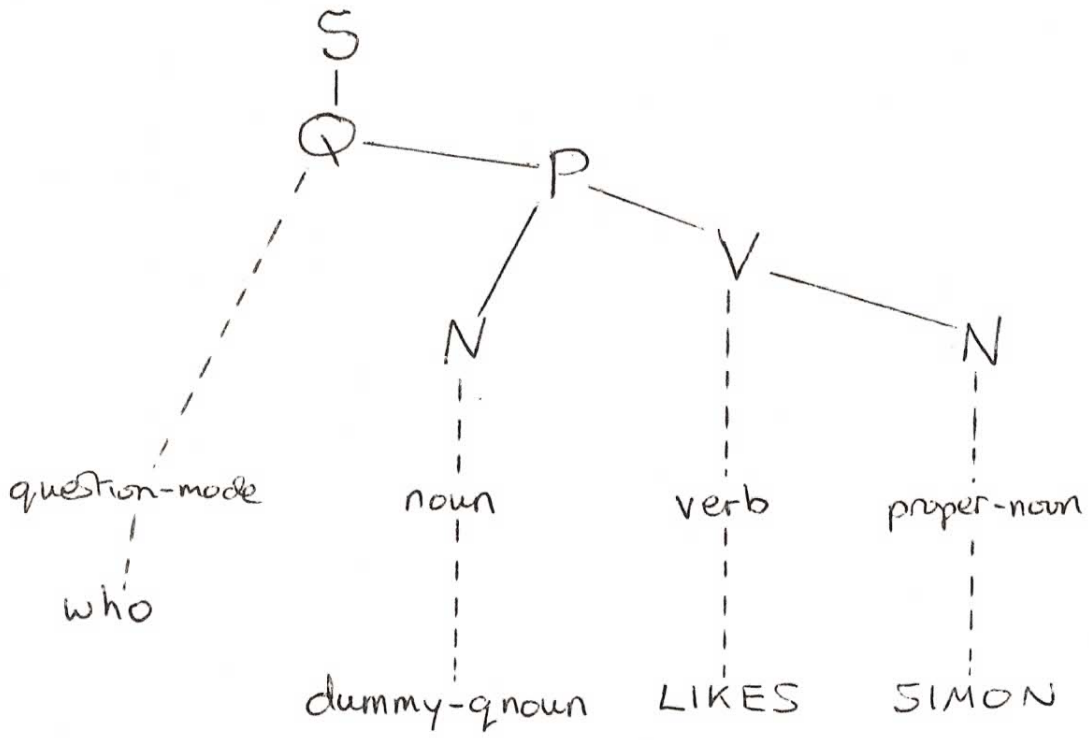


Fig. 17 Syntactic structure for example **who likes Simon**

FURTHER SYNTACTIC ISSUES

The method we chose to improve the ATN was to 'generate and create' a wide vari e t y of different phrases and, provided these had sufficiently general grammer, we tried to design the sub-network to cater for them. Using this method, the ATN in figure 17 was developed.

The same philosophy can be implemented to include the following grammer:

a) Bi-transitive verbs

we have dealt with intransitive and transitive verbs in the existing ATN, in that we can successfully parse phrases with or without object noun-phrases after the verb. However bi-transitive verbs (e.g. bring give take ...) can have two objects:

I gave the girl a rose

This can also occur in the 'dative' form:

I gave a rose to the girl

b) Ordinal Numbers -e.g. first third thirtieth ....

Cardinal numbers (one three thirty ...) have been catered for, and ordinal numbers can be treated in the same vein.

c) Possessive Pronouns -e.g. mine yours his ....

These can be dealt with similarly to the 'demonstrative pronoun' (this that these ....)

d) Pre-determiner -e.g. all half ....

As the name suggests, these can be tested for before the determiner in the noun-phrase.

e) Quantifier -e.g. each every some most many few ..

deal with similar to a determiner.

f) Describers

Adjective	the **happy blue** bird
Present-participle	the **laughing dancing** women
Past-participle	the **tired broken** student
Noun	the **steel kitchen** knife

These describers can be catered for in an ATN but the exact rules for selecting and ordering these elements are very subtle, they have to do with a kind of scale of 'inherentness' on a semantic level, and there is no good clear formalism for them.

However these describers can also be used as the complement of an attributive clause:

the bird was blue the women were dancing

g) Binders e.g. because so since until before while ..

These are connectives like conjunctives and sub-ordinates and can be treated as such in the ATN.

h) Embedded Phrases and Phrase modifiers
without an introducing relative pronoun (who which that . . .)
For example:
I went to the lecture **to see the professor**
the most exciting student **I will ever teach**
the project **they all wanted to read** was late
These phrase types could be handled using multiple entry points into the networks.

i) Adverbs
The type we've catered for is the 'Manner adverbs' (happily quickly . .) which are regular, however there is a very wide collection of adverbs. These have to be treated as 'special phrasal lexicon', because most of the adverbial phrases are unique combinations of irregular words:

Time and place	- no sooner had he said it . . . . .
	- a year ago . . . .
	- the way there
Intensifiers	- an almost millionaire
	- a sort of lecture
Alternatives	- only Simon understood it
	- even the professors understood
	- the students understood too
Negation	- I don't quite have any time
	- a not quite real experience
Frequency	- usually never sometimes probably . . . .
Reasoning	- therefore thus so however . . . .

j) Interjection e.g. Oh, Hey, Ouch, Well . . .

k) Questions
try and cater for more complex questions as in the following examples :
by what method did he come
how many times had he done it

Particularly in the noun-phrase sub-network, there are conditions which prevent incorrect syntax to be correctly parsed in the noun-phrase, when person and number disagree, hence the following examples will not be parsed:
a cars
a three cars
the three car

This method can be extendend to prevent parsing of the following incorrect syntax:

l) Verb Sequence

he is being had will had were gone

This seemingly random sequence of modal verbs can be prevented from being parsed as there is a formalism which defines the order a sequence of verbs can take:

(modal)	(have)	(be1)	(be2)	main-verb
will	have			gone
	has	been		writing
	had	been	being	corrected
will		be		arrivng
shall				arrive

The parentheses indicate that an element is optional, i.e. a clause must have a main-verb. Each element determines the form of the one following it according to theses rules:

If the element contains:	Then the following verb element is:
modal	Root
have	Past-participle
be1	Present-participle
be2	Past-participle

These rules can be implemented within the verb-phrase sub-network of the ATN.

II) Case agreement of pronouns

This can be solved by forming a case register for the pronouns with the choices: Objective (accusative), Subjective (nominative) and Possessive:

objective - her us them ...
subjective - he she I who ....
possessive - my your her ....

Conditions can then be placed on the traversing of arcs, thus preventing a pronoun case conflict.

III) Agreement of person and verb tense

he love ....
the students loves ...

By consulting the noun-phrase 'number' register and a verb-phrase 'tense' register then can conclude whether the noun-phrase, verb-phrase combination is feasible, preventing above examples being correctly parsed.

Another improvement could be the inclusion of an incorrect spelling test in the ATN. This could exploit the 'morphology' of a Natural Language to check that the right prefixes and suffixes have been used on words. This would be used 'hand in hand' with the regular-endings searches (see section 8).

So far the General Parser generates a surface (phonological) structure. Still more regularity could be found if the surface structure were transformed to another structure called the underlying structure (or 'deep' or 'conceptual' structure). The fundamental differences between surface structure and underlying structure is the word-order and the insertion and deletion of information. For example passive and active phrases have different surface structures, however they can have identical underlying structures.

