THE UNIVERSITY *of* EDINBURGH

COMPUTER MODELLING
OF ENGLISH GRAMMAR

GRAEME DONALD RITCHIE

PHD. UNIVERSITY OF EDINBURGH 1977

ABSTRACT

Recent work in artificial intelligence has developed a number of techniques which are particularly appropriate for constructing a model of the process of understanding English sentences. These methods are used here in the definition of a framework for linguistic description, called "computational grammar". This framework is employed to explore the details of the operations involved in transforming an English sentence into a general semantic representation. Computational grammar includes both "syntactic" and "semantic" constructs, in order to clarify the interactions between all the various kinds of information, and treats the sentence-analysis process as having a semantic goal which may require syntactic means to achieve it. The sentence-analyser is based on the concept of an "augmented transition network grammar", modified to minimise unwanted top-down processing and unnecessary embedding. The analyser does not build a purely syntactic structure for a sentence, but the semantic rules operate hierarchically in a way which reflects the traditional tree structure. The processing operations are simplified by using temporary storage to postpone premature decisions or to conflate different options. The computational grammar framework has been applied to a few areas of English, including relative clauses, referring expressions, verb phrases and tense. A computer program ("MCHINE") has been written which implements the constructs of computational grammar and some of the linguistic descriptions of English. A number of sentences have been successfully processed by the program, which can carry on a simple dialogue as well as building semantic representations for isolated sentences.

CONTENTS

INTRODUCTION

The work reported in this thesis tries to achieve two closely related goals. Firstly, to combine some of the ideas and techniques of recent work in artificial intelligence to form a model (or partial model) of language which allows linguistic description of English in processing terms. Secondly, to investigate in more detail some of the processing mechanisms that must be present in such a model in order to describe the conversion of a string of English words to a representation of the corresponding "meaning".

The second, more specific, goal was the main aim, but it was found necessary to spend some time establishing a frame of reference, since there is currently no accepted processing model of language. The framework used here (called "computational grammar" for ease of reference) is not a totally new model, since it relies heavily on the work of Chomsky, Fillmore, Montague, Woods, Winograd and others (as will be seen from Chapters II and III). In order to clarify some of the ideas involved in computational grammar, a computer program (the "MCHINE" system) has been written, which can convert a small range of English sentences to a semantic representation or carry on a very simple dialogue.

There are thus several facets to the investigation. There is the general framework of computational grammar (set out in Chapter IV), which is based on certain assumptions (outlined in Chapter I) and on some reactions to existing models (discussed in Chapter III). Next, there is the application of computational grammar to various fragments of English, and the consequences which these analyses have

for the processing mechanisms that must be postulated (see Chapter V). Finally, there is the MCHINE program, which implements some of the English descriptions and acts as a crude test of the devices of computational grammar; the program is described in Chapter VI.

Computational grammar is neither wholly adequate nor even complete (as indicated in Chapter VII), but it has provided a basis from which some useful points have emerged.

The processing mechanisms have several interesting properties. Various "bottom-up" principles have been included, which reduce the range of options that have to be specified explicitly in a grammar. There is a general, partially-hierarchical system of information storage which has, as an automatic consequence, the phenomenon previously described in transformational grammar by the "Complex Noun Phrase Constraint". Although the hierarchical surface structure of English may be fairly complex, the processing can be simplified by a technique which makes a special provision for right-branching structures, thereby avoiding unnecessary embedding of processes.

The rules used for combining meanings are also important, since they lie on the traditional boundary between syntax and semantics. A single set of rules simultaneously defines the possible surface structures and states what semantic combinations these structures represent.

Within the area of semantic representation, a particular kind of structure has been devised which is particularly suitable for the representation of the meaning of a wide variety of surface structures, and which therefore allows various semantic processes to

be simplified.

Several areas of English grammar have been explored using the computational framework, and these are described in Chapter V. Sections V.1, V.2, V.3 and V.5 illustrate how the various devices in computational grammar operate, and their appropriateness for certain phenomena. The analysis of "wh"-clauses, in Section V.4, shows how an otherwise arbitrary linguistic constraint can be re-expressed naturally in computational terms. Section V.6 combines concepts from logic and computation to develop a way of describing referring expressions. A detailed investigation of tense and time is set out in Section V.7, and the description of verb phrases in Section V.8 shows how careful use of lexical entries can simplify the grammar.

CHAPTER I

METHODOLOGY AND METATHEORY

Section I.1 : Theories, Paradigms and Frameworks

The phrases "theory of language" or "linguistic theory" are often used in discussing research into natural language. The aim of this section is to question whether we always have "theories", and to suggest that something weaker is guiding current research.

One widely-accepted version of how scientific investigation proceeds is characterised by Hempel (1966). In such an approach, a theory is a coherent, structured body of ideas concerning some subject, which makes predictions that may be verified, and which can be used to "explain", in some sense, observations made about the subject matter. Hempel presents a very neat picture of a highly organised objective scientist using a carefully constructed theory in an unprejudiced fashion . A somewhat different analysis is presented by Kuhn (1970), who has less faith in the objectivity of the scientist. Kuhn sees science working within a series of "paradigms". A paradigm is an accepted theory which not only guides research, but positively straitjackets it, by defining problems, and possible solutions, in such a way as to narrow the scientist's view greatly. A paradigm is not just a theory, but has an attached set of ways of working and a whole terminology of its own. Hence it imposes a very definite structure on the way research proceeds. Whereas Hempel discusses the notion of choosing between competing theories by gathering evidence, Kuhn asserts that these rational choices are not possible between paradigms, since each paradigm defines differently the criteria for such choices.

A case could be made for regarding linguistic research as proceeding in Kuhnian paradigms. A paradigm in this sense contributes to research at two levels - it provides a theory, which can be regarded as "true" or "explanatory"; it also provides guidelines for the everyday investigations of the subject matter. It is this latter aspect that is more relevant in artificial intelligence and computational linguistics, since both lack an overall "theory" to provide a received body of "facts" or "explanations". In the period before the development of a paradigm, there is, according to Kuhn, an atheoretical stage in which a body of techniques and concepts are built up, which serve to guide investigation and description of a topic. Let us call such a collection of devices a "framework". A framework is weaker than either a theory, in Hempel's sense, or a paradigm, in Kuhn's sense. It makes no really precise predictions, since it is not sufficiently elaborated. It cannot, therefore, claim to provide any explanation, since it does not relate all observed phenomena to some systematic generality. It has no use for "critical experiments" or "crucial counter-examples", since it is not an integrated edifice that can be demolished by removing one brick. It contains no "truths" which can be taken as unquestionable by the community using the framework. However, it is of great use to what Fillmore (1972) calls the "ordinary working grammarian". It provides tools for investigating and describing the subject-matter. It provides a terminology which workers in the field can use to communicate with each other. It suggests problems that must be attacked, and, often, ways of attacking them. It provides some means of assessment of putative solutions (usually using some general classification of "elegant"

against "ad hoc"). It is a cluster of related ideas, rather than a tightly structured whole, so that different workers may disagree over the acceptance of certain aspects of the framework, and yet still be able to share the rest of it.

Current artificial intelligence work definitely has a framework, but it is doubtful if it has a paradigm, since it lacks any real theory. To demonstrate that there exists a common cluster of ideas and techniques, it is necessary only to list some of the terminology currently used in artificial intelligence: top-down, bottom-up, depth-first, breadth-first, goal-directed, procedural-embedding, distributed knowledge, flow of control, access environment, subroutine call, pattern-invocation, demons, middle-out, backtracking, plan formation, interacting goals, updating a world model, etc.

It might be thought that, without the concept of a critical experiment, there could be no notion of "refutation" or "incorrectness" within a framework. That this is not so can be demonstrated by considering the form of argumentation conducted in transformational linguistics during the 1960s. Although the results of Peters and Ritchie (1971, 1973) showed that the mechanism of transformational grammar was so general as to be virtually irrefutable, linguists continued to produce "counter-examples" to each other's claims. Despite the fact that the formally defined transformational grammar could, strictly speaking, perform any operations whatsoever, there was an additional notion of "simple" against "ad hoc". This could be used to eliminate many of the technically possible manipulations - a "counter-example" was an

"example which could not be handled neatly" .

This has to be borne in mind when discussing computational linguistics. In order to cope with the detailed complexity of grammatical phenomena, computational descriptions generally contain mechanisms which are formally as powerful as a Turing machine. However, what makes such models distinct from each other is the different structuring that is imposed on the mechanisms. Coupled with the notion of "simplicity" or "elegance", the different models (or rather, the different sub-frameworks) may well make different claims.

Section I.2 : Some Assumptions

Part of a framework consists of the assumptions it makes in order to provide an initial basis for investigation. This section outlines the assumptions for later chapters, without providiing any real justifications for any of the positions adopted.

I.2.1 Semantic Structure

It is logically possible that a language-understanding system might be constructed in which all tasks (such as inference, detection of ambiguity, etc.) can be performed using the word string as the sole representation of the input sentence. That is, no independent level of "meaning" is required to show, for example, the similarites between the pairs of sentences below, or to detect the ambiguity of (3).

(1)

(a) John bought an alligator from the zoo manager.

(b) The man who administers the zoological gardens sold John an alligator.

(2)

(a) The alligator attacked John.

(b) John was attacked by the alligator.

(3) Attacking alligators can be dangerous.

If such an approach is viable, the onus of proof rests with those advocating it, since years of linguistic and philosophical research have demonstrated the unsuitability of word strings for manipulations such as inference.

This does not necessarily involve claiming that the surface string contains "less information" than some other "semantic" structure for the sentence; if the latter is computed from just the surface string, this cannot be true, although the information may be in a more useable form in the computed structure. However, the computation might use information from outside the word string itself (contextual clues), which will make a contribution to the semantic structure. Not performing the conversion might entail storing these contextual clues (if they could be isolated) along with the surface form, so that the information they contained could be used whenever the "meaning" of the sentence was required.

Some balance must be struck, nevertheless, between the simplicity of semantic operations (for inferencing, etc) and the complexity of the way of converting surface form to semantic structure. (Cf. comments in Section II.2 on interpretive semantics).

The assumption used, therefore, in this project, is that there is some level of representation which can be called "semantic structure" which is logically distinct from the surface form, and which is computed from the surface form together with other (contextual) information.

It might be argued that to produce a "structure" is misguided, as the best way to analyse an English utterance, in context, is to consider the effect that was intended by the speaker (on any hearers) and/or the effect that the utterance had on the hearer(s). Thinking in terms of conversational effects is extremely valuable, and may be the most general way to describe an utterance - but this does not contrast with the "structure" approach outlined here, except in terminology. Suppose we have a linguistic model in which sentences are described in terms of their effect on the speaker and/or hearer. This model (unless it is to be totally unilluminating) will have to provide a precise representation, in some formal terms, of these "effects"; hopefully, this representation may even allow some comparison of "similar effects", so that similarities like those in (1) and (2) are shown. Such a set-up will not differ fundamentally from the vague outline presented already - "semantic structure" is being used in such a broad sense that it does not contrast with "representation of conversational effect".

I.2.2 : Syntax

In certain logical languages like the predicate calculus, syntax has a clearly-defined purpose and is distinct from (though systematically related to) semantics. Syntactic rules specify which strings in the language are to be classed as "well-formed", and only well-formed strings need be considered for inference or other semantic manipulation. This is similar to the notion that Chomsky adopted in his theory of natural language (1957, 1965), although it has now been widely questioned (see Section II.1 for further discussion). It is logically possible that a natural language may not

have a comparable concept of "well-formed/ill-formed" characterised by a set of syntactic rules, although some sentences may express an odd idea, and some sentences may be more complex to process. Also, some sentences may sound odd because they are uttered in an inappropriate context.

The investigation in later chapters assumes that we need not search for rules of syntactic well-formedness, and that it is sufficient to search for rules of semantic manipulation, together with rules for converting between surface form and semantic structure. Many of the phenomena attributed to syntax (particularly Chomskyan "grammaticality") may be more accurately describable in terms of semantic or conversion rules(see also Section I.3 below). This may sound as if syntax is merely being banished to lurk under another heading, since syntax has traditionally been a fundamental part of the process of conversion between surface form and meaning. This is true, since any conversion rules which are postulated will almost certainly have to make extensive use of traditional syntactic notions such as "verb", "preposition", possibly "transitive" and perhaps even "subject"; the change that is being highlighted here is more a change of emphasis. In this project, the primary aim of writing these grammatical rules is not to specify any "well-formedness" for sentences; the goal is to show how surface strings relate to semantic structure. Having made that point, the terms "syntax" and "syntactic" will be used in the later chapters for rules or structures whose sole task is to guide the conversion process, and which cannot reasonably be said to be describing patterns of meaning. For example, the class of verbs which appear at the front of the sentence in English questions (auxiliary verbs) is a

useful syntactic class, since it aids in the detection of the question construction; it is not obvious that auxiliary verbs are a semantic class (although some more sophisticated analysis might show this to be so).

This does not mean that no notion of "well-formedness/ill-formedness" will result from such rules. If there are conversion rules and semantic rules, then certain items may fail to be processed by one or the other. An accurate model would be one whose rule-failures, or complexities of processing, correspond to utterances or strings which sound bad to the traditional native speaker. However, it is the model as a whole which will characterise such ill-formedness, not just one component. In fact, the existence of these native judgements of "good/bad" provide much of the evidence for rules of any kind, and the Chomskyan style of argument can be adopted in discussing such data. (See I.3 and III.8 for further remarks on this topic).

I.2.3 : Recognition Rules

So far, the (deliberately vague) term "conversion" has been used for the relationship between surface form and semantic structure. This concept can now be examined in greater detail.

A complete model of a language user will have to specify, among other things, how semantic structure can be converted effectively to strings of words (production) and how strings of words can be effectively converted to semantic structure (recognition). One obvious way of making such specifications would be to present two (probably enormous) algorithms, one for production and one for

recognition. To allow for alterations to the language (or idiolect, if modelling a specific user), it might be preferable to separate out some items called "rules" which the algorithm uses to perform its task; language-expansion could consist then of alteration to the rule-system, or to the algorithm, or to both. The two algorithms are distinct, since the form of input used for one is the form of output for the other (and vice-versa). It might be possible, nevertheless, to find common aspects of the two processes. Ideally, it would be neat if all such points of similarity could be separated out and built into the rules; then there might be two totally distinct interpreters (in the computational sense) working on one common set of data (the rules). This could of course be done trivially, just by taking the union of the two disjoint sets of "rules" to be the common set, but that is not what is meant. (See Kay (1975) for some comments on this topic).

Some linguists claim to be building a model of language which is not biased towards either production or recognition (this was one of the claims of Chomksy (1965)). In a sense this is true, but it is reminiscent of the joke about a blind horse seeing equally well at both ends. Chomskyan linguists usually ignore the need for effective production and recognition procedures, and so in a sense the models are equally incomplete answers to both problems. Such an approach also has to assume that all aspects of language can be described in such a neutral, non-directional way. Hence, if there are any phenomena which are direction-specific (i.e. are consequences of the algorithmic parts) either these phenomena will not be describable or else the neutral formulation will have to be distorted to accommodate them.

Computational approaches to language have generally treated the two processes separately. This is largely because computer models must include effective algorithms for whichever of the two processes they simulate; hence the task of investigating both problems at once is inconveniently large. Also, there is probably a greater conviction among computational linguists that the two processes have substantial differences, than there is among theoretical linguists. The justification for this approach is that similarites between the two processes are a further topic for exploration. Common sub-processes are to be found empirically, and we do not start from the assumption that the whole processes differ only trivially.

Chomsky has often stated that any human language must be subject to constraints on what it is possible for the human mind to "know", in some sense. It should not be forgotten that whatever "know" is taken to mean here, it must cover the constraint that language must be able to be produced and to be recognised. The assumption in Chomsky (1965) seems to be that the latter constraints are both obvious and trivial (e.g. the limits on multiply centre-embedded structures), but this is not necessarily true. Hence some characterisation of the production and recognition processes is just as fundamental to the study of linguistic universals as the investigation of "deep structures".

The focus of the research reported in this thesis has been on the recognition process, but some attempt has been made to separate out generalisations in the form of rules, so that frequently-occurring mechanisms are explicitly emphasised. However, there has not been time to formulate any bi-directional rules,

although there are some parts of the description which obviously ought to be shareable with a production-oriented model. The detailed specifications of many of the rules are (ultimately) in programming code - very much a one-directional, algorithmic form.

Section I.3 : Terminology and Grammaticality

In later chapters, expressions such as "noun phrase" and "subject" will be used frequently in discussing examples of English grammar, but this does not mean that these terms are constructs in computational grammar. "Noun phrase" is a term which is extremely useful for describing English informally, but which is very difficult to define rigorously. Some traditional concepts (e.g. "verb") are used directly in computational grammar; others (e.g. "subject") could be defined if necessary, although they are not used at present; some (e.g. "person" and "number" markings) are not used in the standard way. Nevertheless, these traditional expressions are so well-known that they provide the best expository vocabulary for describing English informally, and they are employed for that reason.

Chomsky (1957) used the word "grammatical" to refer to a string of words which a native speaker accepts as belonging to his language; strings which are not in the language were "ungrammatical". In 1965, he introduced a further classification - a sentence could also be classed as "acceptable/ unacceptable", depending on whether or not it was easily processed (under "performance conditions"); also, a sentence could be classified as "semantically normal/ semantically anomalous", depending on the well-formedness of the meaning expressed. Chomksy made the assumption that these three dimensions of classification were relatively independent (apart from the fact that grammaticality was a pre-requisite for the other classifications to be made). He also assumed that it was easy enough for the linguist to say which native-speaker judgements pertained to which dimension.

Later papers in semantically-based grammar (see Fillmore and Langendoen (1971)) use the asterisk (the traditional Chomskyan sign for an ungrammatical string) to condemn sentences (or strings) for a wide variety of reasons. McCawley (1971) uses [*] to mark "presuppositional oddity", and the device of inserting one or more question marks to mark degrees of "oddity" is very common in linguistic literature.

What emerges from these post-1957 develoments is that the "oddity" of a putative sentence (as judged by a native speaker) is graded not just along one dimension, but along several. Also, despite Chomksy's original assumption, neither the native speaker nor the linguist may have clear intuitions about which dimension(s) he is using to classify a string as "odd". It is the task of the linguist to construct a theory in which "odd" strings are classed as ill-formed in some respect (if the theory is to model human behaviour), but the judgements have not been rigidly pre-sorted into boxes labelled "perceptually difficult", "semantically anomalous", etc.

This is a suitable point to introduce some informal terminology. The term grammatical will be used not in the Chomskyan sense, but in the sense of "pertaining to grammar"; its contrary will therefore be non-grammatical, rather than "ungrammatical". An utterance which sounds, in some way, strange to a native speaker, will be classed as odd; the contrary to this is acceptable. Both "odd" and "acceptable" are classifications of the data, with no assumptions about the type of strangeness involved; both may be context dependent, since the judgement may be of the form "sentence S would

sound strange in context C". An utterance which would sound strange in virtually any context may be referred to as ill-formed (this is the nearest equivalent to Chomsky's "ungrammatical" that will be used). An utterance whose strangeness results entirely from contextual effects (rather than from its internal properties alone) will be termed inappropriate. A string which is deemed, for some theoretical or linguistic reason, to be abnormal in some way will be called anomalous. That is, "inappropriate" and "anomalous" refer to the way that the linguistic theory or grammar classes the item, not to the judgements about the original data.

These do not constitute precise definitions of the way that these words will be used, but they should clarify the informal usage in later chapters.

The aim of a grammar (including both semantic rules and recognition rules) is to specify how a sentence or phrase can be converted to semantic representation. Since there may be several possible ways of performing the conversion, we must introduce more detailed evidence concerning the way that it is done. The assumption will be made here (following Chomskyan linguistics) that "odd" sentences result from rule-failures of some kind, or from abnormal structuring. That is, there should be a direct correspondence between anomaly (something going wrong in processing or in the final structure) and oddity (sentences sounding bad). Adding this criterion to the grammar-writing methodology enriches the information available concerning the details of English grammar (see Section I.6) and has non-trivial consequences for producing linguistic analyses (see also Section III.9).

Section I.4 : Computer Modelling

Most of the time spent on the project reported here was spent constructing and debugging the program described in Chapter IV. It is therefore necessary to state why the program was written. None of the justifications are novel, but it is worth pointing out which of the possible arguments were thought valid and which are regarded as irrelevant. Most of the reasons are of a practical nature, rather than being strong theoretical justification.

In addition, one practical disadvantage of writing a full program, in a project like this should be mentioned. Program-writing and debugging is time-consuming, and usually introduces many implementation problems which have no theoretical content. This means that there is a "law of diminishing returns" concerning what should and should not be programmed, in that a section of program is worthwhile only if it will demonstrate some useful point, or will in some way enlighten the programmer himself. Unfortunately, often this can be determined only after the programming has been done.

One advantage of expressing ideas in program form is that one can build on an existing descriptive language. In trying to describe some complex subject matter where there is not an established theory, problems of notation quickly arise. It is difficult to make statements which are detailed and yet comprehensible to the reader, if the descriptive system is completely new and ad hoc. To some extent the widely-used transition network notation (see Section II.8) has helped the situation in computational linguistics, but there are still uncharted areas. If the model-builder uses a programming

language, he can at least express his constructs in primitive terms which will be understood by those who know that language. If, for example, the way that some "mapping" operates is slightly obscure, the corresponding code can be examined to provide an alternative description.

A computer also acts as a stern (if somewhat undiscriminating) critic. To have a machine take all of one's half-formulated ideas to their logical (if absurd) conclusion is an edifying (if somewhat humbling) experience; to see the intermediate stages of this development (by tracing the steps of the program's execution) is positively enlightening. In this way, interactive debugging of a computationally-expressed grammar develops a strong intuition for the details of a language. It also forces the grammar-writer to be rigorous and precise in the expression of his rules - the device has yet to be designed that takes hand-waving as its input.

However, perhaps the biggest advantage of using a computer to test one's ideas is the complexity of model which can be handled. It is very difficult to keep track, on paper or in one's head, of how various rules and different sub-components interact, so that non-computational models must remain either fairly simple or wholly unchecked.

One of the biggest contributions that computer modelling has made to the ideas expressed in this thesis is in the range of concepts it has provided. As noted in I.2, one obvious way to express language comprehension is in the form of steps that must be performed on an input sentence. Computer programming is based on algorithmic description, and hence provides a natural way of

expressing a series of instructions which must be performed. There are also certain computational concepts (subroutines, registers, assignment, etc) associated with this form of description, which are useful notions to use in describing a process. The augmented transition network notation, for example, is really just a form of programming language, with primitives which are particularly useful for describing the task of processing English. and many of its aspects are just re-formulations of ordinary programming devices. (see Winograd (1972, pp.44-46), and Ritchie (1977)).

It is also worth mentioning some possible aims or assumptions that are not relevant to this project, although they might apply to other computational approaches to English. A possible aim of using computers in linguistics might be to allow the use of English as a high-level programming language. This is not only not the aim of this project, it is a somewhat misguided aim. The reason English works so well as a vehicle of communication is that people are extremely clever at using it. Partial utterances, oblique allusions, etc., are frequently used in everyday speech in a way which would be hopelessly inefficient if one was not talking to a very sophisticated understanding device. In situations where efficient communication is required between humans (for example, between aircraft) stylised forms are often used. If we had a machine which was as intelligent and subtle as the human hearer, English might be a feasible means of communication - but it would still not be the best. (See Longuet-Higgins and Isard (1970) for discussion of some of the difficulties).

Also, there is not a significantly useful similarity between the structure of English and that of a progamming language, although inspection of certain program texts in COBOL, POP-2, etc., (full of "IF...THEN...ELSE", "MAKE", etc.) might suggest to a naive reader that there is no essential difference between the two. This is not very helpful, as the differences between English and a programming language are greater than the similarities. Many of the similarities are superficial, and often result from the attempts of the language designer to make the text understandable to humans. "IF....THEN...ELSE" is more transparent than (say) "TEST...OPTION....DEFAULT", but it should not delude the reader into thinking he is reading English. Ambiguity, use of pronouns, lack of a clear notion of syntactic well-formedness, the ability to be self-referential, are all features of English which are generally absent from (and perhaps even undesirable in) programming languages. This is not to say that it is impossible to build a language which has some, or all, of these features - but this would be narrowing the gap by constructing an unusual "programming language", rather than showing the gap to be small already.

For some time, artificial intelligence was a field in which the main research activity was writing programs. Sometimes no separate statement of theories or suggestions was forthcoming along with the program. If challenged, a writer might reply that his program <u>was</u> his theory - that the very code embodied clear hypotheses about intelligent behaviour, which could be assessed by running the program. Such a claim cannot be taken literally, unless the LISP programmer intended to assert that intelligence is formulated in CARs, CDRs and CONSs, or the FORTRAN programmer meant that

non-recursive subroutines were fundamental parts of his model. Clearly there are some parts or aspects of a programmed model that can be taken to constitute the substantive proposals, and some which are merely implementation details. Fortunately, this deficiency is being gradually remedied, with A.I. workers being more explicit about what they are suggesting, rather than dumping a mass of code and output in the lap of the reader. Although the implemented version of a model is the only one that can actually be regarded as "tested" in the fullest sense, it is still necessary to be clear about which aspects of it are to be taken seriously. (See the exchange between Simmons and Giuliano, following Simmons (1965)).

Allied to the approach just mentioned is the attitude that any program which actually works and produces impressive output must be of value, and that any ideas which cannot be directly expressed in a working program are meaningless hand-waving. Probably no one holds this view in such an extreme form, but it is a detectable thread in some discussions in artificial intelligence. This yardstick would make ELIZA (Weizenbaum (1966)) the most important language model so far produced, and dismiss as worthless all of theoretical linguistics (including many of the ideas of Schank (1969, 1970) prior to their being programmed). Although the first part of this attitude is slightly absurd, it cannot be denied that much of the impact of Winograd (1972) thesis came from the sample dialogue.

Nevertheless, even people who hold very strongly that ideas must be programmable, tend nevertheless to criticise or praise ideas on grounds (e.g. generality, elegance, intuitive appeal, etc) other than whether they have been (or can be) programmed.

Ultimately, ideas can be assessed in various terms, and the existence of an implemented program is just one (important) criterion among several.

An even stronger version of the "working program" attitude just described further stipulates what kind of performance the working program should have. Since one of the essential uses of language is to carry out conversations, it is a very rigorous test of a linguistic model if it can describe precisely the process of human dialogue. In fact, Schank (at the NATO Advanced Study Institute, Santa Cruz, 1975) commented that modelling conversation was perhaps the hardest task facing researchers in language processing. It is somewhat unreasonable, in the current state of the field, to demand of any computational linguistic hypothesis that it be demonstrated within a fully conversational system. Many interesting theories and programs have been produced which would not meet this criterion. (Again, ELIZA would score disproportionately well on this count).

Section I.5 : Detailed Analyses

Another justification for writing the MCHINE program (described in Chapter VI) is a general argument which applies to linguistic description whether done with pencil and paper or implemented on a computer. It is important not to lose touch with the real details of grammatical phenomena, since any hypothesis should derive from natural language observations, and must fit future observations. It is easy to become intoxicated with strong hypotheses and seduced by loose conclusions, and not devote enough time to seeing if the claims are even half-true. This is not to say that strong, false hypotheses are worthless - the Popperian idea of learning by refutation is relevant - but we must first find that they are false, not assume them to be true.

This is not to suggest that linguistic investigation proceeds in the fashion outlined by a textbook on scientific method - hypothesis construction, experiment design, experiment execution, confirmation or refutation. The whole process is much less tidy than this, with partially-formed ideas being tried out on small collections of data, only to be modified in the light of what is found, and then applied again to more items. There is a constant iteration between hypothesis-formation and testing, with concepts growing organically (and often quite patchily) as a result. Neatness can often be achieved by working apart from the data for a while, but one always has to return to the examples eventually, whether at the computer terminal or the desk.

Certain ideas which were held at the start of this project have been found, on examination of English data, to be incorrect. For example, the idea of semantic rules which operate incrementally as a sentence is processed (see Section III.3), the definition of a "constituent type" as a state and a list of registers (see Section III.7), and the idea of selecting semantic rules by examining the semantic items so far found (see Section III.9) all proved to be more complicated (or less adequate) than was first thought. Similarly, the hypothesis that a sentence-analyser used only semantic information (cf. Riesbeck (1974)) and that much of the processing could be performed locally (cf. Marcus (1975)) were explored at an early stage, without success. This represents a gain in knowledge which might not have been achieved without an attempt (not necessarily in program form) to follow up the full implications of these notions.

Section I.6 : Psychology, Linguistics and Artificial Intelligence

Linguistic theories frequently occupy an ambivalent position with respect to psychology. One one hand, language is an essentially human activity, and the only evidence for the linguist is the way humans use language, (where "use" is taken in a very broad sense, to include, for example, the intuitions and judgements employed as data by transformational grammarians). On the other hand, linguists do not generally claim to be carrying out psychological experiments, and their notion of "empirical" differs radically from that of an experimentalist. This dual position was highlighted by Chomsky's controversial attempt to distinguish between "competence" and "performance". (Chomsky (1964, 1965)). Generally, linguistic descriptions have been what Chomsky would regard as "competence" theories, since linguists have usually aimed at describing patterns in linguistic structure, without specifying processes for producing or interpreting sentences. Hence the methodological stance of linguists has been fairly consistent - they merely attempt to describe, in a theoretical fashion, the nature of linguistic structure; how this would actually be used by a human speaker or hearer in a real situation is another matter, to be investigated by psychologists. Unfortunately, anyone attempting to design a "performance" model is in a more difficult position, and it is desirable to clarify here some of the aims of the framework described in this thesis.

The partial model presented in later chapters is primarily a linguistic model of performance, with an obvious but wholly speculative relationship to a psychological model. This approach was adopted largely because it seemed intuitively to be more productive, but it can be justified in greater detail (see Section I.2). The main point is that it is observable that people do interpret and produce sentences (subject to philosophical provisos which are too complex to discuss here); hence it is reasonable to investigate language in terms of two analogous processes. There is no guarantee that any regularities or constructs will be formed in language outwith these processes (e.g. Chomskyan "competence"), so to search for such non-processing patterns may well be fruitless.

The evidence used in constructing the model has been wholly linguistic in nature, since it consists of considering intuitive judgements about the use of language (oddity of utterances, difficulty in understanding of sentences, notions of similarity or difference in meaning, etc.). The main advantage of using such "evidence" is its potential richness. If we allow judgements of synonymy, oddity, etc., as data for our model-building, we will be able to fill in many details which are not easily amenable to treatment by experimental technique. The linguistic style of argument (largely developed within transformational grammar) yields a mass of "information" of debatable reliability, whereas the experimental method gives strongly-supported results in minute areas.

This raises another point - the need for a fairly full model. Psycholinguistic evidence does not yet constrain the set of possible theories far enough to tell us much about it; stipulating that the

theory should be detailed, coherent and fairly consistent does, however, provide a fairly stringent set of constraints. In fact they are probably enough to rule out all current linguistic theories or frameworks, including the one outlined later in this thesis, but these properties can be regarded as goals that the ideal theory should aim at. They have the advantage that they can be applied at every stage in theory construction without recourse to experiment.

If we ever get near to having a full model, its viability as a psychological theory becomes relevant. This does not imply that even a detailed model is easily tested within the experimental paradigm, since it is not a trivial matter to state how constraints in the model should reveal themselves in measurable psychological parameters. For example, certain experiments (see Carey et. al.(1970), Garrett(1970)) suggest that when people are faced with a potential structural ambiguity while listening to a sentence, they consider all possibilities. This seems, at first glance, to support a "breadth-first" interpretation model rather than a "depth-first" one (see Section III.6). However, further consideration shows that this is an over-interpretation of the results. The results really show that when a hearer is faced with a multiple choice, he requires more processing time - it shows nothing about how he uses this extra time, and so does not distinguish between two models in which all options are attempted in some way; it is also compatible with a strategy in which a careful selection of one option is made after some estimating process.

Having stated these disclaimers, some questions remain concerning what the model in this thesis is a model of. It is not a model of some abstract "competence", but it cannot be claimed to be a psychological model of a human. This dilemma is not unique to linguistic performance models, but is a common aspect of much of artificial intelligence. The main justification that can be offered for artificial intelligence models, from a psychologist's viewpoint, is that, while not directly psychological, they may furnish suggestions and concepts for an eventual psychological description. (This is also true of linguistic models which, like Chomsky's, aim to form part of an eventual performance model). The disciplines of cognitive psychology and artifical intelligence now overlap, as some psychologists recognise the need to work out full models of a fairly speculative nature before attempting to seek experimental support.

The framework in later chapters forms a partial linguistic model of the English language. It is phrased in terms of "processing" primarily because this seems to be an obvious way to look for patterns in language. It includes the concept of a "model hearer", which is a device capable of performing certain operations on linguistic structures. The model hearer includes certain subparts (e.g. an "analyser") which deal with specialised tasks in these operations. Occasionally the notion of a "program" will be referred to. A computer program is intended to be an implementation of the model hearer, which may serve to clarify certain points of detail, but it is not the central descriptive device (see Section I.4). The data to be used in sketching the model hearer include various intuitive judgements about English sentences, their meanings and about English dialogues. This follows the principle that all we know

about the workings of language is what the language-users can tell us about it; this principle is as old as field-work in linguistics, if not older. A fairly agnostic approach will be taken towards judgements of "oddity" (see I.3), in that as few assumptions as possible will be made about the cause of the oddity (see also Section I.2.2).

CHAPTER II

OTHER FRAMEWORKS

Section II.0 : Preamble

Work in artificial intelligence tends to impinge on the fields of linguistics, psychology and philosophy, so there are relationships (of varying strength) between the work reported in Chapters III, IV, V and VI and a wide variety of other investigations of the English language. No attempt will be made here to give a complete guide to these other frameworks, but the aspects most pertinent to later investigations will be discussed. The more detailed arguments of Chapter III will also incorporate comments on the other frameworks, particularly those most relevant to computational grammar.

It is noticeable that most of the schools of thought that will be examined here are "frameworks" in the sense of Section I.1, rather than theories. Generally, linguistic research operates with a cluster of ideas and guidelines (which often are not related by any necessity), rather than some monolithic theory with clear boundaries. For example, it is hard to say what are the defining principles of the "conceptual dependency" of Schank (1972). It is fundamental to conceptual dependency that meaning can be described by a small set of primitive elements which are not language-specific. However, Schank's work has associated ideas and methods which are not logical conclusions from this assumption - e.g. the use of a particular set of primitives, the avoidance of traditional syntax, etc. Someone could therefore adhere to the central tenet (meaning description in non-linguistic elements), but still not be working on conceptual dependency as it is normally recognised. Some frameworks are even harder to define precisely, although all are clearly recognisable to

their users. Katz explicitly states (1972, Chapter 8) the defining principle of deep interpretive semantics, and discards as peripheral certain notions often attributed to him. His minimal definitive statement might surprise some linguists, since it contains so little of the framework that has come to be associated with interpretive semantics.

The discussions in this chapter will therefore deal with the loosely linked packages of ideas that are normally taken to characterise the various viewpoints, rather than attempting to isolate and examine defining properties. A framework exists in the minds of the users, rather than in explicit definitions, unfortunate though this often is.

Sections II.1 to II.5 describe the framework that provided the background in linguistics and logic for computational grammar, Sections II.6 to II.9 outline the artificial intelligence influences on the framework, and Sections II.10 and II.11 summarise work which was carried out independently of this project but which has certain similarities to it.

Section II.1 : The Aspects Model (Chomsky)

One of the most influential linguistic theories of recent years has been transformational grammar as developed by Chomsky. (Chomsky (1955, 1956, 1957, 1958, 1959, 1961, 1964, 1965, 1966)). The remarks here will be addressed to the more complex 1965 version, rather than the 1957 prototype.

Chomsky proposed a model of grammar in which there were separate (but closely related) syntactic, semantic and phonological components. (The phonological component (see Chomsky and Halle (1968)) is not of interest here). Although some work was done on the semantic component by others (see Section II.2), Chomsky's own suggestions were primarily concerned with the syntactic component.

There were to be two main kinds of syntactic rules - phrase structure and transformational. The former characterised a set of labelled trees, and the latter defined a set of tree-manipulating operations. Each sentence was analysed as having a surface structure (a labelled tree whose terminal nodes represented the words of the sentence), a deep structure (a labelled tree describable by the phrase structure rules) and a derivation (an ordered subset of the transformational rules, which defined the relation between the deep structure and the surface structure). The deep structure was supposed to be the level at which certain linguistic generalisations (e.g. categorisation of verbs) were represented, and formed the interface with the semantic component (in that the semantic rules were defined on phrase-structure-generated deep structures). Ambiguous sentences, therefore, might have more than one associated deep structure, thus

providing more than one possible input to the semantic component.

One major asset of this model was that it was a fairly detailed and complex system, which attempted to systematise a wide range of phenomena. Certain problems and linguistic patterns were simultaneously drawn to the attention of linguists, and described in an elegant fashion. This stimulated a great amount of work which uncovered a vast number of regularities in natural language (especially English). The resulting literature provides a vast store of linguistic patterning which subsequent theories can look on as partially-digested data.

Over recent years, many criticisms have been aimed at Chomsky's work, but only the three most relevant to this project will be mentioned here. Firstly, the model was not one of language-processing (despite a tentative suggestion at the end of Katz and Postal (1964)), but was a static description of patterns. (See Section I.2.3 for a discussion of this issue). Chomsky (1965, pp.30-31) seems to imply that the mode of employing transformational rules to comprehend language would be some kind of general rule-interpreter, but no details are given of how such a mechanism might work.

Secondly, the model was dominated by syntactic generalisations. Although deep structures were supposed to provide the interface with the semantic component, the criteria for postulating particular deep structures and transformational derivations were almost always based on syntactic patterns. The assumption seemed to be that if the syntactic generalisations were captured, the semantic rules would work correctly (see Section II.2 for further remarks). Thirdly, (as a

consequence of the two preceding points), the model was disproportionately dependent on the separation between ill-formed and well-formed sentences. Instead of this providing just one criterion to apply, it formed the central task of the grammar, with the latter being wholly syntactic. (See Sections I.2, I.3 for more discussion).

Section II.2 : Deep Interpretive Semantics (Katz)

Katz and Fodor (1963) sketched a semantic theory which was intended to provide a semantic component suitable for the linguistic model being developed by Chomsky (see Section II.1, above). They provided some simple examples of how their model was to operate, but not all the details were explicitly stated. Katz and Postal (1964) elaborated on this theory, and suggested how the semantic component might interface with a syntactic component in the style of the "Aspects" model. This semantic theory was gradually developed in a series of articles (Katz (1967, 1970, 1971) and the fullest description so far is contained in Katz (1972). It is this latest version which will be discussed here.

Katz maintains the position that the relationship between syntax and semantics is as follows. For each sentence, there is an associated deep structure, in the form of a labelled tree, related to the surface structure by a sequence of syntactic transformational rules. (See Section II.1). There are semantic interpretive rules which relate each deep structure to a semantic representation. The semantic representation for an item consists of a set of semantic markers. Other forms of information in Katz's model are held in redundacy rules, which impose a hierarchical organisation on the markers, and antonym sets, which cluster mutually exclusive markers. These latter two devices seem intuitively to capture generalisations, and are a useful contribution to the problem of defining and describing "semantic anomaly".

A semantic marker is not (as it appeared from the original Katz and Fodor formulation) an unanalyseable unit (like a syntactic feature), but can have internal structuring. That is, "semantic marker" is roughly equivalent to "piece of semantic structure", with no atomic connotations. Hence the 1972 version allows a wider range of operations to be defined on markers, whereas the 1963 theory permitted little other than ordinary set operations (union, intersection, etc) on the sets of markers. This can be regarded as an improvement or a retrograde step, depending on the metatheoretical standpoint, but from the point of view of building working models, it is a useful enrichment for exploring the complexities of natural language semantics. Unfortunately, Katz uses a semi-formal notation for his markers, so it is hard to determine all the details. For example, it is not clear exactly how the internal structuring of a marker like

(EVAL : seat for one)

can be operated upon, or what elements are atomic in his system. Also, it may be that Katz has not taken advantage of this potential enrichment, for the following reasons. In the 1963 and 1964 versions of the theory, there were <u>projection rules</u> which showed how the meanings of the immediate constituents of a syntactic item were combined to form the meaning of the whole. There were different projection rules for different syntactic constructions, and so different rules could combine meanings in different ways (subject to the proviso that the items they were acting on were unordered sets of atomic elements). Katz suggests that such rules can be removed from his theory and their function performed by syntactic annotations on

each semantic lexical item, showing how other items may be attached to or combined with it. He illustrates this by describing how the lexical meaning of a verb can have annotated "slots" showing which syntactic constituents will provide the meanings to be inserted in the meaning of the main verb. It is not clear how this approach will work in general; presumably the lexical meaning for a noun will require annotations showing where all possible modifiers can be attached. Possibly only nouns and verbs will need these markings, and other items can be dependent on them (cf. dependency grammar, Robinson (1970)). This seems a little cumbersome - having semantic items carrying details of all possible combinations that they can undergo, including syntactic details of the way that their associated items will be expressed (cf. comments in Section IV.8 on where to represent information). Also, it restricts the modes of combination, unless the marking system is to be expanded in some way to allow the markings to be lists of different ways of combining. In Katz's verb example, there is one mode of combination (roughly equivalent to filling in a case-frame, in the model of Fillmore (1968)), so no more needs to be specified. This seems to be inherent in the mechanism, whereas the 1963 version allowed the projection rules to describe different semantic patterns in terms of different inter-constituent relationships. Abolishing projection rules seems to tie the linguist's hands, and eliminate one of the more useful parts of the theory (cf. Montague's rules (Section II.5)).