An example of the insertion of 'understood' words between the surface structure and underlying structure can be seen in the following:

he waved (his hand) goodbye
the lecturer had been seen (somewhere)

The phrases in parentheses represent information that is 'understood' and that would appear in the underlying structure but not in the surface srructure. ATN parsers are capable of manipulating the underlying structure whilst parsing. Using this method, it is possible to deal with non-standard structures and reduce ambiguity of a phrase.

Ellipsis is one of the most persuasive phenomena in natural language - the omission of elements that would be expected in the full, syntactically correct form of the phrase, for example:

I gave Henri a cup of coffee and Simon _____ tea
I can do it better than you _____

The whole issue of ambiguity is very difficult to deal with syntactically without the use of semantics.

I saw the man on the hill with a telescope

This phrase is open to interpretation, however the ATN syntactic structure generation we have used is right-branching. A way this could be dealt with, is to form a 'canonical' (flattened) structure. This would probably require more than one parsing and would be greatly improved using parallel exploration of alternatives.

Another ambiguity problem is that of 'homonyms'. These are words which can belong to more than one word-class, for example :

(i) nouns & verbs - lecture swallow fly duck fish . .
(ii) nouns & adjectives - flat light firm blue . . .

In (i), because of the implicit expectation of word order sentences containing these words they are usually parsed correctly. However problems can occur in imperative and question phrases:

lecture the students
who lectures the students

In the above examples 'lecture' is parsed incorrectly.

In (ii), both noun and adjective occur within the same network. This causes a problem in that 'flat' (a noun, say) is tested in the noun-phrase network firstly for being an adjective. This is true and therefore the network will be successfully taken, however 'flat' has been stored in the register as an adjective.
These problems could be alleviated by more stringent conditions placed on the relevant nodes, causing minimal backtracking.

Therefore, in conclusion, the ATN can be modified to cater for as wide a variety of grammer as needed, however there will probably always be an 'exception to the rule', no matter how comprehensive the ATN.

8. ATN IMPLEMENTATION

The practical foundation for the GENERAL PARSER was the ATN **INTERPRETER** of Winston & Horn. We have taken their program (which is no more than skeletal) and adapted and added to it to meet our needs. The first requirement before writing the interpreter, was to decide on a suitable notation that captured the idea of nodes and transitions. Once the notation was invented, then the interpretive program was written to follow it one step at a time. The ATN interpreter, INTERPRET, follows a retained description and notational details are a matter of taste. Again we have largely followed the example of Winston & Horn.

The notation we have used is as follows:

```
(<NODE_NAME> (IF <CONDITION> ---> <NEW_NODE> AFTER <ACTION>))
```

IF, --->, and AFTER are included for clear representation and are never evaluated. CONDITIONS and ACTIONS are as described in the definition of ATN's. Within this syntax, PARSE-SENTENCE, PARSE-NOUN-GROUP, PARSE-VERB etc are defined.

INTERPRET

```
(DEFUN INTERPRET (NETWORK PARENT-NODE )
   (PROG (MODAL NEWSTATE  STATE-DESCRIPTION RULE HOLD THIS-NODE PARENTP)

         (SETQ MODAL NIL PARENTP PARENT-NODE)
         (SETQ HOLD REMAINING-WORDS)
         (SETQ THIS-NODE (GENNAME (CAR NETWORK)))
         (SETQ NETWORK (CDR NETWORK))
         (SETQ NEWSTATE (CAAR NETWORK))
            GET-STATE-DESCRIPTION
         (COND ((EQUAL NEWSTATE 'WIN) (GO WIN))
               ((EQUAL NEWSTATE 'LOSE) (GO LOSE))
               (T (SETQ STATE-DESCRIPTION (ASSOC NEWSTATE NETWORK))))
            TEST-TRANSITION-RULE
         (COND ((SETQ STATE-DESCRIPTION (CDR STATE-DESCRIPTION))
                (SETQ RULE (CAR STATE-DESCRIPTION)))
               (T (SETQ NEWSTATE 'LOSE)
                  (GO GET-STATE-DESCRIPTION)))
         (COND ((EVAL (CADR RULE))
                (SETQ NEWSTATE (CADDDR RULE))
                (COND ((CDDDDR RULE)
                       (MAPCAR 'EVAL (CDR (CDDDDR RULE)))))
                (GO GET-STATE-DESCRIPTION))
               (T (GO TEST-TRANSITION-RULE)))
            WIN
         (ATTACH THIS-NODE PARENT-NODE)
         (SETQ LAST-PARSED THIS-NODE)
         (RETURN THIS-NODE)
            LOSE
         (SETQ REMAINING-WORDS HOLD)
         (RETURN NIL)))
```

This is an auxiliary function which makes use of NETWORK, an atom whose value is the network described by the syntax. For example, assume NETWORK has the following value:-

```
(S1 (IF <CONDITION1> ---> S2
     AFTER
     <ACTION>))
(S2 (IF <CONDITION2> ---> S3
              etc
```

1) INTERPRET requires two arguments (i) a network description, (ii) a parent_node.

2) INTERPRET maintains certain free variables (i) REMAINING-WORDS, (ii) CURRENT-WORD, (iii) LAST-PARSED.

3) INTERPRET uses a number of bound variables:-

i) NEWSTATE : always holds the name of the current node e.g. S1

ii) STATE-DESCRIPTION : contains the subnetwork associated with that node
e.g. (S1 (IF <CONDITION1> ---> S2
AFTER
<ACTION1>))

iii) RULE : contains the CONDITION (a test) associated with one arc out of a node. In the case of multiple arcs, RULE assumes the text of each CONDITION in turn until one test is successful. So the value of RULE might be:-
(IF <CONDITION1> ---> S2
If the result of evaluating RULE is non-nil then the next node will be (CDDDR RULE) i.e. S2 in this case.

iv) The MAPCAR. . . . EVAL combination causes any ACTION's to be executed following an AFTER.

DICTIONARY

This is the database for use by both the GENERAL PARSER and QUESSIE. If a word is to be recognised by the GENERAL PARSER, then it must either be defined as a root word or irregular form in the DICTIONARY, or be a regular form of that root word (these are recognised by the set of REGULAR ENDINGS routines). So the DICTIONARY contains all the words we want to use or recognise. They are stored in this form:-

```
(DEFPROP CAT (NOUN SINGULAR) FEATURES)
(DEFPROP SIT (VERB SIT ROOT INTRANSITIVE) FEATURES)
(DEFPROP SAT (VERB SIT PAST-PARTICIPLE INTRANSITIVE) FEATURES)
(DEFPROP ON (PREPOSITION) FEATURES)
etc
```

The LISP function DEFPROP puts lists of atoms such as (NOUN SINGULAR) under a property-value title such as FEATURES.

Associated Functions

a) RECORD : takes a network as its argument. It makes the network a property of the network title (e.g. PARSE-NOUN-GROUP) and then executes INTERPRET with the correct arguments.

b) GETF, TESTF : collect and test the features list of a word for the occurrence of a given feature (e.g. NOUN).

c) INTERSECTION : found here in its normal logical sense; it is not a standard LISP

function.

d) GENNAME : generates a unique new atom each time it is evaluated. If its argument is PARSE-NOUN-GROUP, then the resulting new atom is PARSE-NOUN-GROUPi, where i increases by one each time GENNAME is called with PARSE-NOUN-GROUP as its argument.

e) PARSE-WORD : uses GETF and TESTF to tet if a given word belongs to a certain feature group (e.g. NOUN).

f) SETR, GETR, ADDR : place, retrieve, and append the REGISTERS of a node as the result of effecting an ACTION. The REGISTERS are used by CONDITIONS and also in building the structure of the complete sentence.

g) ATTACH : connects networks together by making one network the PARENT of the other and the latter the CHILD of the first network. PARENT and CHILDREN are the titles of two property values of any network.