The annotations on semantic items can only specify one syntactic form that related items may have, so this innovation is, like most of Katz's theory, heavily dependent on having a canonical syntactic form (deep structure) for the semantic rules to operate on. This

dependence means that any simplicity or elegance that Katz's rules achieve is conditional upon the syntactic component being able to define the requisite deep structure. Whereas Chomsky made the assumption that a grammar which captured syntactic generalisations would define the appropriate deep structures for the semantic component, Katz makes the assumption that the syntactic component will provide whatever structures are needed for the semantic rules. Such assumptions are unavoidable in a modular model (this is how some complex computer systems are written, for example) but great care needs to be taken to ensure that the interfaces are compatible, and that simplification on one side of the line does not cause complication on the other.

Section II.3 : Surface Interpretive Semantics (Jackendoff)

One development of the "Aspects" model was the introduction of semantic rules which operated on syntactic structures other than deep structures, as proposed by Jackendoff and others. (Jackendoff (1968, 1973), Dougherty (1969), Bresnan (1970)). There are certain aspects of this approach which bring it slightly closer to computational grammar than the original Katz semantic theory. Unfortunately, there are many unexplained details in Jackendoff's proposals (as has been observed by opponents of surface interpretive rules) so it is hard to deliver an overall verdict.

Jackendoff suggests the use of a wider variety of semantic representations rather than just sets of semantic markers (e.g. "coreference tables", a "thematic hierarchy", etc.). This is an important step, since it frees the linguist from the straitjacket of having to capture every aspect of the meaning of a sentence in a set of markers. It seems plausible that coreference can be incorporated more easily in a model of language use if it is explicitly handled as part of referential semantics rather than being squashed into syntactic rules. It is not quite so easy to see how benefits will accrue from Jackendoff's notation for opaque contexts, since no mechanisms are outlined (e.g. inference) which use the notation.

As observed in Section II.2, deep interpretive semantics depends on a fairly sophisticated syntactic component to connect deep and surface structure, and so presents problems for a model of language understanding. Surface interpretive semantics replaces many previously-formulated syntactic transformations with semantic

interpretation rules, and so makes a start on the problem of determining semantic representation from surface form. However, it is only a partial step, since very few of Jackendoff's rules operate directly on the surface structure. The rules usually operate on some structure intermediate between deep and surface, and so it would be more appropriate to call them "derived structure interpretation rules". From the point of view of trying to model language understanding as a process, this introduces as many problems as it eliminates (if not more), since the analyser would have to perform semantic processing at arbitrarily interleaved parts of the syntactic processing. That would not be impossible, but it requires a more complex processing model than the term "surface interpretive semantics" might suggest.

One issue which becomes rather confused in the course of Jackendoff's arguments is the question of "syntactic well-formedness" (or "grammaticality" in the Chomskyan sense). Although he adheres to the position that there is a notion of syntactic well-formedness which is independent of semantics, his proposals regarding undeveloped nodes lead to rather bizarre strings being classed as syntactically well-formed. Jackendoff admits that his notion of well-formedness has changed from the original Chomskyan idea in certain respects, but does not regard this as a problem. His position is quite consistent (and it emphasises the point made in Section I.3 that the kind of well-formedness (or ill-formedness) that a string has is defined by the theory, rather than being given in the data), but it raises anew (without fresh debate) the question of what role syntactic well-formedness is to play in the theory.

Section II.4 : Generative Semantics (Lakoff, McCawley, Postal)

Another development stimulated by Chomsky's work in the mid-1960s was that of "generative semantics" (Lakoff and Ross (1967), McCawley (1968, 1971a), Bach (1968), Postal (1972)). This framework retained certain parts of Chomsky's model (in particular, the notion of a transformational derivation of tree-structures) but discarded the idea of a separate syntactic component. A generative semantic derivation related the semantic representation (a tree of predicates, arguments and propositions) to surface form by a uniform sequence of transformational rules, with no intermediate level of syntactic deep structure. A later development (see Lakoff (1971)) was the incorporation of global derivational constraints, which were general rules with the power to influence the operation of transformational rules in any way.

The initial tenet of generative semantics (that it is undesirable to posit separate syntactic rules and structures) is shared by computational grammar, but there are few other similarities. The main difficulty in assessing the advantages of the theory (as opposed to the attractiveness of its assumptions) is that it is very hard to assess whether a given solution falls within the predictions of generative semantics, since global derivational constraints allow any manipulation whatsoever, and the concept of "transformation" has changed so much. As observed in Section I.1, most theories include theoretically powerful devices, restrained only by the idea of "ad-hoc-ness", but this theory seems to allow any "rule" to be inserted without the linguist's conscience complaining.

Also, since there is no good independent definition of "syntax" or "semantics" (a frequent problem), it is difficult to see whether the hypothesis of there being no wholly syntactic rules has been verified. (These issues have beeen covered very fully elsewhere - Partee (1970, 1971), Lakoff (1970), Chomsky (1971, 1972), Postal (1972), Katz (1970)).

The methodology adopted in Chapters III, IV and V is that independent syntactic constructs should not be postulated unnecessarily, but that the linguistic description (or the notion of "semantic") should not be distorted to protect this guideline.

Section II.5 : Montague Grammar (Montague)

At present, one of the few formally defined frameworks for describing language is that of Montague (1968, 1970a, 1970b, 1972), which is unparallelled in its rigour and precision.

Montague's system contains many traditional notions in a mathematically formalised form. The syntactic rules are very like phrase-structure rules (see Chomsky (1957)), in that they characterise a tree-like structure with words as terminal elements. Most of the syntactic terminology is merely a tidying-up of notions like "lexical entry" and "phrase". Syntax and semantics are defined separately, but there is a well-defined interface in that each syntactic rule has an associated semantic combining rule. This is closely related to the Tarski semantics for predicate logic, and is also very similar to the Katz-Fodor system of "projection rules" (see Section II.2 above). The semantic concepts are related to formal logic, and provide a very comprehensive generalisation of notions like "sense and reference" (Frege(1892)) and "indexical expression" (Bar-Hillel (1954)).

The main advantages of Montague's work are its precision and generality. Also, it has provided a start to illuminating such difficult and confused areas as the distinction between categories of meanings and categories of objects referred to, and the way that meanings can vary systematically with changes in the context of use.

The main disadvantages of Montague's work are that it is not a process model, and leaves no place in the theory for any form of production or recognition rules.

There seems to be a feeling within artificial intelligence that Montague's semantic system is based too strongly on traditional, Tarski-style truth-values. For example, Wilks (public lecture at Edinburgh University, 1976) stated that one deficiency of Montague's ideas was that it assigned the same "meaning" to "Snow is white" and "Two is prime", namely "TRUE". This misrepresents Montague's theory, which provides two distinct intensional meanings for these two sentences, with the same extensional meaning in some worlds. Also, it is currently fashionable to talk in terms of describing sentences in terms of their use in a dialogue. It is important to notice that the latter does not necessarily contrast with a truth-valued system of meaning, unless a model of conversation can be developed which genuinely uses a different method of describing conversational effect. For example, the SHRDLU, LSNLIS and MCHINE programs (see Sections II.8, II.9 and Chapter VI) all treat a yes-no question as requiring a pattern (the "meaning" of the sentence) to be tested against the "hearer's" model to see if that state holds (or held at some time). This is very much a truth-value system, although improved by making "truth" relative to each hearer or speaker.

The MCHINE system uses many traditional syntactic and semantic ideas, and Montague's writings have been particularly useful in clarifying certain distinctions (particularly in semantics) which are often not considered elsewhere.

Section II.6 : Preference Semantics (Wilks)

One of the most distinctive approaches to language-processing in recent years has been that of Wilks (1972, 1973, 1975). Although the implemented versions have generally been aimed at machine translation, it is worth considering whether any general principles of language-processing can be extracted from the ideas he uses. The form of "interlingual representation" used is influenced by the translation task, so it is not very relevant to try to assess its adequacy for general purpose meaning representation. However the processing strategies are interesting, since Wilks claims that they are predominantly semantic, rather than syntactic, and are capable of processing metaphors and other non-standard constructions.

A preference semantics grammar can be thought of as being in two sections - structures and procedures. There are three principle kinds of structures - primitive elements (like Katz-Fodor "semantic markers"), formulae (corresponding to lexical entries) and bare templates (corresponding to structural rules or patterns). Each word has an associated formula, which consists of a binary tree of primitive semantic elements, with one element designated the "head". A bare template is a triple of primitive semantic elements representing three formulae-heads to be sought in the input words.

The procedures are many and varied. The input string is first put through a "fragmenter", which clumps the words into small phrases ("fragments"). There is a matching routine, which tries to find a bare template to match each fragment, inserting "dummy" elements where necessary to complete the match. The bare template together

with the elements that matched it are called a "full template". Then various routines (in the 1973 paper, referred to as EXTEND and TIE) attempt to find links between the filled templates so far built; in later versions of Wilks' system, further structures called "paraplates" help to perform these connections. The notion of "preference" is introduced because the routines which fill out connections between structures will not discard a structure simply because its internal semantic specifications are not met - however, a structure where all the specifications are met will be preferred to a deficient one, thereby resolving possible ambiguities on semantic grounds.

The main advantage that this system has is that it has been implemented, and has translated passages of English and French. The main difficulty in trying to assess the details of the mechanisms is that Wilks tends to highlight certain areas at the expense of others. His papers generally concentrate on the notions of "template", "formula" and "preference", and imply that most of the linguistic information is contained in the structures plus the simple notion of counting semantic connections. In fact, much of the work is done by the routines for matching templates and tying together structures, and information is encoded in a wide variety of forms. Although the central aspect of the system is supposed to be that it is based on semantic matching, and not syntactic structure-building, much of the processing (e.g. the "fragmenting" and "extending") is traditionally syntactic. Since the preferential semantic accounting occurs only once the building routines have made ties, the overall demarcation is not vastly different from that in the programs of Woods, Winograd, or Marcus (cf. Sections II.8, II.9, II.10), where semantic checking is

applied to the results of syntactic building. The main differences are that the initial template-matching is at least partly semantic, and that Wilks' semantic checking does not discard possibilities, but merely gives them a preference ordering.

As discussed in Section III.6 and VI.5, a form of "preference" facility has been implemented in the MCHINE program (although it has not been fully exercised).

Section II.7 : Conceptual Dependency (Schank, Riesbeck)

One of the more influential artificial intelligence approaches to natural language in recent years has been "conceptual dependency" as developed by Schank and others (Schank (1969, 1970, 1972a, 1972b), Riesbeck (1974, 1975), Goldman (1973), Rieger (1974)). The main aim of conceptual dependency is to provide a system of "conceptual representation" which can be used to describe the meaning of natural language in terms which are unanalyseably primitive and which are independent of the natural language involved. By constructing a vocabulary of primitive constructs, and rules for using them, Schank hopes to develop a single representation system which will serve for all language-processing and inferential tasks, e.g. machine translation, question-answering, etc. This seems a worthwhile aim, and the desire for generality is a great improvement over earlier artificial intelligence approaches to meaning-representation. As argued in Chapter I, some of the most interesting questions in language research concern the processing that must be done either to convert a string of words to the meaning representation, or vice versa. Although it is clear that Schank regards language comprehension as fundamental, his articles have tended to concentrate on issues regarding semantic representation, rather than processing.

Nevertheless, papers by Riesbeck (1973, 1974, 1975) have given some indication of how a conceptual dependency sentence-analyser is intended to operate, and this outline will be examined here, as it is the aspect of conceptual dependency most relevant to the MCHINE project. There are two slightly different proposals concerning

analysis strategies; the first is described in Riesbeck (1973, 1974: part I) and the second in Riesbeck (1974: part II, 1975).

Roughly the system is as follows. The central device is a "request", which is a "demon" - i.e. a test + action pair (cf. Charniak (1972)). Each word has an associated list of requests, and the analyser maintains a list of currently active requests. As each word is taken in, its requests are added to the list. The list is then scanned, and if the test part of any request yields TRUE, the action part of that request is carried out, and the request removed from the list. The second version of Riesbeck's system attempts to fill in more details of this process. "Expectations" (an alternative term for requests) are now annotated with information about what they do, in terms of which variable value is relevant to the TEST (or "FOCUS"), and which slot will be filled if the action is executed (the "NEED"). This seems to help in indexing the expectations, since the FOCUS device will make the implementation more efficient, but it does not impose any structure on how the expectations operate.

The main problem with Riesbeck's suggestions are that they say virtually nothing about the the details of language processing. All that he says about the form of the processing strategy, or the shape of grammatical rules, is that the grammar is in the form of "demons". This can be contrasted, for example, with the more subtle proposals of Marcus (1974, 1975) (see Section II.10), where the notion of "demon" is taken as a building block for making further detailed suggestions (rather than being left as the entire theory). Riesbeck asserts that his program treats syntax and semantics in a uniform manner and allows concepts of both types to communicate freely. This

is true, in as much as all information is programmed into expectations, but it could equally well be said that any program written wholly in LISP achieves the same uniformity - the whole problem is in elucidating how different information interacts, not merely expressing it in some common computing language. Demons (or expectations) are a wholly general computational device, and this kind of mechanism is a common implementation detail of language processing systems - what are needed are some rules about how these demons can or cannot be used.

Despite Riesbeck's claims, it is not the case that syntactic information is not used in his program. He merely puts it into expectation form and re-labels it. For example, the worked example in Riesbeck (1975) starts with the expectation "Is INPUT an NP ?" - a very appropriate (but syntactic) prediction. In the same example, the word "a" causes the action to be performed :

"save the current set of expectations and replace it with `6 - does INPUT end an NP ?` "

This seems to recognise the surface integrity of the traditional noun phrase, and the saving of expectations seems to be analogous to the nesting devices in syntactic systems (e.g. the "PUSH" in Woods' transition networks (see Section II.8), or PROGRAMMAR's "PARSE" (see Section II.9)).

This saving of expectations raises doubts about one of Riesbeck's other examples.

(4) John was mad at Mary. He gave her a sock.

He suggests that in (4), the word "sock" would be disambiguated as soon as the word "sock" was read (not when the whole NP was processed) using the expectations set up at the higher level. If "a" has caused the higher level expectation list to be temporarily shelved, it is not clear how this can happen.

Even Riesbeck's attempt to enrich his theory by annotating each expectation with a "NEED" and a "FOCUS" does not help much. He suggests that if expectations E2 and E3 are both active, and both would have to fill the NEED of E1, and if E2 "cannot produce a structure that will satisfy the predicate of the first expectation" (i.e. E1), then E3 is tested first. This is strange for two reasons. Firstly, if E2 "cannot produce a structure" that is suitable, why test it at all ? It could just be discarded. Secondly, where does this useful information come from ? Neither NEED nor FOCUS indicates what kind of structure an expectation-action will produce.

The NEED/FOCUS system can apply only to some of the manipulations performed by the analyser since, as Riesbeck admits, "Unfortunately, there are no constraints on what sets of functions can appear as predicates and programs", and arbitrary side-effects are allowed. In an attempt to broaden the notion of a NEED to cover a particular operation that he needed, Riesbeck (1974) distorts the original idea to the extent of making it vacuous. He argues that certain constructions can be handled by the "need" for all words in the sentence to be used up; this is hardly the name of a conceptual slot being filled.

Fundamental to the approach of Schank and Riesbeck is the idea that conceptual (not syntactic) predictions will provide the mechanism for analysing a sentence. They seem to over-estimate the extent to which predictions of conceptual meaning will constrain the possible input forms (which is what the analyser ultimately has to deal with). Suppose that the conversational situation facing the analyser (and accompanying semantic model) is as follows. The context (to talk in the terms of Riesbeck (1973)) is that of being in a bar, drinking. The input utterance starts as in (5).

(5) I like....

This utterance might continue in any one of several ways, for example those in (6). The analyser would have to process vastly different sequences of words in these cases, and the "context" does not help it much in this task.

(6)

(a) ...this pub.

(b) ...peanuts.

(c) ...McEwan's Export.

(d) ...that girl we met last night.

Even if the "context" were much narrower, say, the consumption of peanuts, the analyser would be given no help in working its way through the surface strings in (7), even though some of them have very similar meanings.

(7)

(a) ...eating peanuts.

(b) ...to eat peanuts.

(c) ...these peanuts.

(d) ...crisps as well as peanuts.

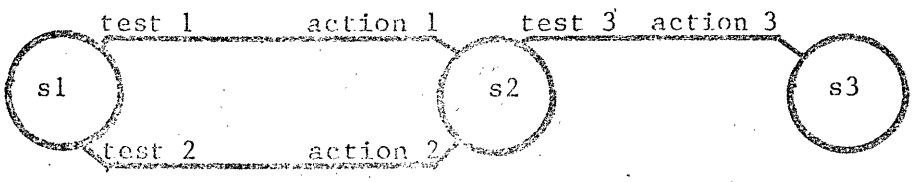
Schank and Riesbeck suggest that, while it is necessary for a theory of language to specify some way of converting surface strings to meaning representation, it is not important to specify that process in any theoretically interesting way. If a program can be written which produces the desired output from particular inputs, that is all that is required at present. As can be judged from Chapter I, this is a fundamental difference in attitude between computational grammar and conceptual dependency.

Section II.8 : Transition Network Grammars (Woods)

The LSNLIS program (Woods et.al. (1972)) is a major implementation of various ideas which contribute to the basis of computational grammar. In particular, the recognition grammar is written using a formalism which has become widely used in language processing, and which has been the basis of the mechanisms used in the MCHINE program, so it is worth digressing to give the background to this formalism (see also Bobrow and Fraser (1969), Woods et. al. (1969), Woods (1970, 1973)).

The kind of grammar Woods uses are known as augmented transition network grammars, or "ATN grammars" for short. (The abbreviations "RATN" and "AFSTN" are also used sometimes). The grammar is a standardised description of a sequence of tests and operations that have to be performed when a sentence is being processed. This can be represented graphically by a directed graph, where arcs represent test-action pairs and states (or nodes) indicate common points joining arcs, e.g. (8) (cf. Conway(1963)).

(8)

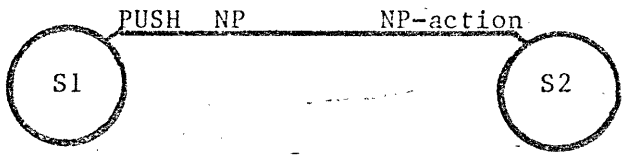


The tests are generally (but not necessarily) conditions on an input word and the network represents the possible ways of analysing a sentence. A complete analysis is produced by a path through the

network on which all the tests are satisfied by the input words (which are scanned gradually as the path is traced through the network) and all the actions are carried out in order. The "augmented" in the title refers to the fact that any arbitrary tests or actions can be included, making the ATN a totally general computational formalism.

One of the most important aspects of the ATN formalism is the mechanism for handling embedded constituents. Where a test on an arc simply checks the grammatical category of the next word, processing is simple, but if the network is to check for the presence of a particular kind of phrase, things become more complicated. There is a special kind of arc (called a "PUSH" arc by Woods) for including phrase-tests.

(9)



In a network like (9), the PUSH NP indicates that the category being checked (i.e. NP) may be made up of more than one word, and independent processing space is needed for it. The destination state (S2 above) is saved (as is the "NP-action", in some versions) and some other part of the network (indicated by the label "NP") is used to process the incoming phrase. If the phrase is processed successfully, another special arc (a "POP" arc) will be encountered in the NP network. Traversing a POP arc causes processing to be resumed in the saved part of the network (S2 in example (9)), with the whole phrase that has been found acting as the input item for the

action on the original PUSH arc (in (9), "NP-action"). This jump and return mechanism (analogous to a subroutine call in a programming language) can be invoked again while processing a phrase, so that constituents can be nested inside each other, to an extent limited only by the space for storing information about the return point.

The LSNLIS program contains a large bank of geological data (concerning moon-rock samples) and the syntactic-semantic components are designed to allow a scientist to interrogate this store of facts. Hence there is a slight bias in the type of sentence tackled, since questions and certain imperatives were the main types needed. There is a general semantic representation (related to predicate logic), which provides the necessary devices for searching the data-base for specific details (see also Woods (1968)). The system has a syntactic recognition grammar (written in ATN form) which converts input sentences to (approximately) a deep-structure representation in the sense of Chomksy (1965). This syntactic structure is processed by a set of interpretive rules into the semantic form necessary for examining the data-base. All conversion of syntactic form to semantic representation is performed at the end of a sentence, with no interaction between the two components other than this. The semantic interpretive rules are hierarchically organised, so that they are applied to the syntax tree as a whole and each rule can call other rules on the branch of the tree.

There are few significant differences between the overall organisation of the LSNLIS and the MCHINE systems. The main difference is that LSNLIS includes a full, separate syntactic component, which the semantic interpretive rules are geared to

handling, whereas the structure built in the MCHINE program is directly part of the setting up of the semantic rule hierarchy - the surface structure, in computational grammar is the interpretive rule sequence.