REGULAR ENDINGS

The size of the dictionary can be kept to a minimum by exploiting the regular endings of English words. LISP's atom splitting functions make it possible to look at the last few letters of any word. There are regular prefixes as well as regular suffices (endings) but prefixes do not often confer the same change of meaning on different words. For instance, return means 'to go back', yet replay means 'to play again'.

In the case of suffices, there near total uniformity in meaning imposed on the root words but there are more irregularities in the spelling. So we have a situation of regular irregularities; these classes are large enough on the whole to be worth writing routines to find there endings. For example, in these two comparative adjectives, (i) small ---> smaller, (ii) big ---> bigger, in the second case the last letter of the root word is duplicated.

In the GENERAL PARSER, there are systems which recognise the various regular forms of nouns, adjectives, and verbs. The capability could easily be extended to adverbs. As an example of how these subroutines work, the search for the comparative and superlative forms of adjectives is detailed.

REGULAR ADJECTIVES

MODE	SUFFIX ADOPTED	EXAMPLE	ROOT
comparative	-er	taller	tall
superlative	-est	tallest	tall
comparative	(last letter x 2)-er	bigger	big
superlative	(last letter x 2)-est	biggest	big
comparative	(y-->i)-er	happier	happy
superlative	(y-->i)-est	happiest	happy
comparative	---	more happy	happy
superlative	---	most happy	happy

'More' and 'most' can be used as modifiers with exactly the same semantic result, so the regular endings program looks for these two modifiers and if the next word is a root adjective then the mode can be set to either 'comparative' or 'superlative'.

The Program

Take for example the search routine for comparative adjectives:

```
(defun comparative-adj-search ()
   (and (setq current-word (car remaining-words) mode 'comparative)
        (setq w-c (reverse (explode current-word)))
        (cond ((testf current-word mode)
               (setq root (cadr (getf current-word))) (join1))
              ((equal current-word 'more)
               (join1)
               (setq root 'none)
               (cond ((testf (car remaining-words) 'adjective)
                      (setq root (car remaining-words))
                      (join1))) t)
              ((and (equal (list (cadr w-c) (car w-c)) '(e r))
                    (cond ((and (setq root (implode (reverse (cddr w-c))))
                                (testf root 'adjective)))
                          ((and (setq root (implode (reverse (cdddr w-c))))
                                (testf root 'adjective)))
                          (t nil)))
               (join1))
              ((and (equal (list (caddr w-c) (cadr w-c) (car w-c)) '(i e r))
                    (cond ((and (setq root (implode (append (reverse (cdddr w-c))
                                                            '(y))))
                                (testf root 'adjective)))
                          (t nil)))
               (join1)))))
```

1) This search routine caters not only for regular forms, but also searches the dictionary to see if the current-word has been defined as an irregular adjective, such as 'better' (root 'good'). These irregular forms return true in (tesf current-word mode); the mode is set; and the root is found from the dictionary.

2) If the current-word is 'more', (equal current-word 'more) returns true and the next word in the sentence is tested to see if it is an adjective. If it is, then comp-adj-search is successful and it returns the root adjective and mode = comparative (executed by join1).

3) If the current-word is 'bigger' say, then w-c takes the value (reverse (explode current-word)) i.e. (r e g g i b). Then (list (cadr w-c) (car w-c)) produces a list (e r) which is one of the endings being tested for. Then root is given the value (implode (reverse (cddr w-c))) i.e. (bigg). However, (testf 'bigg adjective) fails, so then root is updated by (implode (reverse (cdddr w-c))) which is the correct form (big).

4) In the case of 'happier', the current-word fails the first few tests and will drop down to the last test which removes the -ier ending, recognises this as a possible comparative ending, appends -y onto the remains of the word which here is 'happ'. This is then stored in root and the dictionary is searched for this root. The word 'happy' is found so the test is successful.

Thus the process is defined and it is a simple extension of this technique which allows other regular endings and their root words to be recognised.

NOUNS

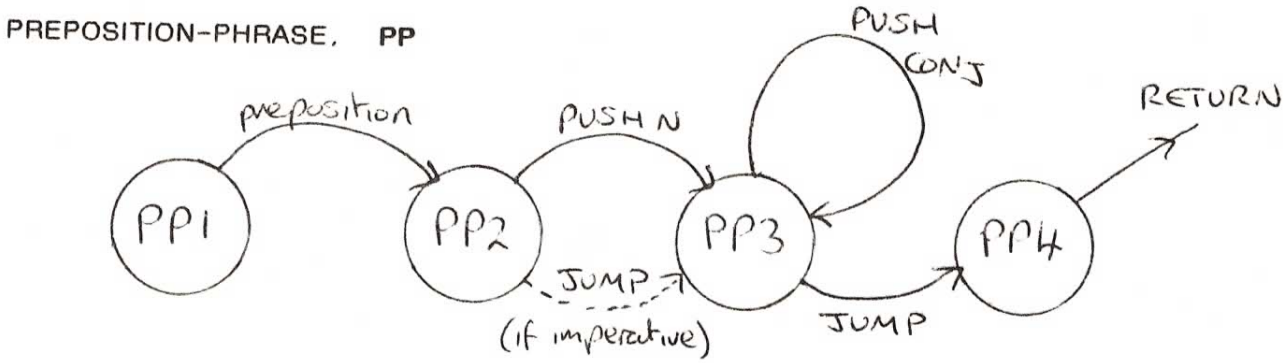
MODE	SUFFIX ADOPTED	EXAMPLE	ROOT
plural	-s	blocks	block

VERBS

TENSE	SUFFIX ADOPTED	EXAMPLE	ROOT
present	-s	works	work
present-participle	-ing	working	work
past-participle	-ed	worked	work
past-participle	-n	taken	take
present-participle	(last letter x 2)-ing	fitting	fit
present-participle	(drop e)-ing	taking	take
past-participle	-d	liked	like
past-participle	(y --> i)-ed	studied	study

ATN IMPLEMENTATION

The reason for using INTERPRET is that it is a relatively easy task to transform a sub-network diagram into the form accepted by INTERPRET. This is illustrated using the preposition sub-network.




```
(RECORD PREPOSITION
    (PP1 (IF (PARSE-WORD 'PREPOSITION) ---> PP2
             AFTER
             (SETR 'PREP LAST-PARSED)))
    (PP2 (IF (PARSE-NOUN-GROUP THIS-NODE) ---> PP3
             AFTER
             (SETR 'NOUN-PHRASE LAST-PARSED))
         (IF (EQUAL IMP-PHRASE 'IMPERATIVE) ---> PP3))
    (PP3 (IF (AND (SETQ LOCAL-PARSE 'PREPOSITION)
                  PARSE-IND
                  (PARSE-CONJ PARENTP))      ---> PP3
             AFTER
             (CONJ-SORT))
         (IT T --> WIN
             AFTER
             (SETQ PARSE-IND T))))
```

RECORD defines PREPOSITION as a function with the following network as a property. The 'preposition' test arc at PP1 is represented by :

```
(PARSE-WORD 'PREPOSITION)
```

If this is true then onto node PP2. The ACTION is to set register (SETR) 'PREP to the value of LAST-PARSED, which, in this case, is the last-word parsed - the preposition. If this test fails, as there are no more (IF (.....)) in NEWSTATE then the sub-network fails.

At PP2, PUSH N is represented by (PARSE-NOUN-GROUP THIS-NODE). Every defined network function requires an arqument, which will be the 'parent', in the parent/child relation, of the sub-network. THIS-NODE contains the value of current network being parsed, so when this PARSE-NOUN-GROUP network is PUSHed with argument THIS-NODE then the calling PP network will become the 'parent' on successful completion of the PARSE-NOUN-GROUP network. The ACTION is another SETR, however LAST-PARSED is the value of the GENNAME of PARSE-NOUN-GROUP. Another arc can be traversed from PP2, this is a JUMP arc, with the CONDITION that the IMP-PARSE flag has been set to IMPERATIVE in the case of imperative mood phrases.

At PP3 the CONDITIONs are, for the successful traversing of the LOOP arc:

(i) that 'local-parse' is set to value of calling function (PREPOSITION)
(ii) PARSE-IND flag has value T (true)
(iii) PARSE-CONJ is successful

PARSE-IND is initially set to truth value, T, however when a PARSE-CONJ function is called then part of the ACTIONs of this function is to set PARSE-IND to NIL; this prevents recursive structuring. In the case of more than one 'joined' phrase, which would give an embedded conjunction structure, it instead generates the required conjunctive structure.

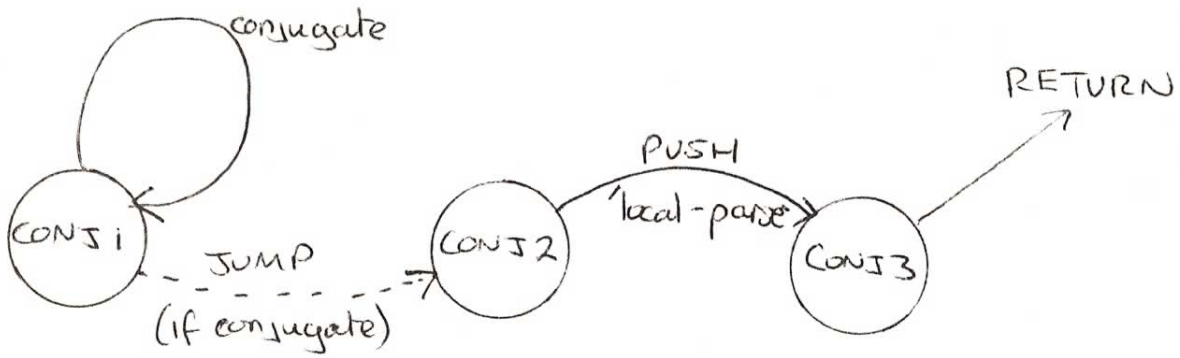
The PARSE-CONJ call has the argument PARENTP instead of the usual THIS-NODE. This means that the PARSE-CONJ structure (on successful completion) will be attached to the 'parent' of the calling PP function. The ACTION, when the LOOP arc is traversed is the function CONJ-SORT:

This function rearranges the parent/child relations to form the required conjunction structure. MODAL is set to the last successful phrase before the successful conjugate parse. If MODAL is not a conjugate phrase then it is ATTACHED to the just successfully parsed conjugate phrase. THIS-NODE is then set to conjugate phrase (to be ATTACHED in INTERPRET).

The JUMP arc at PP3 has the ACTION of resetting PARSE-IND to T, and '--> WIN' means go to the WIN in INTERPRET, this corresponds to the last node and the RETURN in a sub-network.

```
(RECORD PARSE-CONJ
   (CONJ1 (IF (AND (PARSE-WORD 'CONJ)
                   (OR (NOT (MEMBER 'COND (GETF LAST-PARSED)))
                   COND-IND)) --> CONJ1
                   AFTER
              (COND ((TESTF LST-PARSED 'NEGATIVE) (SETR 'NEGATIVE
                                                        LAST-PARSED)))
                   (SETQ PARSE-IND NIL)
                   (ADDR 'CONJ LAST-PARSED))
          (IF (GETR 'CONJ) --> CONJ2))
   (CONJ2 (IF (LOCAL-PARSE THIS-NODE) --> WIN
              AFTER
              (SETR 'SECOND-JOINED-PHRASE LAST-PARSED))))
```

PARSE-CONJUGATE, CONJ



The conditions for traversing the LOOP arc are:
(i) the first word is a conjunctive,
(ii) as 'then' is a conjunctive and a conditional pointer, to prevent it being parsed as a conjunctiv in a conditional-clause, the other CONDITION is included.

COND-IND is set to truth-value t, when a conditional-clause is being parsed. The ACTIONS, after a successful LOOP,
(i) set a NEGATIVE register if the conjugate was negative,
(ii) set PARSE-IND to nil as mentionned above,
(iii) set CONJ register to the conjugate.

The JUMP arc from CONJ1 has the CONDITION that a conjugate has been found.

At CONJ2 the CONDITION for traversing the arc, following a successful conjugate phrase, is that the 'local parse' function is successfully taken.

The rest of the sub-networks can be transformed similarly to the (RECORD ...) format, and the full implementation of this can be found in Appendix 3.

Associated Functions

a) BEGIN : this prints out an introduction instructing the user on which words can be input to the GENERAL PARSER. Then depending on the argument of BEGIN, proceeds to parse the input sentence. If the parse is successful, then the syntactic structure is TREEed.

b) HAPPY : initialises variables and then for a general parse (BEGIN 'PARSE-SENTENCE) is executed.

c) TREE : prints out, using PROPS, the syntactic structure for the argument X, by searching through the parent/child relations.

d) PROPS, P : prints out register properties.

e) NOT-INTERSECTION : the elements which are not common to two lists, e.g. the NOT-INTERSECTION of (WHICH IS A RED) and (A RED) is (WHICH IS).

f) SWAP : exchanges values of SUBJECT and OBJECT variables.

This interprtive computing method makes the implementation of node-networks easy. Improvements to this program could be: compiling to improve efficiency; and a graphical output for TREE. The present (worded) output can be seen in the examples of Appendix 5.

9. F.R.I.L. INTERFACE

INTRODUCTION

The specific needs of FRIL have, naturally, shaped the criteria for our FRIL interface. We set out to achieve a working interface for restricted types of query, so the interface certainly does not cover all the functions and set theoretic relations which FRIL offers, however, within the narrow domain that it operates it has proved successful. Before the FRIL interface is explained in detail it would be useful to say a little about FRIL in general and also relational knowledge representation.

F.R.I.L.

FRIL (standing for Fuzzy Relational Inference Language) is a new high level computer language for designing inferential knowledge base systems. FRIL consists of Base Relations (which are stored as tables), Virtual Relations (which define high level relations in terms of Base Relations or other Virtual Relations), Set Theoretic Relations (such as AND, NOT, OR), Functions (such as CARDINALITY), and Control Constructs (such as SEQUENCE, CONTROL, POINTER). The user acquires information from the system by asking a question through the query language. Queries take the form of WHICH(...) and DOES(...) queries and contain variables which are bound to attributes. There is an obvious motivation to work in a higher language than the very stylised FRIL. By interfacing Natural Language to FRIL, ordinary English questions can be translated to the appropriate FRIL format and the solution can be computed by running a FRIL implementation (there is one in LISP) with the relevant knowledge base.