One other main difference is in the aims of the two programs. LSNLIS is a practical working system, using generalisations and patterns only to improve the efficiency of the overall service to the user. The MCHINE program, on the other hand, has been written only as a testing device for linguistic ideas, and the general aim is to experiment with linguistic descriptions and devices, in order to illuminate the structure of language. Woods (at the workshop on Theoretical Issues in Natural Language Processing, Cambridge, Mass., June, 1975) claimed that working with the goal of achieving practical efficiency will achieve the goal of linguistic discovery as a by-product, but this remains to be seen.

Section II.9 : The SHRDLU System (Winograd)

The SHRDLU program (Winograd (1972)) is one of the best known English dialogue programs of recent years. It is interesting here since it includes some of the devices to be developed and used in computational grammar. Winograd did not use the ATN notation, but instead designed a programming language PROGRAMMAR, which provides more or less the same facilities (he comments (pp.44-46) on the similarities of these two mechanisms). The overall organisation of the SHRDLU syntactic and semantic components is similar to that of the LSNLIS system, and the two programs are very close in their linguistic analyses (although Woods gives credit to Chomsky for the structures, and Winograd discusses Halliday's systemic grammar).

Syntactic classification is entirely in terms of features, where a feature is an unanalyseable marking. The written description of SHRDLU (Winograd (1972)) emphasises that these features are interdependent in ways that can be perspicuously represented by a particular form of graph (a "systems network"). Since Halliday (1967a, b, 1968) uses this notation, this has been taken to mean that Winograd is using Hallidayan grammar. This is slightly misleading, as the program itself does not use the systems networks - they are merely a good expository device for describing the relationships between the features. The MCHINE program also uses features, but at no stage have these been organised into systems networks; nevertheless, a close examination of the MCHINE program would allow a reader to draw up his own system network of how the features are distributed throughout the grammar. This would be true of any

grammar that employed features. There is more to Halliday's ideas than this part of the notation.

The lexical entry for a word consists of a list of features, and a semantic entry, the latter being a data structure which will act as the "meaning" of the word. The parser, written in PROGRAMMAR, builds a surface syntactic tree. Each node in the tree has a category label (indicating that it is a Noun Group, Clause, etc.), a list of features, and an associated semantic structure. The features are attached to the nodes directly by the parser, on the basis of the words it finds in the sentence (and other tests). The semantic structure is constructed by a "semantic specialist", which will be described below.

The PUSH/POP facility of the augmented transition network is replaced in PROGRAMMAR by the "PARSE" command, which takes a constituent label (e.g. NG) as an argument. Hence

(PARSE NG....)

is a command to try to parse a Noun Group (the other arguments "...." indicate what to do on completion or failure, using various standard options). Successful execution of this command should produce a subtree, which is then attached to the part of the syntactic tree currently being constructed. During the execution of this command, the line

(PARSE DET ....)

might be encountered. This is similarly a command to parse a

particular category (this time one word) and attach it to the tree. In this way, the tree-structure results from the hierarchical processing; parsing commands may be nested within each other to produce a correspondingly branching tree structure.

One exception to this left-to-right, top-to-bottom flow of processing is the use of "demons". Winograd suggests that the best way to handle a conjunction (a difficult construction for any parsing system) is to define a special type of lexical entry for "and". Instead of "and" having a small "semantic structure" which is handed statically to a semantic specialist, the entry for "and" contains a special program, which must be run on encountering the word in the input, causing an interruption of the ordinary flow of instructions. The "and" program makes the PROGRAMMAR system suspend its current parsing task, and start trying to parse another constituent of the same type as the one it is currently parsing.

The SHRLDU program converses about a small world of toy blocks sitting on a table ("the BLOCKS world"). It has an internal representation of these items in the form of a relational structure. That is, there is a data-base of items where each item represents a relationship between entities. Each entity is represented by a LISP atom. Thus the data-base item

(SUPPORT B1 B2)

records the fact that there is a relation of "SUPPORT" between the two "blocks" B1 and B2. The whole state of the BLOCKS world can be recorded in statements like this. The "meaning" of a sentence for SHRDLU is therefore expressed in terms of items and operations in the

BLOCKS world data-base. For example, the meaning of "a red cube" is a small program to find an equidimensional block which is marked as red. The meaning of "pick up a red cube" is a program to find such a block and pick it up.

The conversion from input words to BLOCKS world actions is in several stages (not necessarily consecutive, but logically separate). Firstly, the syntactic tree is built. Secondly, each major constituent node (NG, CLAUSE, etc.) is converted to semantic representation, by the "semantic specialists". Thirdly, the semantic representation is used to examine or alter the BLOCKS data-base. In fact the syntax to semantics conversion may be carried out at stages during the syntactic tree-building, since, when a major category node is completed, the semantic specialists may be used to transform the node to a BLOCKS world item. (Unfortunately, Winograd's exposition is not too clear about certain aspects of the interleaving - see Ritchie (1976)).

The SHRLDU program is related in various ways to the MCHINE program, but there are major differences, as in the case of the LSNLIS system, in the methodological approach. Winograd was not constructing a practical program for non-linguists to use for information retrieval, but the goal of implementing a working program with impressive performance was fairly central. This may have led to some blurring of the distinction between implementation details and proposed principles, and to the use of slightly ad hoc measures. It is possible to look on computational grammar (and the MCHINE project in particular) as an attempt to clean up some of the ideas in SHRDLU, LSNLIS and related systems in order to perform linguistic

description.

Section II.10 : Wait-and-See Strategies (Marcus)

It is worth examining in detail the suggestions made by Marcus (1974, 1975), since he adopts goals and techniques which are closely related to those of the MCHINE project. The emphasis in Marcus' work is on deterministic processing, and he wishes to design a parser which contains enough grammatical knowledge to avoid making mistakes or having to follow up several analyses simultaneously. This is a worthwhile aim, and work done with this goal should help to elucidate the sentence-understanding process. Marcus observes that many language-understanding programs in artificial intelligence rely on exhaustive, mistake-driven strategies, instead of incorporating more specialised decision-making devices (see Section III.8 for further comments on this). He then proposes a system which, he claims, will perform more efficiently without such mistake-based control.

Marcus' grammar is organised as follows. The smallest units are modules, which are structure-building procedures, invoked by the presence of certain features in the input (each module has its own feature pattern), and modules are grouped into packets. A module can be invoked only if it is active, and a module is active if and only if some packet which contains it has been made active. Certain packets are active at the start of the analysis, and others become active as a result of modules explicitly activating them. Modules have a priority number and those with higher priority numbers will be invoked in preference to those with lower priorities. Processing occurs at two levels - group and clause - with each level having an input buffer of items waiting to be tested by the modules, and an

output buffer of structures produced. At the group level the input buffer contains the words of the sentence, and the output buffer contains the groups (small phrases) formed by clustering the words. At the clause level, the input buffer is the output buffer of the group level - i.e. it contains groups - and the output buffer contains clauses. The feature-pattern on a module is compared with the first few items in the input buffer, and if the features match, the module is invoked. If the feature-patterns of several active modules match the input, the analyser may use a "differential diagnostic" to narrow down the range of possible hypotheses. Differential diagnostics are a "series of easily computed tests that decides between the competing hypotheses" (1975, p.7). Clause structures are built by "a case-frame interpreter that is intended to serve as an interface between it [the parser] and deep world modelling" (1975, p.8). Also, "the parser can ask pre-compiled fill-in-the-blank questions of the world model itself (the world model in question being the author" (1975, p.8).

This overall outline is similar enough to other current frameworks and programs not to be too controversial. The interesting aspects of Marcus' suggestions are the constraints within which he wants the mechanism to operate, and the performance which he claims is possible within these bounds. Marcus makes various speculative claims regarding processing time, use of world knowledge, and ways of processing semi-grammatical or elliptical sentences, but these are more peripheral aspects. The main strong claims regard syntactic processing.

Marcus claims that, at any point in the analysis, the feature-patterns of active modules are to be matched against only the first few items in the input buffer - no extended look-ahead is required. This initial testing should narrow the possibilities down to what he terms two or three "hypotheses". If several hypotheses are possible, the parser executes a differential diagnostic to determine which single "hypothesis is correct, before attempting any of them " (1975, p.6). It is crucial to Marcus' claims that this test uses very limited lookahead, is simple, and yields a definite result. It is unclear from the two papers cited whether recursive levels are allowed, or whether just one clause level and one group level can exist simultaneously. If recursive levels are involved (e.g. a clause level outputting items into the input buffer for a group level) then look-ahead of three constituents is not very restrictive, since one constituent may contain an arbitrarily large nested structure; if recursive levels are not incorporated, presumably short look-ahead tests will fail in any example where some embedded structure occurs in the group that is being built (e.g. a relative clause within a noun phrase).

The simplicity of the tests is also open to question. The example he gives (1974, pp.15-16) includes a test to see if the next item will be an adverb or a time adjunct; this does not seem to be a simple test, and would probably require the item in question to be parsed. Marcus' grammar is fairly narrow in scope, and it is fairly easy to construct simple rules if only a few examples are to be covered; it is another step to claim that grammars of this form will never need complex rules. It is not clear whether "hypothesis" corresponds to "successfully matched module" on a one-to-one basis;

if they do not, it is not clear how "hypotheses" are to be counted. It may be that the counting of hypotheses, and resorting to a differential diagnostic if the total exceeds one, occurs in the grammar-writers mind, rather than in the execution of the program, since Marcus talks of a differential diagnostic being a particular kind of module, constructed so that it is invoked only when the modules it has to choose between will also have been invoked. Maybe this describes the way that the modules are constructed, rather than an algorithm which they actually obey.

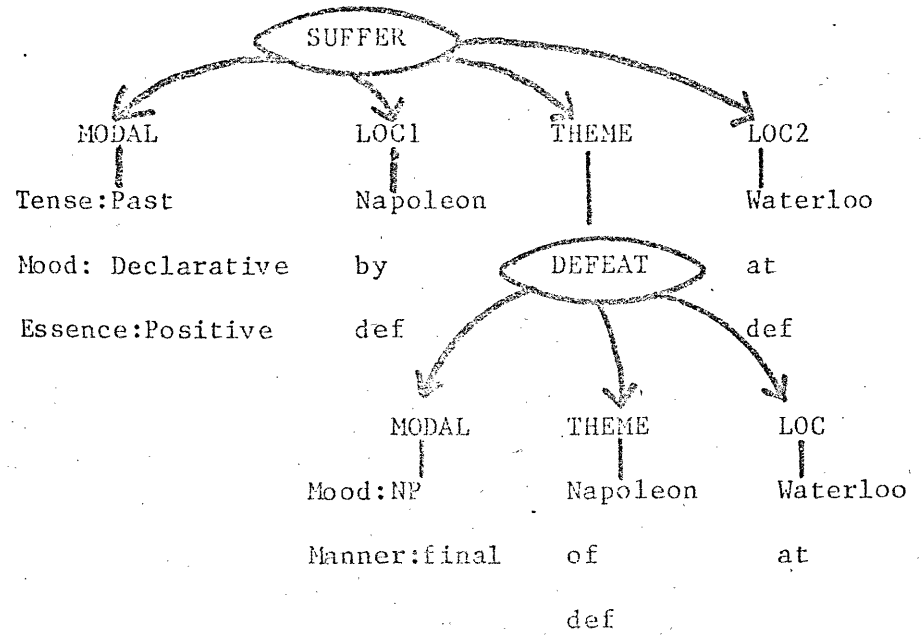
There are several aspects of Marcus' work which are similar to the work in this thesis. For example, both projects accept the principles that structure, once built, must not be re-built or dismantled during an analysis, and that look-ahead should be avoided if possible. However, the more central claims of the "wait-and-see" approach remain unproven so far.

(For further comments, see Sections III.6 and III.8, and Section VII.2.4).

Section II.11 : Semantic Networks (Simmons)

Simmons (1973, 1975) has outlined an English language understanding system which has many similarities to the framework used here. Simmons uses a form of augmented transition network grammar to process input sentences directly into a "semantic network" representation.

A "semantic network" is a form of representation closely related to the standard semantics for predicate logic (cf. Shoenfield (1967)). There are a set of "relations" which can occur between items, and the items can be virtually anything. The word "network" is used because such relational structures are often represented as graphs with labelled edges, and "semantic" merely refers to the use of this representation for language meanings. For example, Simmons gives the representation of "Napoleon suffered defeat at Waterloo" as:



(For other semantic network systems, see Quillian (1969) and Rumelhart and Norman (1973)).

This method has been used to build a program which accepts declarative statements (e.g. "A clown holding a pole balances on his head in a boat") and displays corresponding pictures on a graphics screen.

The grammar used does not alter the surface order or relation of the input string greatly, but each verb has an associated set of "paradigmatic ordering rules", which express permutations of the surface subject, object and indirect object into a deep case frame for the verb. This is very similar to the computational grammar way of handling verbs, as will be seen in Section V.8. However, this action is performed in Simmons program by a particular operation ARGEVAL, rather than by having anything corresponding to structural combining rules (see Sections III.1 and IV.1). The semantic network is built directly by the actions on the ATN arcs. As argued in Ritchie (1977), this approach is difficult, since the suitable constructs for a surface ATN are different from those for a semantic representation, if there are no intermediate rules. Simmons can use one-stage processing largely because his semantic network is very close to a traditional surface syntactic structure, apart from the permuting of the verb arguments, and any deeper semantic patterns have to be covered by inference rules. It is not very clear how the verb-permuting operation (ARGEVAL) chooses which "paradigmatic ordering rule" to use in arranging the verb arguments, since Simmons says that this is done using the semantic characteristics of the items. Since much of the case information in English is conveyed by surface syntactic aspects (order,prepostions, inflections, etc.) alloting items to verb places using solely semantic information will not in general be possible.

The main difference between computational grammar and Simmons' model is that the latter has no constructs corresponding to "structural combining rules". Hence Simmons' grammars have to try to capture all syntactic and semantic regularities at a single level of semantic representation, and this semantic structure is built directly on the ATN arcs.

CHAPTER III

IMPROVING THE EXISTING CONSTRUCTS

Section III.0 : Preamble

As a preliminary to presenting the computational grammar framework in Chapter IV, this chapter discusses in more detail some of the questions which arise when constructing a computational model of English language understanding. The arguments are not directly based on any one of the theories or systems in Chapter II, although many of the points are expressed as criticisms of the LSNLIS or SHRDLU systems, since these two models are closely related to the MCHINE program.

Although many of the ideas used in computational grammar are not totally new concepts, most of them have been modified, to some extent, to overcome various problems. Chapter III describes these basic concepts and the reasons for the revisions, so that Chapter IV can simply summarise the framework without digressing to discuss the background justification. In some cases, the modifications are minimal, and the corresponding part of Chapter III merely summarises the reasons for adopting a particular device.

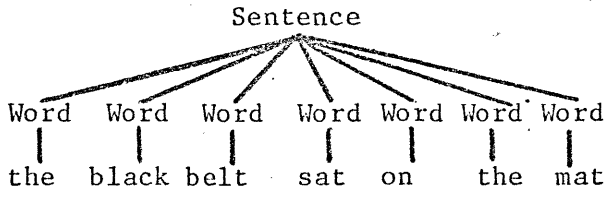
Section III.1 : Structural Combining Rules

Syntagmatic relationships between units are decoded in two ways in Winograd's program. Firstly, the parser builds the syntactic tree. This gives a grouping into constituents, together with a labelling (using features) of each constituent. Secondly, these labelled subtrees act as inputs to the semantic specialist programs, which produce semantic forms. The semantic specialists are treated more or less as "black boxes" constrained only to produce a particular output from a given input. The lack of constraints on these two stages (other than the need to interface with each other) has two slightly unfortunate consequences. The inscrutability of the semantic specialists means that they contribute little to the important question of how syntax is related to semantics. The second consequence is directly linked to this lack of a systematic method of semantic interpretation. The syntactic component lacks criteria for what is "good" or "bad" grouping or labelling of a surface structure, other than the fact that it should "work" when handed to the appropriate semantic specialist. Ad hoc feature markings (of the sort discussed in Section III.11) are successful solely because they serve as the input to the equally unconstrained semantic interpreter.

If we wish to illuminate the process of extracting meaning from surface strings, we must dissect the semantic specialists, and try to organise the inner mechanism as systematically as possible. Consider the noun group semantic specialists. Winograd used what he called the "slot-and-filler" approach to the structure of the noun group - there are various slots for modifers, etc., on the group node, which

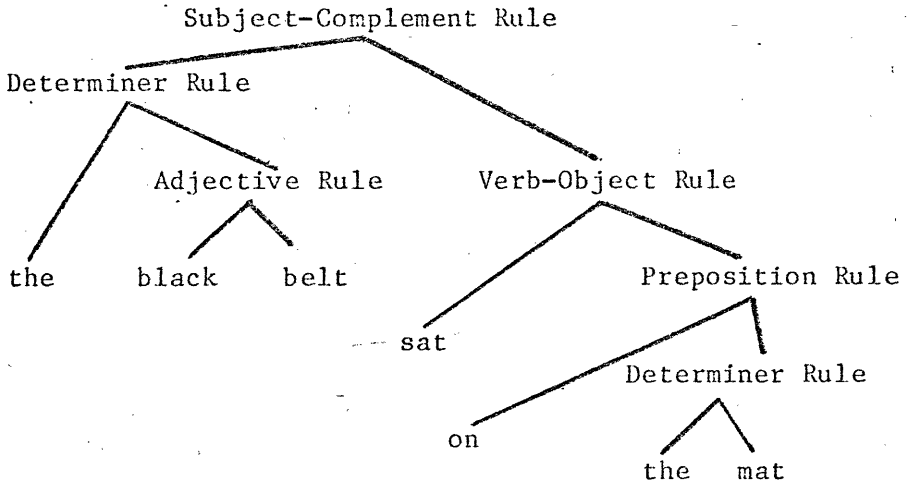
may or may not have entries. This meant that "null" entries had to be made on many noun groups (giving rise to features like NDET - see Section III.11). The semantic specialist had to examine all the slots and make its decisions on the basis of the various entries. This obscured any internal semantic regularities between subunits. For example, the relationship between determiner and head noun is not separable from the relationship between adjective and head noun - both are buried somewhere in the deliberations of the semantic specialist. Winograd suggests that his "flatter" trees are more semantically useful than more deeply-branching trees of transformationalists (pp.16-17), but this overlooks two points. Firstly, he uses elaborate feature markings, some of which convey further structural information (which is inherently present in the transformationalists' tree already). Secondly, the deep branching was an attempt to reflect semantic groupings; with Winograd's method, one has to deny these groupings, or hide them in the viscera of a specialist, or encode them in further "features". If we were to take Winograd's method to an absurd extreme, we might end up with trees like (10), a single semantic specialist (for "Sentence"), and a vast collection of features.

(10)



The whole point of further grouping, as Winograd himself states, is to find generalities that can be dealt with at a local level, and make use of all semantic regularities. If we can find syntagmatic regularities within the noun group, then there is a case for separating these out into individual semantic specialists. During the analysis of a sentence, the program would need to invoke only those semantic specialists for which non-null arguments existed. The tree might look like (11), where each node is marked with the semantic specialist used.

(11)



It may appear that this produces a proliferation of specialists, but this is misleading. The rules which we are referring to as semantic specialists here are much smaller than those in Winograd's program, and have been obtained by dismantling his rules. One could keep a small number of large specialists by building more and more alternatives into the body of each rule, but that would not really be simpler.

In Winograd's system, the semantic specialist was selected according to the major category label on the node (e.g. NG). If we do not have specialists indexed to major categories, the question arises of how to select the rule. This question is not trivial, and will be considered next (Section III.2).

Section III.2 : Syntax and semantics

Winograd's parser acted as a pre-processor for the semantic specialists, by grouping and labelling constituents. As commented in I.2.2, the only justification for syntactic rules is that they contribute to the language-decoding (or encoding) process. The question is whether this two-stage process has any advantage over a one- stage conversion. There is a sense in which Winograd's parser makes all the semantic decisions, since the grouping and labelling defines how the semantic specialists will act; on the other hand, these decisions are re-encoded (in arbitrary features) instead of being used directly to build a semantic structure. The feature-laden tree is then dissected by the semantic specialists so that these earlier decisions can have semantic effect. Winograd states (pp.16-17) that the sole justification for his features is that they convey meaning, so presumably all the features (even bizarre ones like DPRT) perform this function of passing decisions between the syntactic and semantic components.

One obvious modification to investigate is the abolition of the arbitrarily-labelled syntactic tree as a mail-box between the components. If we could integrate the semantic specialists and the parser more closely, we could perhaps allow the parser's decisions to be more directly related to the meaning-manipulatios that they cause.