(i) The DOES query requires a true/false answer, e.g.
'does Peter study Eng_Maths' which in FRIL reads,
DOES (STUDY (=PETER =ENG_MATHS))

(ii) The WHICH query requires, as a solution, a list of tuples (rows of a table) which satisfy the relations in the query, e.g. 'which students study maths' becomes
WHICH (X STUDENT(X) AND ATTEND(X =MATHS))

Where STUDY is a Base Relation which might look like this:-

STUDY	NAME	SUBJECT	Chi
	Peter	maths	1.0
	Rupert	politics	1.0
	Simon	maths	1.0
	...	...	..
	...	...	..

The Chi value indicates the degree of uncertainty of each statement; this is the Fuzzy nature of FRIL.

Interface

The system of problem generation and tabular manipulation which computes the solution of a FRIL query is not our concern here. Instead the problem is how to find a logical sequence of events to transform a Natural Language question into the FRIL query form, or more generally, how is Natural Language converted to a relational form? There is more than one way of handling this transformation, however there are some basic principles.

When studying FRIL, it is clear that it has (i) a simple syntax, and (ii) rigorously defined semantics. The semantics in FRIL manifest themselves in the declaring of argument-types for each relation and also domains over which quantitative measures are sensible. For instance, let the relation STUDY have arguments (NAME SUBJECT). Then 'maths studies Peter' is rejected as not conforming to the argument restrictions i.e. it is senseless and that is a semantic judgement. Similarly, 'the tall car' does

not agree with our sense of matching arguments. By declaring arguments for each relation, a rigorously defined semantic code is set up.

So how does this understanding of FRIL help the design of an interfacing process from Natural Language? The approach we have taken is to define a minimal subset of Natural Language which conforms to the same criteria as those of FRIL. The complex natural English syntax, which the GENERAL PARSER was programmed to handle, is discarded in the pursuit of a very simple syntax. So whereas in the GENERAL PARSER, the aim was to recognise and label every syntactic component of the input sentence, now we are specifically concerned with questions and instead of needing to know the the entire syntactic break-down, we are, for the most elementary purposes, only interested in,

(1) the nature of the questioner,
(2) the subject of the question,
(3) the subject of the verb,
(4) the main verb,
(5) the object.

By 'subject of the question', we mean that noun or inferred noun which is required as the answer to the question. For instance, in the question 'Which cat sits on the mat?', the subject of the question (and also the subject of the verb) is 'cat'. In the question 'Who likes Simon?', the subject of the verb is not explicitly mentionned; the inferred subject is a 'name' and this is derived computationally from the arguments of 'like'. Finally, in the question 'Who does Simon like?', the subject of the question is again inferred, yet the subject of the verb is 'Simon'. Here the verb-subject and the question-subject can be resolved from studying both the arguments of 'like' and the word-order of the main and auxiliary verbs (since the auxiliary verb comes first).

Of course, an amount of information is lost if only this skeleton is extracted, however, at first, this was all we could manage. And certainly for very simple queries this model can cope satisfactorily. The principles are simple; the verb is defined as a relation which links the arguments corresponding to the two nouns. The difficulty lies in simplifying the Natural Language enough to make this simple isomorphism of words and relations possible. Once this technique was mastered, we were able to build up the syntax to handle adjectives which modify the noun relations. Now, conjunctions can also be accepted which permits any number of subjects or objects to enter the query.

Our approach is in accord with the work of Alain Colmerauer, a French A.I. expert. He faced the same problem of how to systematically transform a sentence into a semantic formula. He starts with elementary statements involving <u>proper nouns</u> and the following hypothesis; to each verb, to each adjective, and to each common noun there corresponds a property with n arguments, each argument being a proper noun.

1) Elementary statements involving proper nouns:
   (i) noun + verb "to be" -- 'Tabby is a cat' -> ISCAT(TABBY)
   (ii) verb -- 'Tabby trots' -> TROTS(TABBY)
   (iii) adj + verb "to be" -- 'Tabby is spotted' -> ISSPOTTED(TABBY)

2) Three-Branched Quantifiers:

Colmerauer uses the determiner as a three-branched quantifier which relates a variable to two formulae. In the example, 'Arthur owns a car', Colmerauer claims the least ambiguous interpretation is A(X, ISCAR(X), OWNS(ARTHUR, X)). It is interesting to note that this is very similar to the FRIL form except it is a statement not a question.

IMPLEMENTATION

To find out the specific criteria of a question needed for the FRIL interface we have used a modified GENERAL PARSER ATN called QUESSIE. This can be seen in figure 18:

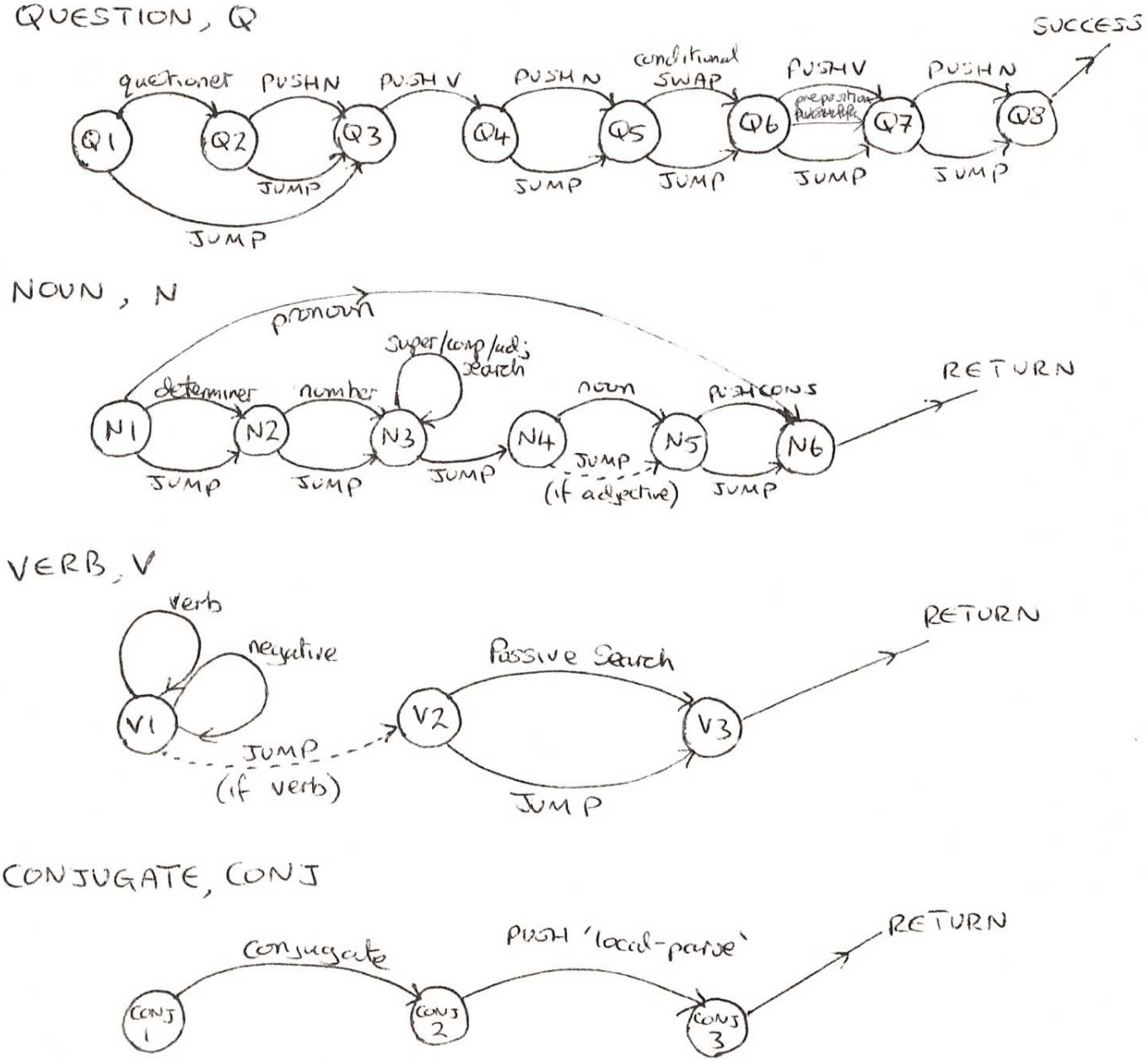


Fig. 18 ATN used for FRIL implementation

In QUESTION, instead of reorganising the question to an acceptable form for a phrase parser, the whole sentence is parsed.

A questioner flag is set as an ACTION on the successful traversing between nodes Q1 & Q2, this distinguishes between an inverted verb question and one which uses a word to introduce a question (who which what ...etc). If the question is of the latter type then FRIL requires the 'subject of the question', i.e. what the question is striving to find out. This is indicated by QUESTION-SUB, which is set between Q2 & Q3. If the PUSH N arc is successful then QUESTION-SUB is set to that noun-phrase, however if the JUMP arc is taken then it has a default value of NIL.

Instead of all the registers set in the GENERAL PARSER's noun-phrase, the only registers required in QUESSIE's noun-phrase are:

	NAME	if the phrase contains a pronoun or proper noun
	NOUN	is set to the root-noun of the phrase
or	ADJ	is set to the root adjectives of the phrase

The modified verb network only parses verb sequences so there is no PUSH N, the registers used are:

ROOT-VERB	is set to the active root verb of the phrase
NEGATIVE	is set if it is a negative phrase

The successful verb-phrase is then stored as QVERB.

Another ACTION of the verb network is to set a VOICE flag to PASSIVE if the question is in passive voice.

If there is a conjunction in the noun-phrase then, instead of the struture reorganisation used in the general parser, the joined noun-phrase is put as a 'child' of the first noun-phrase.

QVERB is set between Q3 & Q4, however if this is not the main-verb of the question then QVERB is reset between nodes Q6 & Q7.

The SUBJECT noun-phrase is set between Q4 & Q5 as an ACTION of the successful PUSH N.

The OBJECT noun-phrase is set between Q2 & Q3 or Q7 & Q8.

These SUBJECT and OBJECT nouns are the subject and object of the QVERB root-verb. So if the question is found to be in the passive voice then the SUBJECT and OBJECT noun-phrases are switched.

TEMP-ADJ is a flag set in the noun network when a noun-phrase is successfully parsed containing no noun i.e. is an adjective phrase. If this occurs then the adjective(s) of this noun-phrase are set to the ADJ of the other noun-phrase.

If the main-verb of the question, stored in QVERB, is modal, then QVERB is nulified. An example of the output of the question parser can be seen below, further examples can be seen in Appendix 7.

are the professors liked by tall women ?

QVERB	root-verb	- like
SUBJECT	noun	- woman
	adj	- tall
OBJECT	noun	- professor

(no questioner, so QUESTION-SUB is set to NIL)

We created our own way of representing the Base and Virtual relations, as the final FRIL format is still uncertain:

Base relations	arguments
student	(name subject age chi)
person	(name sex ht age wt chi)
professor	(name subject age chi)
car	(name colour chi)
dog	(name chi)
tall	(ht chi)
heavy	(wt chi)
like	(name name chi)
eat	(name food chi)
:	:

Virtual relations	definition
woman	((name) <-- (person (name =F ht age wt)))
small	((ht) <-- (not tall (ht)))
lecturer	((name) <--(professor (name subject age chi)))
:	:

The verb (in QVERB), the noun (in SUBJECT and OBJECT) and associated adjective(s) are checked to see if they are in the relations, if not then an alternative can be asked for.

The arguments (not chi) in the verb are then checked against the corresponding SUBJECT and OBJECT noun's arguments, for the above example:

relation	argument
like	(name name)
woman	(name)
tall	(ht)
professor	(name subject age)

If this is true then the subject and object nouns arguments are replaced by a variable and the resulting term is stored:

SUBJECT	Term (sub)	(person (x =F ht age wt))
	variable (subv)	(x)
OBJECT	Term (ob)	(professor (y subject age))
	variable (obv)	(y)

If this is false and the arguments don't match then a 'relational missmatch' has occured. This gives a measure of semantics to QUESSIE.

However if either the SUBJECT or OBJECT do not contain a noun, but contain a 'name', for example in:

is Peter a tall person ?

In this case, the variable is set to **=Peter** and the term is set to NIL.

Also, if either the SUBJECT or OBJECT phrases are NIL, for example:

who likes Simon ?

then a variable is set because this may be the subject of the question (as in the above example).

When a noun has an associated adjective, then the same process gone through with the verb and noun arguments is done with the adjective and noun arguments. If the arguments match then a new variable is put in the arguments of the adjective and noun term and a new term is formed. This can be seen in the 'tall woman' example where the 'sub' term becomes:

(tall (z) and person (x =F z age wt))

However, if the adjective is not of the Interpolative (Fuzzy) type and hence not stored in the relations, for example:

the red car

then a list of such adjectives with their arguments is consulted. Then the same process as above as gone through, however instead of forming a new variable then the adjective itself is put in the noun term:

(car (name =red))

When a noun has a conjunction then the whole process of the noun sort is repeated with the same verb argument. If they match then the term produced is joined to the previous sub/ob term with this conjunction and the variable used is stored with the previous sub/ob variable ~~variable~~. for example:

which tall women and dogs like professors ?

```
sub   - (tall(z) and woman (x =F z age wt) and dog (w))
subv  - (x w)
ob    - (professor (y subject age))
obv   - (y)
```

Once all this sorting is done then the FRIL format can be obtained:
The terms are firstly tidied up, taking all the arguments not required by the FRIL query.

Then the verb term is formed by joining all the combinations of subv and obv with the qverb, as in the above example:

(like (x y) and like (w y))

This term is then joined with the sub and ob terms to give:

```
(tall (z) and woman (x =F z _ _) and dog (w) and
          like (w y) and like (x y) and professor (y _ _))
```

If the original question had a questioner then the subject is taken into account to find the question variable. It is found by consulting the question-sub compared with the subject and object set by QUESSIE. This question variable is then combined with the term and WHICH to form the FRIL query.

If no questioner then the term is combined with DOES to form the FRIL query.

Once the FRIL format is obtained the interface is completed.

Program

QUESSIE uses the same functions as the GENERAL PARSER, other fuctions required are the following, and (of course) the FRIL program.

RUN_FRIL runs the FRIL generation program, QU, and also completes the FRIL interface.

QU prints BEGIN introduction and accepts query, then reorganises the SUBJECT, OBJECT and QVERB and calls the argument sorter function QSORT.