Consider the original form of the semantic specialists (ignoring, for the moment, the breaking down of the specialists into several rules, as outlined in Section III.1). The noun group specialist constructed a semantic structure on the basis of the items

in the seven "slots" (Winograd, p.56) on the NG subtree, together with the feature-labelling of the subtree. In a sense the seven slot-fillers were acting as arguments for the specialist, since they provided the variable information that distinguished one noun group from another. Suppose, therefore, that we re-formulate the semantic specialist to be a procedure which takes seven arguments (some of which may be dummies). Instead of establishing a syntactic NG rule, and attaching labelled daughters, the parser could establish a semantic NG node (containing a copy of this new specialist) and fill in the appropriate arguments. Not only would this avoid some tree-building, it might simplify the internal code of the specialist (since it would not have to "read" the syntactic tree). Also, features which are intended solely to provide structural information (such as NDET) would be eliminated. This does not mean that no features are needed (number-markings would probably still be required) but it would eliminate any redundant syntax, and show more clearly what information was being used where.

If we also adopt the suggestion of Section III.1, and have separate specialists for each syntagmatic combination, the need for dummy arguments should be eliminated. The parser builds a tree, each node having an associated semantic specialist, all of whose arguments have to be filled. (There may be only 2 arguments for some rules, if deeply-branching trees are used). The tree would look something like (11) above.

This proposal is not particularly radical. All that we have done is suggest that, since the parser's decisions determine (ultimately) which semantic specialists are to be invoked, and what

arguments are to be passed to them, these facts should be directly represented. The operation of the semantic specialists was already hierarchically-organised (as observed in Section III.1) so using them as the cement which holds the tree together is not an extension of their use; it merely makes explicit the fact that the hierarchical syntactic organisation was ultimately aimed at channelling the semantic processing. The semantic specialists themselves (both in their original form and in the revised version) have fundamentally the same purpose as the semantic interpretive rules of Katz and Fodor (1963) and Jackendoff (1973), or the rules of Montague (1970a). The revised notation for the semantic/syntactic tree is similar to the analysis tree of Montague (1970b). These rules could be referred to as "structural combining rules" (SCRs), to avoid having to categorise them as "syntactic" or "semantic", as they fulfil a dual role.

The rules as described so far do not fulfil all the functions traditionally covered by "syntax". There is other information to be used in the sentence-understanding process and we need further constructs to handle it.

Firstly, there are certain properties of words and phrases which help to indicate how the various items are to be grouped in building the surface tree, and which seem to have no other purpose. That is, they do not contribute directly to the final semantic structure, but they help the routines which plug the combining rules together. One good example of this is "number agreement" which (as Katz (1972) argues convincingly) is not just a semantic property. "Agreement" between subject and verb is a rather arbitrary, language-specific device which helps to signal to the analyser how (or whether) the

items are to be paired, and can be represented by some suitably neat symbolic markings (see Section V.3). The "object-information" described in Sections III.9 and V.8 is another example. Such information can be recorded by associating a set of "syntactic properties" with each word or phrase; that is, a list of attributes which can have a range of values.

Secondly, there is a need for some way of describing paradigmatic classes. In writing a recognition grammar, there is often a need to include a test for some class of words whose only unifying characteristic seems to be that they can occur at certain points of a sentence. For example, English auxiliary verbs generally occur at the front of a "yes-no" question, rather than after the subject. A recognition grammar needs to be able to check the first word of the sentence to see if it is an "auxiliary", as this suggests that the sentence may be a question. It is hard to define semantically what an "auxiliary" is, and it seems to be a purely syntactic category - "one of those little verbs that precede the main verb and are at the front in questions" (see Section V.2). Other examples are the various constituents within a noun phrase - determiners, possessives, articles, adjectives, etc. These constitute options which occur at various stages throughout the processing of the phrase, and the ordering of them (and the occasional parallel options) can only be described by idiosyncratic markings on the words, since their semantic characteristics provide no clue to what order the options should occur in. This distributional information could also be covered by "syntactic properties" as above, but there are further generalisations about the information. Firstly, a syntactic property may have a range of

values, whereas the paradigmatic information usually gives simple class membership (e.g. "is this word an auxiliary or not?"). Secondly, syntactic properties (e.g. subject-verb agreement) may need to be marked on phrases as well as words, whereas the paradigmatic classes are to guide the analyser when it is inspecting words. In computational grammar, two separate constructs have been incorporated to cover these two kinds of information :

Syntactic properties, which are attributes capable of taking a range of values, and which can be marked on phrases.

Syntactic features, which can simply be present or absent (presence denoting membership of a particular paradigmatic class) and which are marked only on lexical items, not on larger constituents. Notice that syntactic features are not like syntactic categories, in that items may be cross-classified by giving each item several syntactic features, instead of just one. (See Chomksy (1965) and Halle (1962) for arguments in favour of this form of classification).

One further construct is needed to integrate these concepts. If syntactic properties are to be marked on phrases, how are the markings to appear on the phrases as they are built ? There are a number of logical possibilities, ranging from having a phrase inherit all the syntactic properties of all the words in it, to allowing the properties of the phrase to be computed in any way whatsoever from the properties of its words. Writing the MCHINE grammar threw very little light on this, as only one syntactic property was included which had to appear on phrases; namely, the "verb-agreement" class of a noun phrase. The method used in the program was to allow each structural combining rule to have, optionally, a "property

inheritance rule" which specified which properties of the argument nodes were to be attached to the result-node. (Since these properties are not deemed to be part of the meaning, but merely transient annotations to aid the surface-structure routines, they are marked on the nodes, not on the semantic items contained therein). It is impossible to claim empirical support for this method, since only the one example has been tried out.

Section III.3 : Immediate Semantic Processing

The LSNLIS system postpones all semantic processing until the end of the sentence, and SHRDLU performs semantic processing at the end of major constituents. Rieger (1974) criticises Winograd's approach for having too great an emphasis on syntax, and Riesbeck (1974) claims that his analyser can build up a non-syntactic "conceptual structure" as it progresses from left-to-right through a sentence. Let us examine how the SHRDLU mechanism might be modified so that semantic processing may be carried out as early in the analysis process as possible.

If we adopt the modifications presented in Section III.1 and Section III.2, then a first step has been taken. The parser (or "analyser") is building a tree composed of semantic rules with their arguments inserted. To produce the semantic structures, these semantic rules have to be applied. The question is - how early in the analysis can this rule-application be done ? The dismantling of large semantic rules into separate autonomous rules, each having perhaps only two or three arguments, should facilitate early semantic processing, since the semantic rule does not have to wait for the accumulation of several constituents as its arguments. It might be possible to apply each semantic rule as soon as all its arguments have been found. Then semantic processing would occur every time a subtree (no matter how small) was completed.

A further extension is possible. Instead of each rule waiting for all its arguments to be inserted before operating, a rule could operate incrementally. That is, rules could be subdivided into parts (call them "rule-components"), one for each argument-place in the rule. The nth rule-component would represent whatever semantic computation could be performed when the nth argument was found. Hence each rule-component could be applied when the corresponding argument was inserted, causing a gradual building up of a semantic structure. (It might also be useful to include a "zeroth" rule-component in each rule, to set up an initial blank structure for the other rule-components to work on).

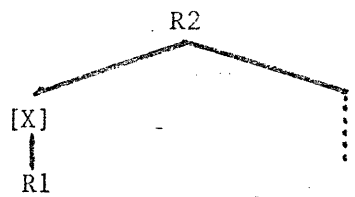
One experimental version of the MCHINE program implemented such a system, and the grammatical rules were re-written accordingly. The system was soon discontinued, since it was very cumbersome in practice, but the short trial was enough to illustrate the limitations of such a scheme. Using rule-components is not impossible, but it is more difficult for the grammar-writer to design rules in the incremental form, and there seem to be few advantages. Two main problems arise.

Firstly, many of the combining rules are such that the first argument conveys little information about the final result. This is particularly true where the arguments are themselves being represented as an operator applied to an operand - both are needed to produce the final result. This problem also arises where the second (or later) argument supplies the main structure for the result, and the first argument fits into some slot or property-value on it. In a number of rules the rule had to be written so that the zeroth rule

component set up a structure of type X, the first component of the rule merely stored the first argument in some arbitrary slot on the X (as a temporary store) and the second (final) rule-component took the first component out of its intermediate storage position so that all the computation could be done. These bizarre manipulations demonstrated how rarely any useful processing can be done without all the arguments.

Secondly, even if such partial building were possible, the semantic processing could not go very far unless the next rule up the tree could accept a partially-formed argument. That is, even if rule R1 (in (12), below) gradually built a semantic item X, the semantic consequences would stop there unless Rule R2 could accept the partially formed X as its argument.

(12)



Possibly the only places where it was possible to segment the rules without losing anything were in the role-placement rules for verbs (see Section V.8). If there is a rule which arranges the object (and indirect object) into slots around the verb, then each rule-component can slot its argument in separately, and any restrictions on the slots (induced by the verb) can be checked immediately. The slight advantage of early checking of verb-restrictions has to be offset against the other difficulties created by having to write incremental rules.

Section III.4 : Sense and Reference

Winograd's program keeps a single data-base which describes the state of the BLOCKS world. No distinction is made between how the blocks are actually arranged and how the program "thinks" they are arranged - since one data-base serves for both, no discrepancies are possible. Hence there can be no real distinction made between "Is there a red block on the table" and "Do you think that there is a red block on the table?". There are two ways of regarding this single world model :

(13)

(a) It represents the "real" state of the BLOCKS world, as a simulation of a physical system.

(b) It represents the "mental" model of the BLOCKS system that the program (regarded as a simulated speaker) has.

The consequence of interpretation (13)(a) is that the program performs all its operations, including linguistic ones, by examining the "real" world. That is, to work out exactly what the phrase "the red block" refers to, the program must scan the world (not its memory of the world) to find such a block. Hence no provision is made in this model for talking about items which the hearer cannot currently perceive, which is rather a limited approach to language.

The consequence of interpretation (13)(b) is that any operations which the program is to perform on actual objects (e.g. "pick up") must be carried out on mental constructs. In this approach, actions

are performed by fiat, analogous to a person playing chess without a board. This makes no allowance for the case where a "real-world" interface is needed (e.g. playing chess with a board).

There seems to be a confusion in the Winograd model between two logically distinct processes, which might be called "reference evaluation" and "referent recognition". When someone attempts to understand a phrase such as "the President of the United States", there is no need for him physically to search the White House to "find a referent" for the phrase. If the hearer has sufficient world-knowledge, he will be able to work out that a particular person (probably not currently perceptible by sight or touch) is being described. To do this, some manipulation of the hearer's memory may be involved - but he can do this sitting down with his eyes shut. Let us call this mapping of a description to a particular object "reference evaluation". (It will be discussed in greater detail in Section V.6). On the other hand, if someone has to obey the command "Shoot the President of the United States", some further processes are necessary. The person will have to get himself into a physical position where he can perceive the President, and must be able to recognise the President (probably visually). The latter stage (perception and recognition) could be carried out by a deaf-mute to whom the assassination command had been conveyed by pictures, and is not essentially linguistic. Let us call this "referent recognition".

A brief digression here will show that this confusion is not confined to Winograd's program, and will suggest some possible consequences of this distinction. Some writers imply that a phrase will "have a referent" if and only if it describes some real world

object(s). For example :-

"There are also clear advantages to a language that can employ expressions which have no referent or which have not been secured a referent.... We surely want our language to leave open for its users the possibility of constructing theories which hypothesize the existence of such things as phlogiston, ether and animal spirits..." (Katz (1972, p.142))

"The reference of a proper name is the object itself" (Frege(1892, p.60))

Brown (1958) uses "referent" in a language-learning context to describe concrete objects. Russell (1905) uses the word "denote" to mean "standing for real world objects", and says that the phrase "The present King of France" does not denote, since there is no such existing object. However, for well-formed, successful conversation, it is the reference evaluation step that is important, not the referent recognition. As Russell(ibid.) comments:

"It often happens that we know that a certain phrase denotes unambiguously, although we have no acquaintance with what it denotes".(p.479).

Certain noun phrases in English are a clear example of a linguistic device for automatically providing referents for the hearer, although he may have no perceptual data from the associated real-world object (see Section V.6 for further discussion) :

(14) My oldest brother lives in Glasgow

(15) His father gave him a cheque.

Here the term "referent" will always be taken to refer to the construct that is determined by reference-evaluation in some model of the world. The corresponding object in one particular model (known as the "real world") will be referred to as the "concrete referent".

One advantage of making this distinction is that we can distinguish between the concrete referent of a phrase and different people's referents for that phrase. In the classic example, the phrases "the morning star" and "the evening star" have the same concrete referent; but someone who does not know any astronomy may regard the phrases as referring to different objects . Such a person can be said to have two different referents for the phrase (in his notion of the world).

When describing the semantics of phrases which refer to something (e.g. noun phrases) it is important to make a distinction between the "meaning" of the phrase (usually described in some formal representation) and the set of things the phrase refers to (as observed above, these things are not usually in some absolute "real" world). This is (roughly) the traditional distinction between "sense" and "reference" (Frege(1892)).

Montague (1968, 1970, 1972) formalises this distinction in a rigorous way. His model uses the notion of a "point of reference", which is used as a parameter in deciding what an expression refers to. These points of reference (corresponding to "contexts" or "states of the world") are sometimes represented as ordered pairs (a

possible world paired with a situation of use), but they could be decomposed further, depending on how many factors we wished to separate in a state of the world. The point of reference has a direct influence on how an expression refers, because Montague defines the meaning of a term to be a function from points of reference to functions which define sets of objects. A single meaning (i.e. the same function) may refer to different objects, if it is evaluated at different points of reference.

There is an obvious analogy here with program-evaluation. A piece of program must be evaluated in some context (i.e. some state of the machine). The influence of the context is particularly relevant for languages which determine all the variable values at run-time (cf. Moses (1970)), and some recent developments in artificial intelligence languages have allowed program-contexts to be manipulated as items, so that a piece of program may be executed in different contexts on different occasions (McDermott and Sussman (1972), Stansfield (1975), Davies (1973)). Even a simple piece of program such as (A+B) will refer to different numbers depending on the values of A and B.

Expressions like "I" and "he", whose reference is entirely dependent on the context of use, are sometimes classed as "indexical expressions". Bar-Hillel (1954) gives an elegant treatment of such expressions, in which he points out that, to discuss reference, we must always include the context of use in the calculations. Indexical expressions are merely those where the influence of context is very obvious - in principle, the context must always be taken into account. Sentences which are non-indexical (such as "ice floats on

water") are those whose reference is unaltered by changing the context. (Compare this with the program "sin(0);" , which should yield 0 no matter what program context it is evaluated in).

If we let the "meaning" of a noun phrase be a piece of program, which, when run, will produce the set of objects referred to by the phrase, then the sense/ reference distinction is automatically incorporated. Winograd uses pieces of program to represent the meaning of noun phrases, although it is not clear that they are used as systematically as Montague's intension functions (see Ritchie (1976)). There are certain difficulties involved in using pure programs for noun phrase meanings (see Section V.6), but it is worth noting the analogy between program-evaluation in a program-context and determining the set of referents of a meaning. It seems desirable to retain this aspect of the Winograd representation in any modification.

Section III.5 : Registers

As observed above, computational grammars often store partially-built structures in registers. In fact, most of the operations performed during sentence-analysis use the contents of registers in some way, and if we are to investigate how sentence-processing operates, it is desirable to investigate how registers are, and can be, used. In writing the MCHINE grammar, an attempt was made to keep track of the various uses of registers, as these operations form a major part of the work done by the analyser. The following classifications proved useful for describing this area.

The first useful distinction is between flags, structure-holding registers, and pointer-holding registers. A flag is generally used as a location to test for some simple condition, and holds one of a small, fixed set of values (e.g. "TRUE", "FALSE"). The idea is that when some condition occurs during an analysis, this fact is recorded for later use by setting the value of a flag; at some later stage(s), when the analyser needs to test this condition, it need not do any re-computing, but merely examines the current value of the flag. Some recognition grammars use a two-valued flag to record whether a passive or active verb-configuration has been formed; the analyser, on encountering a "by"-phrase, can react differently acccording to the value of the flag.

A structure-holding register provides temporary storage for a piece of structure which has been built but not allocated to a position in some larger structure. For example, a register HEADNOUN might be useful for holding the head of a noun group while checking

if any adjuncts follow the group. A pointer-holding register is used for keeping track of where, within some larger structure, a particular item is. For example, a register CURRENT-NODE might be useful for keeping track of the node currently being processed. If this distinction seems slightly vague, consider the following. Failing to assign an item to a pointer register makes that item difficult (but probably not impossible) to find, and does not alter the relationship of that item to any containing structures. Failing to assign an item to a structure-holding register loses it completely, since it is not yet attached to any other structure. (In languages like LISP and POP-2, such unassigned structures would be garbage-collected).

Another distinction (independent of the 3-way classification above) is between interpreter registers and grammatical registers. In the course of analysing a sentence, it is necessary for the program which scans the input and the network to keep track of where it is in the network, and where it is building structure. For example, the CURRENT-NODE register mentioned above might be needed, and a register CONTINUATION for holding states still to be processed. These interpreter registers are more or less independent of the transition network grammar being interpreted, although the linguistic theory within which the grammar was written will affect the choice of such registers. On the other hand, certain specific grammatical registers will be needed, according to how the grammar is written. For example, the HEADNOUN register instanced above might be useful in one grammar, but irrelevant in another.

Generally, registers are used just like variables in a programming language, with each register having a distinct name to identify it. The analyser can examine specific registers by name, and have separate registers for as many items or pointers as it needs. One possible variation on this is to have just one or two general purpose registers, which are used for different purposes at different stages in the parsing. Call these work registers. Such registers are trickier to use, since the parser has to keep track of what has been most recently placed in a work register, and make sure that the stored values for one process do not interfere with those for another. The only advantage to be gained is the general methodological one of restricting the mechanism as much as possible. It would be interesting to construct a parser which requires only a very limited amount of storage space; using work registers instead of named registers is one way of investigating this. (It would also avoid the covert decision-making described in Section III.8, (28)(f)).

As described in Section III.7, constituents are processed by nested subunits. In the course of such processing, working space for the different constituents must be kept separate, so that stored values do not corrupt each other. Suppose there is a grammatical structure-holding register HEADNOUN which is used to process a noun group. If several noun groups are nested within each other, each different noun group may need a separate copy of HEADNOUN. Since the processing is hierarchically arranged, one way to achieve the integrity of working space is to allow certain registers to act like local variables in block-structured programming languages (e.g. ALGOL, POP-2). That is, a fresh incarnation of a local variable is

available to each hierarchical unit that requires that variable; these incarnations exist only temporarily, as long as the hierarchical subunit is processing, and then they vanish. One way to look on such a system is to regard the registers in question as push-down stores; the current value is kept on top of the store, and the store can be pushed down on commencing a nested unit, providing a fresh value slot while storing the previous value temporarily. If we call such registers stack registers (or simply stacks), this gives another possible classification, independent of the previously described distinctions.

Thus we can class a register as being:

pointer-holding, structure-holding, or a flag

grammatical or interpreter

work register or named register

stack or non-stack.

Notice that this section has not suggested any concepts that are not already in use in existing programs; it has merely drawn attention to certain possible classifications that are possible, so that later sections can be described in a more precise terminology. This is necessary if we are to examine fully the linguistic formalisms being proposed. Hitherto, this level of detail has not been discussed, but it is essential to the development of more complex (or more restricted) devices. Since most (if not all) of the operations carried out during sentence analysis use registers (and the transition network configuration merely expresses the list of

options) elucidating the uses of registers is the major part of elucidating the processing operations.

Section III.6 : Control structure

Owing to the ambiguity of natural language, an analyser is frequently faced, during sentence analysis, with having to explore several possibilities. There are two standard strategies that a parser can use to follow up multiple options. A program is said to use a depth-first strategy if, at each choice point, it chooses one of the options and follows it up fully, to the exclusion of others. If a choice proves to be wrong, the program "backtracks" - i.e. it returns to the most recent choice point, undoing all or most of the processing performed since that point, and tries another option. When all the options at a choice point have been tried unsuccessfully, the program backtracks to the previous choice point. This has the advantage that there is only one partial analysis being maintained at once (with choice points recorded in some way), and (if the program makes the right choices) the correct analysis is found very quickly. One disadvantage is that if the program makes the wrong choices, it has to do a great deal of backtracking, and this may involve undoing processing which will have to be repeated when other options are taken. A drawback to having only one partial analysis in existence is that there is no way of comparing several possibilities, so that the "best", in some sense, may be chosen. Several analyses can be compared only by producing complete analyses. Instead of the analyser stopping when it has found one successful path, it continues to explore systematically all the other choices that it has not yet tried (in the same way that it would if it were backtracking).

A program is said to use a breadth-first strategy if, at each choice point, all the options are developed simultaneously. This has the advantage of finding all the analyses in one pass, if there are a few, and it allows (in principle) the program to compare several different partial analyses, selecting the better ones for consideration. In a pure breadth-first system, such a selection facility would not be used, since all analyses are developed, regardless of merit, but a natural development would be some form of "pruning" of the set of analyses (if some principled method could be found). The main disadvantage is that several analyses must be maintained simultaneously, even although many of them will prove to be transient.