QSORT sorts verb, noun and adjective arguments to produce the FRIL format.
NOUN-SORT checks that the verb and noun arguments match and then produces a noun term.
ADJECTIVES checks the adjective and noun Arguments and produces a modified noun term.
SORT-VAR, CHANGE-VAR, SWAP-ARG checks and changes relations arguments.

ANDP joins terms with 'and'.
CONJ-ARG, FRIL-GENERATE-OB, FRIL-GENERATE-SUB creates verb terms.

TIDY-UP-ARG, ARG-SUBST tidies up term's arguments

Interface Improvements

An improvement to QUESSIE would be to try an cope with the same comprehensive syntax as the GENERAL PARSER by catering for: preposition phrases, superlatives, comparatives, conditional clauses, subordinates . . etc.

By using the MATCH function used in Pattern Matching, see section 4, then FRIL queries such as those below could also be catered for:

how many .... ?
list all possible .... ?
name all possible .... ? etc

The only multiple argument relations dealt with so far are in the Base relations :

i) **Verbs** - two noun arguments
ii) **Nouns** - main argument + as many adjective arguments as needed

However some nouns and adjectives need multiple relations (Base and Virtual), for example:

friends (name name chi) big ((x y) <-- (heavy (x) and tall (y)))

The noun multiple arguments case is usually located in natural language, when a noun is introduced by a preposition-phrase, for example:

Peter is a friend of James

So far, we have only included the one set theoretic, and, however the introduction of other relations, such as NOT, OR, is as feasible. FRIL function and conrtol constructs could also be included. Another property of FRIL, which could be solved with more work, is relations which are word conglomerations such as:

good_athelete tall_person people_on_course . . . etc

More semantics could also be introduced, due to the defined relational arguments:

i) inference
who is tall ?
tall infers to the noun-relation 'person'

ii) proper-nouns
put arguments as in the non l-type adjectives:
Peter (name) artichoke (food) Maths (subject)

iii) ambiguity
for example **lecture** is a noun or adjective

We foresee no great problems in modifying the FRIL interface to deal with these suggested improvements, however as the implementation of FRIL is in it's relatively early stages and, more importantly, our lack of time we have not pursued these further.

11. REFERENCES

Alain Colmerauer *"An Interesting Subset of Natural Language "* (1979) Groupe Intelligence Artificielle, Univ. Aix-Marseille

A.J.T. Davie & R. Morrison *'Recursive Descent Compiling'* (1981) Ellis Horwood

D.R. Hofstadter *'Godel, Escher, Bach: an eternal golden braid'* (1979) Penguin

M. Temple *'Get It Right - A guide to written English'* (1978) John Murray

Harry Tennant *'Natural Language Processing'* (1981) Petrocelli

Joseph Weizenbaum *'Computer Power and Human Reason'* (1976) W.H. Freeman

Terry Winograd *'Language as a Cognitive Process'* (1982) Addison-Wesley

P.H. Winston & B.K.P. Horn 'LISP' (1981) Addison-Wesley

Baldwin J.F. & Zhou S.Q. (1982) *"A Fuzzy Relational Inference Language"* (To appear in 'Fuzzy Sets and Systems')

Baldwin J.F. (1983) *"Knowledge Engineering using a fuzzy relational inference language"* IFAC Symposium on Fuzzy Information, Knowledge Representation and Decision Analysis, Marseilles 1983.

Baldwin J.F. (1983) *"A Knowledge Engineering Fuzzy Inference Language - F.R.I.L"* Proc. of AUWE Knowledge Engineering Conference, 1983.

Baldwin J.F. (1983) *"A Fuzzy Relational Inference Language for Expert Systems"* Univ. of Bristol, Dept. Eng. Maths, internal report. To appear in Proc. 13th Int. Symp. on MVL, Kyoto, Japan, 1983.

APPENDIX 1. CAREER program listing

```
(defun career ()
       (prog ()
             (foreword)                                          ;initial use
             (setq pointer '1 morep t n 'please)
                                                                 ;initialise
             (intro)                                              ;starts off
             (terpri)
        loop (setq answer (read))                                  ;read user
             (setq pointer (add1 pointer))
             (cond ((equal pointer '5) (cond (morep (setq morep nil))
                                             (t (setq morep t)))
                                       (setq pointer '1)))
                                                                 ;reset point
             (terpri)
             (cond ((match '(+ my +) answer)
                    (setq answer (subst 'your 'my answer))))
                                                                 ;change all
             (cond ((badword))                                    ;check for
                   ((you))                                        ;replies cu
                   ((yes-no))                                     ;replies if
                   ((verbs))                                      ;responds c
                   ((I))                                          ;if another
                   ((end))                                        ;nil input
                   ((it))                                         ;if it occu
                   ((more)))                                      ;if all els
                                                                  ;indicated
             (terpri)
             (go loop)))


(defun intro ()
         (print '(Hello,what course do you do?))
                (terpri)
                (setq deg (read))
                (setq answer deg)
                (terpri)
                (cond ((match '(+ do +r ) deg) (setq deg r)
                        (print (append (append '(do you enjoy) deg)
                       ((apply 'or (mapcar 'word-search subjects))
                            (print (append '(Do you enjoy) deg)))
                       (t (setq deg '(your degree)) (print '(Do you e


(defun badword ()
  (cond ((apply 'or (mapcar 'word-search badwordp))
    (print '(Please dont be so insolent  You are the one that is unemp

(defun you ()
         (cond ((or(match '(+ you +) answer)
                   (match '(+ You +) answer))
```

r 05:26 0.095 1 level 3

APPENDIX 2. CAREER sample output

```
(career)
(Welcome to the Baldobrown Careers Advice Program)
(Please answer the questions in reasonable English)
(All replies must be enclosed in brackets (LISP formality))
(When you want to finish Return nil (no brackets))


(Hello what course do you do?)
(eng maths)

(Do you enjoy eng maths)
(yes)

(For what reasons ?)
(you cant ask questions like that)

(Do you want to be an interviewer ? You seem keen to ask questions !
(maybe)

(You are not sure are you?)
(not really)

(What else are you interested in?)
(I like working with animals)

(Oh that s good ! Why do you like working with animals)
(I love looking after them)

(Do you fancy working in a job involving this ?)
(possibly)

(Now tell me about your past jobs)
(i once worked as a shop manager)

(Oh thats interesting Tell me more ?)
(Do you enjoy thinking up inane questions)

(We are discussing you)
(sorry)

(How about working abroad or a job involving eng maths)
(no)

(STOP being negative ! Try and help me ?)
(ok)

(What do you want from life ?)
```

APPENDIX 3. GENERAL PARSER program listing

```
(defun interpret (network parent-node )
       (prog (modal newstate  state-description rule hold this-node
             (setq modal nil parentp parent-node)
             (setq hold remaining-words)
             (setq this-node (genname (car network)))
             (setq network (cdr network))
             (setq newstate (caar network))
             get-state-description
             (cond ((equal newstate 'win) (go win))
                   ((equal newstate 'lose) (go lose))
                   (t (setq state-description (assoc newstate networ
             test-transition-rule
             (cond ((setq state-description (cdr state-description))
                    (setq rule (car state-description)))
                   (t (setq newstate 'lose)
                      (go get-state-description)))
             (cond ((eval (cadr rule))
                    (setq newstate (cadddr rule))
                    (cond ((cddddr rule)
                           (mapcar 'eval (cdr (cddddr rule)))))
                    (go get-state-description))
                   (t (go test-transition-rule)))
              win
             (attach this-node parent-node)
             (setq last-parsed this-node)
             (return this-node)
             lose
             (setq remaining-words hold)
             (return nil)))


(defun record fexpr (x)
          (putprop (car x) x 'network)
          (eval (subst (car x) 'name
                       '(defun name (parent-node )
                              (interpret (get 'name 'network) parent
                                                                  ))))


(defun getf (x) (get x 'features))

(defun testf (node features)
         (cond ((null features))
               ((atom features) (setq features (list features))))
         (equal (length features)
                (length (intersection features (getf node)))))
```

```
r 05:47 0.099 1 level 3

lisp
*
(load 'startq)
t
```