The augmented transition network formalism is neutral between depth- and breadth-first exploration. A particular network provides the arrangement of the choice points, but these may be explored in either fashion. The program of Thorne, Bratley and Dewar (1968) used a pure breadth-first approach, and the LSNLIS system (according to Marcus (1975)) uses the depth-first approach.

If we wish to examine the details of how decisions are made in sentence-analysis (with an eventual aim of constructing a plausible model of human processing), then pure depth- or breadth-first systems are not very helpful. The reason for this is that both methods have inherent inefficiencies which are overcome by brute force. These are exhaustive strategies, in which the grammar does not have to be particularly clever, elegant, or carefully constructed in order to succeed . If parsers are justified solely on the grounds that they "work" with sufficient computation, then we are left with no means of

comparing rival recognition grammars. With a suitably large computer, one could write a parser which would produce any desired analyses for sentences, just by putting suitable (ad hoc) markings in the dictionary, and by making the parsing strategy exhaustive. The whole point is that we want to be able to say something about how the parser operates, and whether one program is neater than another - the fact that it eventually produces the analyses is not a sufficient condition for adequacy. We need to refine our notions of parsing strategies, simply because, as commented in Chapter I.1, our ultimate criterion in a scientific investigation is that we should try to find the neatest solution.

This is not to say that all parsers that use exhaustive strategies are alike. The parser for the IBM REQUEST system (Plath (1973), Petrick (1973)) used a reverse transformational derivation, which resulted in a large combinatorial explosion. Quite short sentences had so many possible surface groupings that even with several 100K of machine space the parsing was difficult (or impossible). On the other hand, the program of Thorne, Bratley and Dewar (1968), using a breadth-first exploration of an ATN grammar, was able to parse comparable sentences quite quickly, using only 16K. It is implausible to suggest that this difference could be due solely to implementation details. The radically different analysis procedures must account for some of this discrepancy - but that is a direct admission that we can appraise a program in terms of how it works. Once this criterion is allowed, the obvious step is to try to refine the brute force exhaustive strategies into neater, more elegant devices.

A more specific objection to exhaustive processing strategies is raised by Marcus (1974). He points out that there exist "garden-path" sentences (like (16) and (17)), where people (usually) make a mistake in perceiving the structure of the sentence on first assessment, and hence need more than one attempt to "understand" them.

(16) I told the boy the dog bit Sue would help him.

(17) In the book the girl took the basket had magical powers.

In such situations people are conscious of making a mistake and having to re-assess the sentence. Marcus argues that if the normal processing method is depth-first (where automatic backtrcking is part of the process), then people must be making continual "mistakes" while processing ordinary sentences, without any disruptive effect. If that is the case, why should the "garden-path" sentences not be handled smoothly by this automatic, unconscious, backup method ? This argument can also be applied to exhaustive breadth-first systems. If people normally consider all partial analyses simultaneously (and throw away those that go wrong without being aware of any oddity), why should they see only one analysis for "garden-path" sentences ? Marcus concludes that people do in fact process just one of the possible analyses (and hence can be "wrong" in a garden-path sentence), but that they choose this analysis very carefully, rather than relying on automatic backtracking to allow an arbitrary blind choice. He suggests that a recognition grammar should proceed deterministically, building structure only if it will not have to alter it later, and holding partial structures in registers until they can be attached in their final position. This matter will be

discussed in greater detail in Section III.8.

Both Woods (1970) and Winograd (1972) suggest the use of a numerical "weight" which can be associated with different analyses in a parse, indicating which of the analyses is the "best". As observed above, partial analyses cannot be compared in a depth-first system, so if Winograd's parser were to make use of the weighting (the SHRDLU parser does not), it would have to compare alternative complete analyses. Woods says that the weight allows one "to suspend unlikely looking paths in favour of more likely ones" (ibid.,p.605), which suggests that partial analyses are being compared. Wilks (1975) suggests a totally different framework which uses a more sophisticated "preference" system (see Section II.6), and presents good semantic arguments for using such a device. (His metric is not used in a gradual left-to-right analysis, so many of the comments here do not apply directly to it).

The question of how to use "weighting" or "preference" is not simple. There are two main issues - relative versus absolute failure, and local versus global assessment. A facile approach to the idea of preference might abolish the notion of "discarding" a partial analysis, in favour of some mechanism of "reducing preference". Although superficially plausible, that would be unworkable for the following reasons. At any given stage in an analysis, there are certain options available, and "failure" occurs when the input word does not meet the conditions for any of these. If we no longer discard failed analyses, we would have to follow up every option, regardless of the input. Leaving aside the problem that the structure-building might be unmanageable if, for example, a

preposition has to act as a surrogate for a verb, we have created a severe combinatorial problem, since the input would not be constraining or guiding the analyser at all. In ATN terms, the ATN interpreter would have to pass through the entire network, keeping different preference counts, with no search paths being terminated by "failure".

Another suggestion might be to assign each option at any stage (each arc, in ATN terminology) a value, depending on how well the input word matches the corresponding condition. The arc with the "best" value would be the only one to be explored. There are two main problems with such an approach. Firstly, it is a form of "depth-first" searching, and hence does not allow the comparison of more than one partial analysis (thus losing one of the aims of a preference system). Secondly, the initial choices in the analysis process will be made on the basis of the "best" options at that early stage, thus discarding any analyses which might prove "better" later.

These two hypothetical set-ups illustrate an important point regarding "weighting" : the weighting system must discriminate sufficiently to ignore some possibilites, otherwise the search space is absurdly large; on the other hand, it must not discard, for local reasons, possibilities which might later prove viable, and it should not always concentrate on just one analysis.

A working system seems to need the notions both of "discarding" and of "reduced preference", with different kinds of "failure" invoking the two reactions. The MCHINE system operates with the following division of mechanisms. Each analysis path has a weight ("TENSION") which can, in principle, be incremented by any part of

the program (TENSION is initialised to zero, and the lower the TENSION, the "better" the analysis). Failure in arc tests and surface structure building causes the analysis to be discarded, but failure in applying any of the semantic rules (or in "reference evaluation" - see Section III.4) merely causes the TENSION to be increased. (This demarcation seems to be similar to that which is implicit in the programs of Wilks (see Section II.6)).

The above discussion has been aimed at establishing two points concerning search strategies in parsers. Firstly, exhaustive strategies, in which arbitrarily many erroneous paths may be followed are both theoretically uninteresting and intuitively implausible. When a recognition grammar is written, the linguist should attempt to make it as deterministic as possible, in the interests of economy. Secondly, where multiple analyses have to be considered, some notion of "preference" would be useful, so that the "best" of several alternative analyses may be chosen. This entails simultaneous development of a few paths, with some part of the program having access to all the partial analyses.

Another way of classifying parsing strategies depends on whether the parser searches for some specific configuration in the input ("top-down") or whether the input is examined first and then arranged into successively larger structures ("bottom-up").

The breadth/depth classification is logically independent of the top-down/bottom-up distinction, but the combination of top-down with depth-first is quite common (what Marcus calls the "Guess-and-Backup Principle"). Purely bottom-up parsers for natural language are rare in the artificial intelligence literature, and many of the devices

currently in use are hard to classify firmly as either top-down or bottom-up.

For example, the notion of a "demon" (as described by Charniak (1972)) has been used as a sentence-analysing device by Marcus (1975) and by Riesbeck (1974) (under the names "module" and "expectation" respectively). A demon consists of a pattern and a body. At any given point in the execution of a demon-based program, certain demons are deemed to be active. If a demon is active, and some item(s) in the input match its pattern, then the body (a piece of program) is executed. In a sense this is a bottom-up device, since the body of the demon will be performed only if the input triggers the demon, via the pattern; however, the demon-based program as a whole is somewhat top-down, in that only the active demons are available for triggering. To some extent the list of active demons defines what the program is "looking for" in the input, but it will react only to those items which are actually present. The options specified by the arcs in an ATN state are similar in this respect to a small list of demons which may be activated together (like a "packet" in Marcus' terminology). The state suggests what few options to look for (top-down), but only those arcs whose tests are satisfied will be explored (bottom-up). (This does not apply exactly to (PUSH..) arcs, which will be discussed later (Section III.7, Section III.8)).

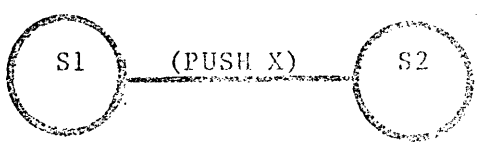
ATNs are often regarded as purely top-down, and demons as bottom-up, but this is slightly inaccurate. It is also not very illuminating to try to put every mechanism wholly into one of these categories, particularly as the most appropriate method for certain tasks may well be some mixture of the two. Various aspects of these

strategies will be discussed further in Sections III.7, III.8 and III.9, but no attempt will be made to say whether a system "should" be top-down or bottom-up.

Section III.7 : Processing Levels

One of the main features of computational grammars is the way in which autonomous subunits can be used to process constituents, so that a constituent made up of several words can be treated in the same way as a one-word constituent of the same grammatical type. The basic actions and operations for performing these nested processes are defined in both the ATN and PROGRAMMAR notations. This subsection attempts to redefine these commands in such a way that the independent decisions in processing constituents may be described separately in the formalism. Some of the changes are simply the introduction of notational conventions, but others represent slightly different ways of controlling the processing subunits. Most of what is said here applies to both the ATN and PROGRAMMAR systems, but the ATN notation will be used as it lends itself to displaying the various facets of the commands. The device provided in the ATN system for activating a subunit is the (PUSH X) arc, where X is a category name (e.g. NP), represented graphically by a section of network like (18).

(18)



Assuming the usual interpretation of the (PUSH) device, the parsing program should carry out various operations on encountering a (PUSH X) arc during an analysis:

(19)

(a) The continuation state (S2 in the above example) is saved on an interpreter stack register.

(b) Some other interpreter stack registers may be pushed down, to keep track of interpreter information at the separate level.

(c) Any grammatical stack registers associated with category X are pushed down.

(d) The start-state associated with category X (i.e. the beginning of the appropriate subnetwork) is used as the next state to be processed.

The operations summarised in (19) define, effectively, the situations where a (PUSH X) arc is appropriate in an ATN grammar :

(20)

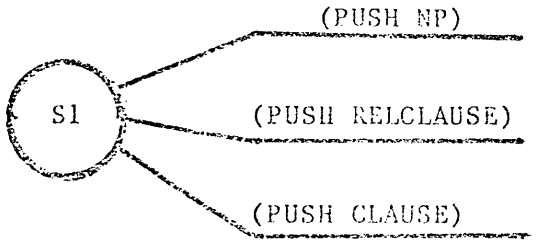
(a) Where fresh working space is needed to process a constituent separately, without destroying information about higher constituents. (This is the reason for (19)(a)and (b)).

(b) Where a particular category of constituent is predicted by the grammar (this is necessary to provide the information X for (19)(c) and (d)).

(c) Where the grammar predicts what state will be appropriate after the constituent has been completed (This is necessary to provide the information for (19)(a)).

(d) Where the grammar predicts that one of several categories is imminent, the (PUSH X) device is the only way to represent these parallel options in one state (e.g. by a state like (21)).

(21)



This displays several aspects of the (PUSH X) notation:

(22)

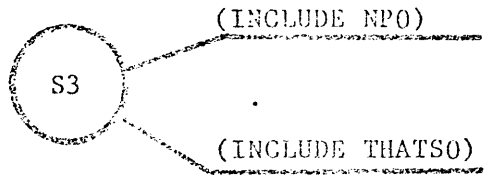
(a) It links into one decision several points which are logically distinct ((20)(a), (b) and (c)).

(b) It forces grammars to be written in a top-down style, since continuation states must be specified, and the grammatical registers are pushed down on the basis of an advance prediction of what category of constituent is coming.

(c) The only way that different states can be merged is to search, top-down, for the various options ((20)(d)). Since this will involve, for each option, the whole register pushing process, it is very messy.

We can deal with (22)(c) first, since it is trivial and largely notational. If there is some combination of options which occurs frequently in a grammar, then there is a good case for constructing a particular state to represent that combination, containing all the component arcs for the various separate networks. This would be slightly redundant, since it would not make use of the fact that these arcs fall into natural groupings according to the various options. What is really needed is a way to handle arcs in small sets, and to produce various states (whose sets of arcs may overlap) by forming unions of these minimal sets of arcs (much as the system of Marcus (1975) uses independent activation of packets). If we introduce pseudo-arcs of the form (INCLUDE X) (where X is a state-name), this will provide this union facility. A state (or rather a pseudo-state) like S3 in (23) can be treated as having all the arcs for states NP0 and THATS0.

(23)



The state-name S3 can then be used in a (PUSH S3) arc, where the pushing down is done once for the whole pseudo-state.

If we examine the decisions and operations initiated by a (PUSH X) arc there seems to be a natural division into what might be termed "global predictions" and "local findings". The former are based on, and are largely about, what is happening at the current level of processing (e.g. what continuation state will be relevant after the incoming constituent); the latter are based on what input comes in

after the PUSH arc, and concern details of the constituent (e.g. what grammatical registers to push down). The global predictions are those aspects of the situation defined by the environment (in particular, the relationship of the incoming item to the existing structure), which are not inherently aspects of the category of item (e.g. NP) involved. The local findings are specific to the processing of the item itself, and ignore how the finished item must be related to its environment.

We can therefore redefine the PUSH command, so that it specifies just those aspects of the situation which will be predictable in advance (start state for constituent, what to do with constituent when found, continuation state if any) and use a different command for specifying those aspects which are better decided within the subnetwork (what structure to start building within the constituent, what grammatical registers to push down). Let us call the modified construct "NEWLEVEL", to distinguish it from the original "PUSH".

On encountering a (NEWLEVEL S3) arc, the interpreter will do the following:

(24)

(a) If a continuation state S2 (not NIL) is specified, this is pushed on to the continuation stack, all interpreter registers are pushed down, and S3 is set as the next state. (This creates a new working level, saving all the information for the current level).

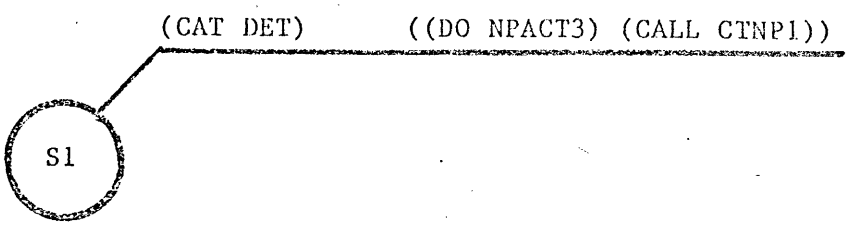
(b) If the continuation state is NIL, the continuation stack is unaltered, the interpreter stack registers are cleared of their current values, and S3 is set as the next state. (This creates a new working level by deleting the information about the current level).

(c) One of the interpreter stacks (call it GRAMREGS) holds a list of the grammatical registers currently in use at the present level; this is pushed down or cleared along with the other interpreter stacks.

That is, the NEWLEVEL arc has no direct effect on the grammatical registers, since what registers are needed can be decided only after the actual input is examined. This means that some construct is needed for explicitly controlling the activation of grammatical registers. Two different devices were tried out at different stages of the development of the MCHINE program. The first one was as follows.

There are in the grammar items called constituent types, consisting of a pair (<list of grammatical registers>, <state-name>). These items are used to describe certain common decisions at the start of processing certain constituents using a command "CALL". This command appears in the action part of an arc, with a constituent type as argument. If there is a constituent type CTNP1, with CTNP1 = ([HEADNOUN], NP3), then it may appear on an arc like (25).

(25)



On encountering the command (CALL X), where X= ((R1....Rn), S), the interpreter should do two things :

(26)

(a) The grammatical registers R1.....Rn are activated; this means that for each Ri the following procedure is carried out. If there is a version of Ri at the current level, it is cleared. If there is none, but there is a version at a higher level, Ri is pushed down. If none exists at any level a new version is created at the current level.

(b) S is set as the next state.

It may seem strange to specify the next state in the action part of an arc, since there is a ready notation for providing the next state (which would not be used on an arc which had a CALL in the actions). The reason for this is that the action of activating certain registers and moving to a particular state seemed to occur together in recurrent pairs, so it seemed useful to group these into units. An arc like (25) is useful at the beginning of processing a constituent, where a specific test (such as (CAT DET)) can be made on the input word, and the parsing continues on the basis of that test.

The reason this device was discontinued was that there did not seem to be enough "constituent types" in the grammar to justify using the concept. The occurrences of state and register list were not as closely related as had been thought, and many of the "constituent types" turned out to have null entries for the register list.

"Constituent types" were therefore replaced, in later versions of the program, by a more direct device, which operates as follows. Two operations PINITREGS ("initialise pointer-holding registers") and SINITREGS ("initialise structure-holding registers") are provided, which take a list of register names as an argument and cause these registers to be activated at the current level (much in the way described in (26)(a) above). This requires the grammar-writer to bear in mind what local registers he needs, and there is no automatic indexing of register-lists to any other items (such as states or rules). There might be a case for a compromise between these two approaches, where a "constituent type" is simply a commonly-occurring list of registers. Then the grammar-writer could keep track of registers more easily by having them in standard clumps, e.g. a list for clauses. Alternatively, it could be made part of the ATN interpreter algorithm to check each state it goes through, to see if there are any associated registers to be activated. Neither of these last two suggestions have been investigated at all.

Levels are terminated in the following way. When a (POP) arc is encountered:

(27)

(a) The grammatical registers active at the current level are deactivated. That is, if previously pushed down, they are popped up; otherwise they are deleted.

(b) All the interpreter stack registers are popped up.

(c) Processing continues from the state stored on the top of the continuation stack.

So far, very few arguments have been presented for the NEWLEVEL...NIL primitive. In fact, it allows more than just the avoidance of specifying a continuation state, as will be seen in Chapter V.

Section III.8 : Decisions, Mistakes and Predictions

If we are to examine sentence-processing in detail, one notion that needs some clarification is that of a "decision". An analysing (or parsing) program makes, as it processes a sentence, various choices and performs various operations - attaching pieces of structure, developing one analysis path but not another, pushing down stack registers, etc. When we talk of a parser making a "mistake" or "wrong decsion", which of these actions are relevant ? The argument put forward by Marcus (see Section III.6) regards backtracking as a form of revoking of "decisions" on discovery of a mistake, and he suggests that people are conscious of these alterations. If this is so, what actions can the hearer revoke without sensing a "mistake", and what re-processing causes a feeling of "oddity" ? If we are to use "garden-path" sentences as clues to what parsing strategies are operating, then our formal model must be explicit about what constitutes a mistake, or we cannot relate anomaly (mistakes defined by the model) to oddity (mistakes detected by the hearer). It conveys very little information to say that a parser makes very few mistakes during parsing, if there is no clear notion of what constitutes a mistake; similarly, it is not a restriction to specify that a parser should not make any wrong decisions, if we do not specify what counts as a decision.

We can tackle this by reviewing some of the kinds of operations that an ATN interpreter performs while it is processing a sentence.

(28)

(a) Testing an arc, using the current word.

As pointed out in Section III.6, there is a sense in which the arcs of the current state predict, since they define the relevant options at the current stage of the analysis. In a normal, unadorned ATN there is no notion of "prediction" or a "decision" in making the individual tests, since the tests are just a way of getting the input word to guide the next step in the analysis. No special structure is built for each arc, and the state of the analysis is not altered until after an arc test has succeeded. As observed in Section III.6, the tests could be regarded as demon patterns which the input may trigger. Hence choosing to test a particular arc will not be regarded as a "decision".

(b) Jumping to a new state without taking in a new word.

This generally means that some further testing is being performed on the current word . It is therefore part of carrying out one of the arc-tests in the original state and hence does not yet constitute a decision to follow that option. Such jumps occur as part of the processing of a (PUSH...) or (NEWLEVEL...) arc (see Section III.7), where the state jumped to is the start state of the subnetwork given.

(c) Taking in a new word and jumping to a new state.

This might seems to represent a decision, and the jump to the new state might prove to be "wrong" (i.e. none of its arcs match). However, this is bound up with the hazy question of limited

lookahead.

Sometimes it is the case in a recognition grammar that a particular decision will be resolved completely once the next input word has been processed. A look-ahead of one word would thus avoid making an unnecessary branch point in the analysis. (The TESSA parser (Soul (1975)) includes a one-symbol lookahead). If the information to be extracted from the next word can be simply expressed (e.g. "is it the word 'not'?"), then the necessary look-ahead can be easily programmed in as an extra condition on an arc. However, in many cases, the information required is exactly a complete parsing of the next word in context (e.g. "can this word start a suitable noun phrase ?"). This could also be directly programmed in, but it would require applying a full set of arc-tests, for some state in the grammar, to the word. This would be slightly redundant, since, if the test succeeded, the parser would then proceed to exactly that state, and carry out all the processing again. Also, to be certain that the look-ahead processing was valid (in terms of the context it was performed in), all the modifications (e.g. movement of the sentence pointer) would have to be made that are part of jumping to the state in question anyway. It might be better just to allow the parser (in these more complicated cases) to continue processing, but not to regard this as non-determinism. That is, branches which last for only one word are not to be looked on as "mistakes".