APPENDIX 4. GENERAL PARSER sample output

```
(happy)
THE DICTIONARY
PRONOUNS-- I you he she it we they
NOUNS-- artichoke bicycle block boy car cat cheese course dog girl h
lecture lecturer mat mouse penguin pub professor student table compu
giant person woman drink drunk beer gin vodka steak potato cheesecak
chicken avocado ** (plus all their plural forms)
ADJECTIVES-- black blue green red yellow white bad boring good happy
big light mathematical cheap expensive maths
mad many much old solid tall young ** (plus comparatives and superla
VERBS-- appeal attend be can eat follow have lecture live love sit
drink own drive teach scare smile study take ** (plus all their tens
ADVERBS-- happily lazily proudly quickly slowly solidly stubbornly
PREPS-- around at by from in into on over to under
REL PRONOUNS-- that which who
DETERMINERS-- a an the these this that those
QUESTIONERS-- how what which who why
CONJUNCTIONS-- and but no not yet
NUMBERS-- one two three four five six
NEGATION-- no not

Please input sentence (in brackets)
(if the three students are sitting happily in the lecture then the
professor who is older than the student can not have teaching)

(parse-sentence no : . parse-sentence1)
parse-sentence1
(phrase --> parse-phrase1)

parse-phrase1
(with-condition --> conditional-clause1)
(noun-phrase --> parse-noun-group3)
(verb --> parse-verb3)

conditional-clause1
(conditional-word --> if)
(conditional-phrase --> parse-phrase2)

parse-phrase2
(noun-phrase --> parse-noun-group1)
(verb --> parse-verb1)

parse-noun-group1
```

APPENDIX 5. DICTIONARY Listing

```
;NOUNS
(defprop I (pronoun singular) features)                    ;pronouns
(defprop you (pronoun plural singular) features)
(defprop he (pronoun singular) features)
(defprop she (pronoun singular) features)
(defprop it (pronoun singular) features)
(defprop we (pronoun plural) features)
(defprop they (pronoun plural) features)

(defprop artichoke (noun singular) features)               ;common nouns
(defprop bed (noun singular) features)
(defprop bicycle (noun singular) features)
(defprop block (noun singular) features)
(defprop boy (noun singular) features)
(defprop car (noun singular) features)
(defprop cat (noun singular) features)
(defprop cheese (noun singular) features)
(defprop computer (noun singular) features)
(defprop beer (noun singular) features)
(defprop food (noun plural) features)
(defprop gin (noun singular) features)
(defprop vodka (noun singular) features)
(defprop steak (noun singular) features)
(defprop potatoes (noun plural potato) features)
(defprop potato (noun singular) features)
(defprop cheesecake (noun singular) features)
(defprop chicken (noun singular) features)
(defprop avocado (noun singular) features)
(defprop course (noun singular) features)
(defprop dog (noun singular) features)
(defprop giant (noun singular) features)
(defprop girl (noun singular) features)
(defprop house (noun singular) features)
(defprop lecturer (noun singular) features)
(defprop man (noun singular) features)
(defprop men (noun plural man) features)
(defprop mat (noun singular) features)
(defprop mouse (noun singular) features)
(defprop mice (noun plural mouse) features)
(defprop penguin (noun singular) features)
(defprop person (noun singular) features)
(defprop people (noun plural person) features)
(defprop pub (noun singular) features)
(defprop professor (noun singular) features)
(defprop student (noun singular) features)
(defprop table (noun singular) features)
(defprop woman (noun singular) features)
(defprop women (noun plural woman) features)


(defprop dummy-qnoun (noun singular) features)


;ADJECTIVES
(defprop good (adjective) features)                        ;irregular
(defprop better (comparative good) features)
(defprop best (superlative good) features)
(defprop bad (adjective) features)
(defprop worse (comparative bad) features)
```

APPENDIX 6. QUESSIE program listing

```
(record conj
     (CONJ1 (if (parse-word 'conj) --> CONJ2
                after
                (setr 'conj last-parsed)))
     (CONJ2 (if (local-parse this-node) --> win)))


(record verb

   (V1 (if (verb-search) --> V1
           after
           (setr 'root-verb root))
       (if (parse-word 'negative) --> V1
           after
           (setr 'NEGATIVE last-parsed))
       (if (getr 'root-verb) --> V2))
   (V2 (if (and (parse-word 'preposition)
                (equal type 'past-participle)

                (or (not (testf (getr 'root-verb) 'intransitive))
                    (parse-word 'preposition))) --> win
           after
           (setq voice 'passive))
       (if t --> win)))


(record noun


   (N1 (if (parse-word 'pronoun) --> win
           after
           (setr 'name last-parsed))
       (if (parse-word 'determiner) --> N2)
       (if t --> N2))
   (N2 (if (parse-word 'number) --> N3)
       (if t --> N3))
   (N3 (if (super-comp-adj-search) --> N3
           after
           (addr 'adj root))
       (if t --> N4))
   (N4 (if (parse-word 'noun) --> N5
           after
```

APPENDIX 7. FRIL INTERFACE output.

```
(run_fril)
 Do you want to ask a Natural Language query for FRIL (y/n) ? y
THE DICTIONARY
PRONOUNS-- I you he she it we they
NOUNS-- artichoke bicycle block boy car cat cheese course dog girl h
lecture lecturer mat mouse penguin pub professor student table compu
giant person woman drink drunk beer gin vodka steak potato cheesecak
chicken avocado ** (plus all their plural forms)
ADJECTIVES-- black blue green red yellow white bad boring good happy
big light mathematical cheap expensive maths
mad many much old solid tall young ** (plus comparatives and superla
VERBS-- appeal attend be can eat follow have lecture live love sit
drink own drive teach scare smile study take ** (plus all their tens
ADVERBS-- happily lazily proudly quickly slowly solidly stubbornly
PREPS-- around at by from in into on over to under
REL PRONOUNS-- that which who
DETERMINERS-- a an the these this that those
QUESTIONERS-- how what which who why
CONJUNCTIONS-- and but no not yet
NUMBERS-- one two three four five six
NEGATION-- no not

Please input sentence (in brackets)
(who studies Maths)

(question no : . question1)
nil
(OBJECT)
(name --> Maths)
(SUBJECT)
(VERB)
(root-verb --> study)

(which ((x) study (x =Maths)))


r0001
name        chi

Fred        1.0
James       1.0

end
 Do you want to ask a Natural Language query for FRIL (y/n) ? y
THE DICTIONARY
PRONOUNS-- I you he she it we they
NOUNS-- artichoke bicycle block boy car cat cheese course dog girl h
lecture lecturer mat mouse penguin pub professor student table compu
giant person woman drink drunk beer gin vodka steak potato cheesecak
```