(d) Pushing down the interpreter stack registers.

As commented already, one of the actions taken on encountering a (NEWLEVEL...) arc (namely, using the start state of the subnetwork as a further set of tests) is not really a "decision". If the analyser does this by jumping into the subnetwork, and using the start-state as the expanded test, then the environment must be adjusted accordingly. The interpreter stack registers (or some of them) have to be pushed down, so that there are new pointers for certain data-structures (the node being currently processed, for example). Such stack alterations are a preparation for a sub-constituent which may need independent work space, but they do not constitute a prediction about what is to come next (since they are not specific to the subnetwork involved). It seems reasonable not to regard the alteration of interpreter stacks as a "decision" (Woods (1970) states that (non-augmented) transition network grammars can be optimised so that the only non-determinism is in the push-down mechanism, which suggests that there is a qualitative difference between the two kinds of operation).

(e) Pushing down a set of grammatical registers.

Grammatical registers are slightly different. The registers needed for a noun phrase are different from those for a clause, and there is no need to push down all the registers for each new constituent. Choosing to push down a certain set of grammatical registers constitutes a decision to process a particular kind of constituent. As pointed out in Section III.7, this part of the (NEWLEVEL) operation can be postponed until the first word of the input has been tested, and this word can influence the choice of registers.

(f) Storing a structure in a register.

This might seem not to be a decision, since one of the advantages of using structure-holding registers is supposed to be that structure-building decisions can be postponed. However, this depends on how the registers are used. Woods (1970) states that using registers enables the postponement of decisions until the relevant information is available, instead of guessing and then altering the situation later (p.601). In fact, in the examples he describes, the decisions are made at an early stage and then changed if wrong. The grammar he gives makes quite specific initial hypotheses, by placing structures into particular registers, then alters these decisions if necessary by moving the structures to other registers. There are several labelled registers, one for each aspect of the syntactic analysis being produced, and the contents of these registers at the end of the parse define the analysis constructed. That is, the assignment to structure holding registers is effectively structure-building, since no separate tree needs to be built out of the contents of these registers later. Hence the sample grammar does make premature guesses about the sentence structure and revise them later - using labelled registers instead of an explicit tree-structure does not alter this fact. Registers could be used to postpone decisions by having them fulfil genuinely temporary roles, but regarding them as labelled slots in the final analysis does not do this.

(g) Attaching a structure to the main surface structure.

Structure-building of this sort is a definite decision, since it directly determines the final form of the analysis, without any further manipulation by the parser of that particular relationship. If we are to have any notion of "irrevocable decision" it seems plausible that structure-building should fall into this category. Winograd makes no attempt to restrict the re-structuring that may go on within a parse, and uses a modified form of depth-first top-down exploration, which allows arbitrary revoking of decisions.

Although it is probably easier to write a parser without constraints on decision-altering, it is not helpful if we wish to develop a detailed model of the decision-making process. The MCHINE program (Chapter VI) included an attempt to write a parser which made as few "mistakes" as possible, and this severely slowed down the programming. The actions which were regarded as "decisions" were structure-building, flag-setting (although the MCHINE grammar does not resort to using flags at any stage), altering the state of grammatical registers, and assignment to specially-named grammatical registers. Since the program uses a modified breadth-first approach, mistakes show up not as backtracking but as analysis paths which terminate prematurely. In view of the comments above about one-symbol look-ahead, branches which are terminated after existing for only one word are not regarded as "mistakes". Any other processing which does not involve taking in a new word (e.g. jumping to a new state) is not regarded as a "decision".

Section III.9 : Bottom-up Devices

One complaint sometimes made about language analysers such as those in the LSNLIS and SHRLDU systems is that they are "too top-down" (c.f. Marcus (1975)). That is, the actions taken by the analyser are determined to too great an extent by the type of input that it is searching for, rather than by what input it has actually received. This section illustrates that certain modifications to the basic ATN/ PROGRAMMAR type of system can introduce a greater degree of influence by the input.

Pre-tests in PUSH or NEWLEVEL arcs

Marc Eisenstadt (personal communication) has made the following observation regarding ATN grammars describing embedded constituents. Suppose a grammar defines one of the possibilities for "sentence" to be a "question"; one of the possibilities for a "question" is a "wh-question"; a "wh-question" starts with a "wh-phrase"; a "wh-phrase" starts with a "wh-word". In analysing the first word of a sentence, such a grammar might go through the following sequence of arc-conditions :

```
PUSH <Question>
    PUSH <WH-question>
        PUSH <WH-group>
            PUSH <WH-phrase>
                FEATURE WH
```

At this stage, the input word is tested; if it turns out to be "does" (in keeping with the hypothesis of a question), this analysis path fails - but it has to nest several levels just to achieve this. The recognition rules have decomposed the initial task very straightforwardly into simpler subtasks, so all the subtasks have to be initiated, even although the input word is unsuitable. This extra work could be avoided if the grammar-writer was able to include pre-tests in some of the PUSH arcs. Instead of a simple (PUSH WHQUESTION) indication, ordered conjoined tests could be included, e.g. ((FEATURE WH) (PUSH WH-QUESTION)). Thus, the full search would be made only if the initial word is suitable.

Notice that this is slightly different from the device used in the program of Thorne, Bratley and Dewar (1968), where the ATN interpreter automatically carried out a form of pre-testing for any subnetwork, as a form of optimisation. Here, the grammar-writer can include an explicit pre-test in the ATN grammar, in cases where he requires some very specific check. It is worth noting that simply by allowing conjoined tests, a pre-test can be incorporated anywhere in a grammar.

Some difficulties still remain. It will not necessarily be easy to find a simple pre-test which is appropriate for a particular kind of constituent, although (FEATURE WH) and (FEATURE VB) are obvious examples for WH-clauses and verb phrases. Ordinary noun phrases, for example, seem to have no natural class of initial words, since; for example, "any", "black", "bananas" and "Harry" can all start a noun phrase. (In the MCHINE grammar, the rather dubious feature "STARTNP" has been included for this purpose, but that is not very

satisfactory. Dewar(personal communication) has stated that the automatic pre-testing in the Thorne-Bratley-Dewar program relied on the presence of suitable category markings, and this also lead to some otherwise unmotivated category assignments).

Dynamic Grammar-building

It may occur that a part of the recognition rules of a grammar shows some internal pattern with certain clearly-defined variations. What is needed is some way for such generalisations to be extracted, thus simplifying the specification of the grammar. One example of this occurred while designing the MCHINE grammar for verb phrases, and the solution adopted has the consequence that the analyser is strongly guided, when analysing verb phrases, by the properties of the input. The situation is as follows.

Verbs in English may have two, one or no object(s), where "object" is used loosely to mean, roughly, "any post-verb constituent whose meaning is to be considered as an argument for the main verb-relation". (See Section V.8 for a fuller discussion of English verbs). This could be described by allowing three options in the grammar, and analysis paths could branch depending on whether the verb required two, one or zero objects. However, the situation is slightly more complicated. The surface forms that the objects may take can vary greatly and will usually depend to some extent on the particular verb involved. If this information were to be incorporated directly into the grammar, the number of options (arcs) would expand excessively. On the other hand, the information concerning the quantity and surface form of the objects can be stored

in the lexicon for each verb, and the grammar need only contain actions which extract this information and use it to build, in the course of the analysis, a section of transition network which sets out the relevant objects.

For example, in the MCHINE program, the lexical entry for a verb includes a list of object-information lists. An object-information list gives the name of the structural combining rule and the names of certain ATN states - one state for each object involved. The analyser, on encountering the verb, consults this part of the lexical entry; a surface subtree is established using the combining rule, and the states are used to construct the ATN which is then used for the next part of the analysis.

In this way, the pattern (that the verb idiosyncratically determines the number and form of the objects) has been extracted and built into the actions which perform the grammar-building. As a consequence, the verb can guide the analyser more strongly than if all possible object configurations had to be tried (see Section V.8 for further details).

Processing units triggered by words

Winograd suggests that certain words could be regarded as "demons". That is, instead of the recognition grammar having rules stating what to do with a particular word, the lexical entry for the word should specify an action to be performed on encountering that word (see Section II.9). In the examples he gives (conjunctions), the actions involved are quite sweeping, as they radically alter the flow

of the analysis process. Winograd observes that such a mechanism is not allowed in the ordinary ATN formalism, but that is slightly misleading. The PROGRAMMAR system in which SHRDLU's parser is written allows two varieties of input analysis - (PARSE X...) statements and demon words - and the PROGRAMMAR interpreter must be written to handle both. In the same way, an ATN interpreter could easily be written which would examine each input word to see if it was a "demon" before passing it on to the arc-testing routines. Such an interpreter would increase the similarities between the ATN and PROGRAMMAR methods.

Many English constructions have the consequence that an initial substring of a sentence will itself resemble a complete English sentence, e.g. (29) and (30).

(29) He wrote several letters after eating.

(30) I saw the full moon when I was in the garden.

One way to allow for this in the grammar is for such constructions to be sought explicitly in a two-stage top-down manner - find a clause, then find the second constituent. This has the slightly inelegant consequence that every sentence is ambiguous right to its clause end. That is, when analysing (30), the analyser has to consider several possibilities (one for each possible optional constituent at the end), including the case of the full sentence being (31).

(31) I saw the full moon.

Intuitively, what is needed is some way for the analyser to process

the initial clause on a single path, and then consider the second clause only if it in fact is present. One way to do this would be to make the grammar contain a form of loop - the sentence-final state could be linked back to the sentence-initial state (or clause-initial state). This possibility has not been explored here.

A more interesting approach is to let the opening word of the second clause directly influence the way that the analyser restarts its processing. One way to do this is as follows. Syntactic features appear in various word-tests in the recognition rules, and hence any set of syntactic features implicitly defines a subset of rules (of arcs, in an ATN), namely, the set of rules such that some feature from the set appears in the condition. Hence we can define a "restart" mechanism for the ATN interpreter : on reaching the sentence-final point in the grammar, if there are still input words to be processed, the interpreter should use the feature-list of the next word to construct a set of appropriate rules (the set of arcs associated with the feature-list, in the way described here) and continue processing using these rules.

Notice that this is different from Winograd's "demons", and is aimed at covering a different set of phenomena. A demon word takes over the analysis process wherever it occurs, and supplies all the necessary actions. The "restart" facility is used only when there is no currently active set of rules (i.e. no non-trivial arcs in the current state) and there are more words to be processed; under any other circumstances the implicit mapping from words to rules is ignored.

It might be interesting to try to generalise the "restart" mechanism so that words can select rules in situations other than clause-endings, (e.g. at the start of the sentence), but this has not been explored here. A simplified "restart" system has been incorporated successfully in the MCHINE program.

Selecting Structural Rules by their Arguments

If a recognition grammar searches directly for two related constituents, the recognition rules can specify directly what building rule should be used to combine the meanings of the two items once found. However, if the constituents are found by some less top-down, explicit method (e.g. analysing a clause, doing a restart (as described above), then analysing a time-adjunct clause), there may be nowhere in the recognition gramar to specify which combining rule to use. This deficiency can be overcome partially by having a technique which can select a combining rule on the basis of what potential arguments have been found so far. (The crudest way to implement this would be to have the analyser search the whole set of combining rules for one for which the found items would be suitable inputs).

This still leaves some difficulties. Firstly, if the combining rules have an input specification in terms of a list of simple predicates, a list of potential arguments may satisfy the input conditions of several different rules, particularly if some rules have very broad input conditions. This could lead to multiple ambiguity at the stage of rule-selection. This problem can be reduced by refining the classification of semantic structures so that

each rule specifies as narrowly as possible what its inputs should be. However, in any non-trivial grammar, several combining rules will have overlapping input specifications to an inconvenient extent. Section VI.4.5 gives a more detailed description of the problems which arose when debugging the MCHINE grammar as a result of these difficulties. The whole question of determining inter-constituent relationships on the basis of the semantic structures of the constituents needs much more investigation.

Section III.10 : Semantic Representation

The focus of this project is on the way that processing occurs during sentence-analysis, rather than on the kind of structure eventually produced. However, it is impossible to study processing in isolation, particularly if a computer program is to be written. This section discusses some of the possible ways of representing "facts", as a preliminary to defining (in Chapter IV) a semantic system which is at least adequate enough not to vitiate any of the rest of the framework. Let us consider some of the criteria that a semantic formalism should meet.

It will have to be general, in two respects. Firstly, it must not be specific to the subject matter involved; secondly, it must not be specific to the language involved. It should allow the representation of any meaning that can be expressed on any subject in any language.

It should interface closely with other parts of the language model. That is, there should be ways of relating the semantic structure systematically to the more structural aspects of language. This may seem an irrelevant comment, but it could be argued (see Section II.7) that one deficiency of Schank's work is the lack of a clear theory of the relationship between surface form and conceptual meaning. Semantic structure should not exist in isolation, however splendid.

It should provide some way of comparing or relating semantic structures, for example by rules of inference, so that "meanings" of different sentences can interact in some way (although not necessarily in traditional syllogisms, for example).

Other desirable attributes of a representation system might be simplicity, perspicuity and having self-evident atomic constructs, but these are less important.

There seems to be a consensus forming within artificial intelligence and linguistics over semantic representation. The systems proposed by Wilks (1973), Schank (1972a), and Rumelhart and Norman (1973) have many similarities. All are proposing systems which purport to be independent of subject matter, and Schank and Rumelhart and Norman claim that their representations are language-independent. (The latter claim is largely unproven, since the systems are mainly illustrated with examples from English or related languages).

Two kinds of devices are used in these systems to express relationships between different meanings. Firstly, there are rules of inference (which are generally discussed informally, so that it is not obvious what canonical form(s) are being proposed, if any). Secondly, meanings (of particular words, etc.,) can be decomposed into smaller units ("primitives") so that similarities in meanings can be displayed in the configuration of primitive elements. Both inference rules and primitive-decomposition are very difficult issues, for which no one has found any good solutions (see the papers in Schank and Nash-Webber (1975)). The main problems in both are when to operate the mechanism, and when to terminate it - how many

inferences should be followed up at any given point, or how much breaking down into primitives should occur?

There is another problem for a system which operates entirely in terms of primitive elements. Some information about particular meanings or concepts will have to be associated with the larger semantic items, rather than with the component primitive elements. To take an example suggested by Wilks (lecture at Edinburgh (1976)), there is information about "smoking" which cannot properly be tied to its representation in terms of "drawing smoke into the lungs through burning tobacco"; also, Charniak has suggested that some aspects of the meaning of "sweat" cannot be indexed under the component items like "water" and "skin". A primitive-based system might associate such information with sub-structures made up of primitive elements, but this would be to acknowledge the validity of these larger "chunks" for some descriptive purposes.

There is an added problem for any system which describes all meanings by associating a static structure with each sentence. Some words or sentences may be better expressed in terms of something other than a simple relationship between the few items involved. For example, if we are to use the meaning of (32) to make any

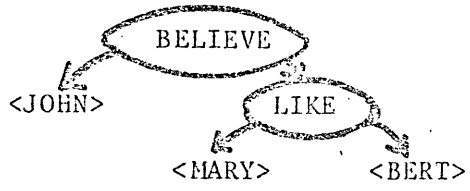
(32) John believes that Mary likes Bert.

(33) Does John think that Mary likes Bert ?

deductions concerning John, it would be useful to have it recorded (somehow) that a particular proposition (namely, that Mary likes Bert) has a particular truth-value in John's model of the world.

Then (32) could be related to (33), for example, without a specific inference rule relating "thinks" and "believes". A simple semantic network like (34) being designated as the "meaning" of (32) would not be very helpful.

(34)



Rumelhart and Norman allow each relation in their semantic system to have an associated piece of program, which can be executed under particular circumstances. This general facility allows the person defining the relations to include arbitrary special effects; in particular, the procedure could be used to re-express the relation in terms of some other conditions on the semantic network.

The semantic system adopted in this project is very similar to that of Rumelhart and Norman. There are "relations", which can be used to form a general relational structure (a semantic net). Each relation has a fixed number of "roles" which can be filled with other semantic items, each role having an associated "restriction" which limits the kind of item which can fill that role. Facilities are provided for both primitive and procedural re-expression, as follows. Each relation can (optionally) have an "expanded form" and an "elaborated form". The expanded form, if present, is a piece of semantic network which expresses the relation in primitive form. Some notational device is necesary to keep track of how the arguments of the main relation fit into the primitive network. The system thus

includes both non-primitive and primitive relations, and the former may be re-expressed in terms of the latter.

The elaborated form (if present) can be thought of as a set of three procedures - a procedure for testing if the relation holds, one for making the relation "true" in the main network, and one for making the relation "false" in the network. (In the implemented version, these three functions are fulfilled by a single structure, interpretable in three different ways - see Section VI.3.7). When matching two pieces of relational network, for similarity, or testing to see if a relation is "true" in the the network, the expanded version can be tried as well. When testing or setting the "truth-value" of a relation, the elaborated version can be used as well as the main relation itself. In this way, the meaning of some relation (e.g. "believe") can have alternative expression in terms of conditions on configurations of other parts of the network.

The interface with other aspects of the linguistic model is achieved by having the SCRs build up semantic networks gradually, so that each subpart of a sentence has its own associated semantic structure. The semantic part of lexical entries for words are always pieces of semantic network. In particular, each main verb has an associated relation. This means that any idiosyncratic parts of the meaning of a verb can be associated with that relation, and its connection with other meanings can, if necessary, be included via the expanded or elaborated forms. The set of roles for a relation thus provides the "case-frame" (see Section V.8) for the verb, and the SCRs which would be regarded as "role-placement" rules, (since they fit meanings into the case frame) are simply building a semantic

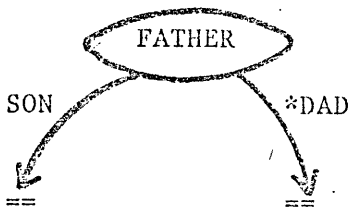
network around the relation of the main verb.

There is one kind of semantic structure which has been extremely useful in finding representations for the meanings of several categories of item. If we take a piece of semantic network and select one of the roles (whether filled or not) as a "focus of attention", the resulting structure can be used in various ways. (In the implemented version, these structures are represented as a pair consisting of a semantic network and the name of the selected role). These structures can be created during processing, if a semantic rule selects some role in a piece of network, or they can be entered directly in lexical entries, with the role already selected.

Let us call such a structure a "definer". It can function as a predicate, which is "true" of all items X for which there us a "true" network matching this one, with X in the selected role. It can be used to find a particular spot in the overall semantic network by finding a piece of network to match, and then singling out the selected role; thus found, any item located at that spot can be examined, or another item can be placed there.

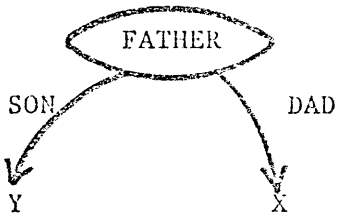
Not all the roles in the piece of network in a definer need be filled for it to operate successfully. Suppose we have a relation FATHER, with roles "SON" and "DAD", which can hold between sets of items (classed in the semantic model as "PEOPLE"). We could construct a semantic item for the word "father" by a definer like

(35)



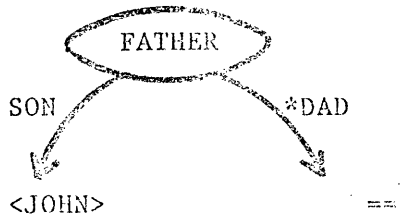
where == is a "blank" entry and * marks the selected role in the definer. This will act as a predicate which is true of any item X for which there is a network like (36) recorded as "true", where Y can be any item.

(36)



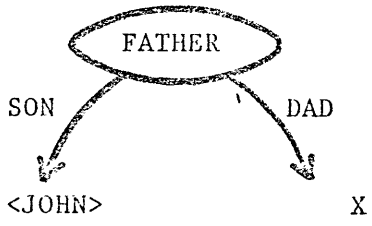
This allows for the use of "father" without mention of the offspring, e.g. "Harry is a father". If the other role became filled (e.g. in the structure for the phrase "John's father"), the semantic item would then look something like (37), where angle brackets enclose an item whose internal details are not important here.

(37)



In this case, the definer can still operate as before, except that the range of semantic networks that it will match is much narrower (since one of its roles now must match a specified value <JOHN>). It could, for example, act as a predicate which is true of any item X for which there is a "true" network like (38).

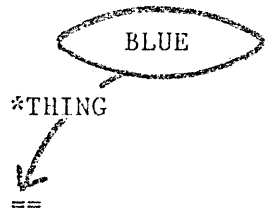
(38)



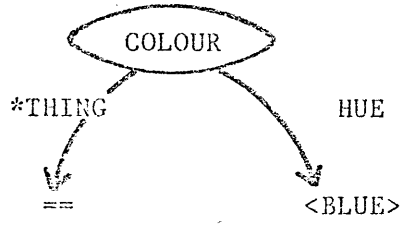
The same definer (i.e. (37)) could also be used to find a position in the network, and the item there (e.g. X in (38)) could then be accessed or replaced. That is, one representation for the meaning of "John's father" can act as a predicate, a way of finding an item, or a path to some spot in the network where some item can be placed. (The latter might be useful for interpreting a sentence like "Dave is John's father", for example).

The other versatile aspect of the "definer" is the wide variety of grammatical categories for which it can be used to represent the meaning. As well as using a definer as the semantic item for a noun (as above), the fact that a simple regular adjective can be regarded as a predicate means that a definer can represent the meaning of an adjective, e.g. for "blue" we could have (39), or even (40).

(39)

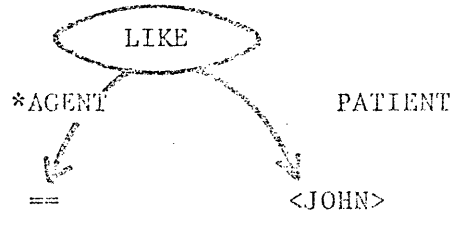


(40)



Since verbs are represented by relations, and case frames by the roles of a relation, role-placement can occur in what is traditionally called a verb phrase. The meaning of a verb phrase can be represented by a definer, where the semantic network part has the selected role unfilled, so that the "subject" of the sentence can be fitted in there. A phrase such as "likes John" would then have a definer like (41) as its meaning (see Section V.8 for more details of verbs and case structures).

(41)



This simplifies the subject-complement rules. Although (42)(a)-(d) have diverse surface structures, the underlined parts can all be represented by some kind of definer.

(42)

(a) John is Harry's father.

(b) John is a doctor.

(c) John is stupid.

(d) John likes bananas.

The subject-complement rule can be designed to insert the semantic structure for the subject into the selected role in the definer from the complement, thus producing a semantic network.

(Bach (1968) discusses certain semantic patterns which cross traditional syntactic boundaries and which seem similar to the generalisations that are attempted here).

Several details have been omitted or glossed over here, since they have not been worked out fully. The notion of a "definer" proved very useful in writing the MCHINE program, and its characteristics seem interesting enough to suggest its general applicability. Some more points are discussed in Chapters V and VI, but not every detail has been perfected yet.

Semantic representations can be classified in many different ways, and it is advisable (to avoid confusion in later chapters) to explain here some of the kinds of semantic categories used in computational grammar. (See Section III.11 below for some comments on classification). The system of semantic representation described here gives rise to three different kinds of semantic classification, as follows :

Referential Classes

The items in the subject matter of the discourse or dialogue can be classified according to their characteristics in the external world. This area was much debated within transformational grammar in the 1960s (under the heading simply of "semantics"), and some of the more intransigent problems are summarised by Bolinger (1965).(See also Katz and Fodor (1963), Weinreich (1966), Katz (1967)). In the MCHINE program, each referential class is represented by a predicate which tests for membership of that category (the predicates are represented as "definers" - see above). The referential classes are structured into a hierarchy of sub- and super-classes, and grouped into antonym classes (as proposed by Katz (1972)) for the purposes of describing "semantic anomaly" (see Section VI.3.11).

Sense Classes

The linguistic representations of meanings (that is, the semantic structures, not the referents) can also be described in terms of their semantic network structure. Different kinds of structures (e.g. "relations", "definers", etc.) have different capabilities for combining with each other, so it could be said that the sense classes describe the "abstract syntax" of the linguistic items.

Sense Properties

As well as the gross network structure of a semantic item, the grammar may need miscellaneous information about how to process the item. For example, "definer" is a sense class, but definers can be used (as described above and in Chapter V), to represent

miscellaneous linguistic items, and they may need different annotations to indicate how they are to be processed. Properties like "definite", "specific", are the clearest examples, giving details of what matching and instantiating should be carried out on the semantic items they are marked on (see Section V.6). The meaning of a relative clause is also representable as a definer, but with different sense properties to record the fact that it must be used in slightly different ways in the semantic routines.

Section III.11 : Levels of Description

There are several ways in which a linguistic constituent could be classified: what its internal structure is, how it is related to other constituents, what steps are necessary to parse it, where it can occur in a sentence, what kind of meaning it has. These six criteria for classifying are logically distinct, and might be described as morphology, syntagmatic relationships, parsing method, distributional (or paradigmatic) behaviour, semantics. Traditional syntactic categories usually try to summarise some or all of these under one label (e.g. "Noun Phrase"), which makes an implicit claim about how these different factors are related. There are some interesting relationships between these classifications (e.g. the internal structuring of an item determines how it must be parsed) but it will lead to confusion if we assume immediately that all 6 give the same classification.

For example, consider articles and possessives in English. Articles ("a" and "the") are paradigmatically related to possessive adjectives ("my", "Fred's", etc), in that they all occur at the beginning of a noun phrase, and there cannot be both an article and a possessive at this stage in a noun phrase. This suggests we might simplify the grammar by creating a category or feature (say, DET) which includes both, and not distinguish the two. However, this ignores the possibility that the syntagmatic relationship between article and head noun may be different from the relationship between possessive and head noun. If our grammar is to represent such relationships explicitly (as was suggested in Section III.1), it may

be necessary to distinguish the two subclasses in the syntagmatic representation, even although they are treated similarly during parsing. Winograd(p.93) has his parser attach possessives under a "DET" label on the syntax tree, and has a class of "determiners" which includes both articles and possessives. This is largely because syntagmatic relationships are not represented in his syntactic tree, but are worked out later by the semantic specialists. Hence his syntactic labelling need not be too fine. It might be asked what the criteria are for assigning different syntagmatic relationships to different combinations. The answer to this is that if different semantic operations must be performed on the combination, then a different relationship is required. This begs the question until we have some clear idea of "different semantic operation", but, as will be argued in Section V.1.1, there is a case for regarding possessives as semantically different from the other "determiners".

Related to this is Winograd's all-embracing use of "features" to classify every aspect of a syntactic unit, which is as confusing as the traditional packing of all characteristics into syntactic categories. Some features are distributional/ paradigmatic, and guide the parser - e.g. DET in the lexical entry for "the". Some features are morphological, and describe the internal form of the unit they are attached to - e.g. NDET on a noun group node means that there is no determiner attached beneath that node. Some features are syntagmatic - e.g. AGENT, when attached to a Prep Group node, describes the relationship of that node to the main verb; when attached to a Clause node, it describes the internal relationships between the items of the clause. Some features are both syntagmatic

and semantic - e.g. NPL marked on a noun group node indicates its number. Some features appear to be wholly semantic - e.g. DEF or INDEF marked on a noun group. The major categories like NG (noun group) contain similar conflations - NG is the name of a parsing method (as in a command (PARSE NG...)); NG is a feature marked on a node; there are a particular set of semantic specialists associated with the category NG.

Marcus (1974) suggests that a feature is any aspect of a constituent that the parser may need to find out by a quick inspection. This sums up one important aspect of features in a recognition grammar (they are there to guide the parser) but it is not very perspicuous to use the same device for describing disparate aspects of structure without indicating the distinctions. This confusion is also present in other areas of grammatical classification, particularly around the boundary between syntax and semantics.

Consider the underlined phrase in (43) :

(43) The man who you saw last night rang up.

There are several statements that might be made about this phrase and/or its meaning. It is a noun phrase. It is made up of a noun group and a relative clause. It is made up of a nominal and a modifier. It is a referring expression, and is definite and specific. It is the subject of the verb in the sentence. It refers to a human, animate thing (or should do, if used appropriately). These make seven different classifications, and cannot be covered simply by providing one set of "syntactic" categories and one set of

"semantic" categories. Strangely enough, the hardest classification to make firm is the traditional "noun phrase". All of (44)(a)-(f) could be classed as noun phrases, and they differ in many respects.

(44)

(a) The dog attacked him.

(b) Flying kites can be tricky.

(c) To have been loved is better than to have been lost.

(d) I don't like what you did.

(e) It amazed me that you arrived here.

(f) I will ignore anything you may do.

It is very hard to lay down levels of classification in isolation from an overall model, since different descriptive frameworks may make different distinctions (e.g. the "deep structure" of Chomsky (1965) is a disputed level). Computational grammar includes various concepts and rules which can be used to induce classifications of linguistic structures at various levels. Some of these categories fit easily under the headings of syntax and semantics, but others are less easy to allocate. The kinds of categories and rules available are the following, grouped under headings that may help to indicate the level at which they operate.

(45)

(a) Concrete Syntax : syntactic properties, syntactic features (see Section III.2)

(b) Abstract Syntax or Intensional Semantics : structural combining rules (see Sections III.1, III.2, III.3), sense classes, sense properties (see Section III.10).

(c) Referential Semantics : referential classes (see Section III.10).

Notice that the only categories which may vary with a change of subject matter are those at level (c) - all other classifications should be general linguistic statements. It is not the case that all semantics is dependent on the domain of discourse - the level of intensional semantics ("sense" as opposed to "reference") should be domain-independent.

These ways of describing items take on meaning only once the full framework has been described in Chapter IV. Drawing attention to these distinctions should clarify the exposition, and avoid the need to class every concept as either "syntactic" or "semantic". Traditional concepts (e.g. "noun phrase", "subject") could be defined in this framework by fitting them into this system. For example, computational grammar does not use the concept "subject" as such, but certain SCRs could be classed as "subject-complement" rules, and an item could be said to be a subject if it is the first argument of a subject-complement rule.

Section III.12 : Conversational structure

Winograd's program did not include any systematic treatment of how one utterance relates to others in a conversation. A question from the person talking to SHRDLU was translated directly into a program for producing an answer, and the reply was given immediately. The only example in the sample dialogue where one conversational exchange (question + answer) seems to be inserted inside another is example 22 (pp.12-13), where an ambiguous phrase is queried before the original question is given a reply. In this case, a stereotyped form is used, and the human's reply must be not another English sentence, but an integer. Hence this is hardly a "conversational exchange". Pronoun reference in the program uses preceding utterances, apparently by consulting a list of the most recent sentences. Presumably the program has the equivalent of a LISP (READ-EVAL-PRINT) loop, where each sentence is processed on its own, before the program goes into a "wait" situation for the next sentence. This has certain disadvantages.

Firstly, a human hearer can generally choose to answer a question or not. Although this is difficult to simulate in a program without including a set of "beliefs" or "goals" which might influence this choice, it is reasonable to have this decision made at an appropriate level in the model. Winograd's approach builds the choice into the semantic interpretation of the sentence, so that "understanding" a sentence includes deciding to answer it. It seems more plausible to have some level of description of conversational behaviour, at which such decisions (relating to the illocutionary and

perlocutionary force of the utterance (cf. Austin (1962)) can be handled.

Secondly, we need to make a distinction between the sentence-type of an utterance (declarative, interrogative, or imperative) and its illocutionary force (a statement, a question, or a command). For a simple conversation, there is a simple one-to-one relationship between the two, and the distinction may be overlooked. However, these are logically separate categories - interrogatives can serve as (polite) commands, and statements can be orders:

(46)

(a) Can you pass me the fingerbowl ?

(b) You are not going to the party.

It might be argued that requests (e.g.(46)(a)) are idioms, but that would not alter the fact that the intended response by the hearer is not the supply of information, despite the fact that the surface form of the utterance is interrogative. This suggests the need for separate levels of description, which separate the hearer's reaction to the utterance from the semantic/ syntactic form of the sentence. (Winograd's sample dialogue includes one command starting "Will you please...", but it is not clear how it is handled by the program).

If utterances are processed on a one-off basis, then there is no way in which different exchanges between the interlocutors can be nested inside each other. To generalise Winograd's ad hoc mechanism for resolving ambiguity, we need some way that the hearer can suspend

his reaction to a question while he seeks further information from the questioner. This may not be a simple clarification of the question, but may be, for example, information about what would constitute an adequate answer for this particular questioner, e.g.

(47)

A : Where is the Bionics Research Laboratory ?

B : Do you know Sandy Bell's pub ?

A : Yes.

B : The Bionics Lab is behind that.

Another related improvement that is desirable is some way of relating utterances to higher goals. If a sentence is treated as an isolated string of words, there is no way of describing how it fulfils a function in a dialogue. If a person wants to find out information, he has to ask a question and know how to use the answer. He must also have some idea of what constitutes a suitable answer, and how to react if this is not given.

To sum up, an adequate description of the conversational use of language will not treat a dialogue as a disconnected series of self-contained utterances.

CHAPTER IV

COMPUTATIONAL GRAMMAR

Section IV.0 : Preamble

Chapters I, II and III have provided the background to the work which is described in Chapters V and VI. Here is a summary of the model (or partial model) which has been adopted. It is based on the work of Woods and Winograd, but inspiration has also come from much of the other work discussed in Chapter II, to varying extents.

It is worth giving a brief outline of the assumptions and decisions so far discussed. (The numbers in brackets refer to the relevant sections in Chapters I and III).

This project is an attempt to examine the structure of the English language and the way that the structure could be used by a hypothetical hearer. The research strategy is to write a recognition grammar for a subset of English (or a series of fragments of grammar) (I.5). Sentence processing proceeds in a strict left-to-right order and as much semantic processing is carried out at each stage as seems feasible. (III.3). The analyser attempts to make only those decisions which are justified at any given point at the input, rather than using exhaustive, mistake-driven techniques (III.6, III.8). To achieve this, registers are used to avoid premature structure decisions (II.5, III.8). In describing grammatical phenomena, a clear distinction is made between syntagmatic, paradigmatic and other types of description (III.11), so that these dimensions may be treated independently if necessary. No separate syntactic structure is built, but a tree structure based on hierarchically organised structural combining rules is used instead (III.1, III.2, III.3). There are many different structural rules, as they have to make all

the distinctions previously made by syntactic rules and semantic rules (III.1). Where the analyser encounters possible ambiguity, it develops both paths in parallel, but tries to order the paths relative to each other (III.6). Constituents are processed at separate levels, with independent local register space, in a way that avoids the creation of too many different levels (III.7). The semantic system includes some form of sense-reference distinction, and allows expressions to refer to non-existent objects (III.4). Syntactic classification is in terms of binary features aimed at guiding the analyser explicitly, and syntactic properties to guide the structure building routines (III.2). A system of conversational rules is used to describe certain aspects of sentence usage which are properly not included in the sentence-grammar (III.12).

The framework will be described in eight sections, as follows :

IV.1 Structural Combining Rules

These combine semantic items to form other semantic items.

IV.2 Recognition Rules

These direct the flow of input-processing.

IV.3 Semantic Representation

This defines the constructs available for building meanings.

IV.4 Syntactic Properties and Features

These markings allow arbitrary processing information to be marked on structures.

IV.5 Analysis Procedure

This sets out how the sentence interpretation proceeds.

IV.6 Registers

These are the facilities for storing information during sentence-analysis.

IV.7 Conversational Routines

These have not been greatly developed, but are included to provide a level of "paralinguistic" description.

IV.8 Guidelines for Analyses

Some informal rules are provided for applying the apparatus of Sections IV.1 - IV.7.

No formal definitions are presented here, and all the concepts are described in an informal way. The outline is brief, but Chapters III and V present the arguments in favour of the various methods, and Chapter VI offers a possible elaboration of the details.

Section IV.1 : Structural Combining Rules

These rules are on the conceptual boundary between "syntax" and "semantics", and so are hard to allocate in either category. They are very similar to the "projection rules" of Katz and Fodor (1963), or the "semantic specialists" of Winograd (1972), since they combine semantic items to form other semantic items. On the other hand, the gradual setting up of rules and arguments while processing a sentence (see Section IV.5 below) results in a tree-building process like a traditional synactic parse. One way to look at a structural combining rule is to regard it as the pairing of a Montague syntactic rule with its corresponding semantic rule (Montague (1970)).

A structural combining rule (SCR) consists of :

(48)

(a) Rule body : the operations to be performed on the inputs. At present, these can be any manipulation, and there is no basic set of "primitive semantic operations".

(b) Input specification : A list of semantic predicates, one for each argument-place, which states what kind of items are allowable as arguments (like type restrictions in some programming languages). The semantic predicates will be composed from sense classes and sense properties.

(c) Output specification : This allows the result produced by the rule to be explicitly labelled with some semantic classification. The output specification will be either a referential class or a

function which produces a referential class by combining information about the inputs to the rule in some way. (See Ritchie (1976) for some of the reasons for this).

(d) Property inheritance rule (optional) : for each argument place, a list of syntactic property names. This is used for handling temporary structural information, during the sentence analysis process (see Section IV.5) and does not directly affect the semantic structure produced by the structural combining rule.

The way that structural combining ruless are used will be explained in more detail in Section IV.5.

Section IV.2 : Recognition Rules

The linguistic model will use the concepts of an ATN/PROGRAMMAR system subject to the modifications in Chapter III . The exact notation is not important here (the representation used in the implementation will be given in Chapter VI), but the ATN terminology is adopted for ease of exposition. This is not a formal mathematical definition, but it is intended to be clear rather than rigorous.

A recognition grammar is defined to be an unordered set of states. A state is an unordered set of arcs. Notice that "state" has a very specific use here (meaning "set of arcs") and is not exactly synonymous with the general notion of "machine-state" or "computational context", although a "state" does represent the computational state of the ATN portion of the program. Since an augmented transition network is based on a directed graph, the term "node" could have been used, but this would be excessively confusing, since "nodes" are used elsewhere in the model. Notice also that the notion of "subnetwork" does not need to be defined, since the notion of "jumping to" or "activating" a subnetwork uses just the first state of the subnetwork. The interconnected aspect of states in a subnetwork is never used directly by the interpreter when making the jump to its initial state. The reason that a state is not defined as an ordered set of arcs is that no principled way of ordering the arcs has been found; if some use could be made of a "priority" rating for each arc, that would have been included.

An arc consists of an arc-head, an arc-action and a state-specification. An arc-head can be any truth-valued function of one argument, and Section VI.3.2 lists the kinds of tests which were found to be suitable, most of which represent tests on the current word. An arc-action is any operation which returns no result. A state-specification consists of either a special null marker, or a pair comprising a state and a tag. A tag can be one of two indicators, signalling one of two possible options to the interpreter of these rules (see Section IV.5).

The way that these rules are used are described in Section IV.5.

Section IV.3 : Semantic Representation

The semantic system is discussed at greater length in Sections III.10 and VI.3.7, and just an outline is presented here.

There is a set of relations. Each relation has associated with it :

(49)

(a) A set of roles

(b) A set of role-restrictions

(c) An expanded form

(d) An elaborated form

Semantic structures are constructed from relations in the following way. A relation-instance consists of:

(50)

(a) A relation

(b) A list of role-values

(c) A truth-value

A role-restriction is a truth-valued function of one argument. An expanded form is a relation-instance, with certain special role-values; these are needed to indicate how the entries in the expanded form correspond to the entries in the main relation (see Sections III.10 and VI.3.7). An elaborated form is a triple of procedures - one has side-effects but no result, and the other two return truth-values.

A semantic network consists of a set of relation-instances with truth-values either "TRUE" or "FALSE".

A definer consists of a pair comprising a relation-instance and a role, where the role is one of the roles associated with the relation in the relation-instance.

There are a set of sense properties, each with a set of possible values (the "range" of that property) and a sense property name. A sense-property list is a set of pairs, each pair comprising the name of a sense property and a value from the range of that sense property. While being processed or constructed, a semantic structure may have associated with it a sense property list, which indicates how the structure is to be processed.

There are certain operations which can be carried out on semantic structures. These include matching one piece of semantic network against another, setting the truth-value of a relation-instance to "TRUE" or "FALSE", using a definer to produce an item from a network, and inserting an item in a network at a point indicated by a definer. As in the case of structural combining rules (Section IV.1) no particular primitive set of operations has been found.

Section IV.4 : Syntactic Properties and Features

Each grammar contains a set of syntactic features.

Every lexical entry has an unordered set (possibly empty) of features, which is a subset of the full set of syntactic features.

An arc-test (see Section IV.2) may stipulate presence or absence of any set of syntactic features.

The associated arc-set of any feature is the set of arcs which include that feature in their arc-tests. The associated arc-set of a list of features is the union of the associated arc-sets of the members of the list of features.

The associated state of a list of features is the state composed of the arcs in the associated arc-set of the list of features.

Syntactic features do not appear anywhere else in the recognition grammar, and are not used at all by any of the structural combining rules or the semantic network system.

Each grammar contains a set of syntactic properties

Each syntactic property has an associated set of values, known as the range of the property, and a syntactic property name.

Each lexical entry has a syntactic property list, which either is empty or is an unordered set of pairs of the form (<syntactic property name>, <value>), where the <value> comes from the range of the syntactic property associated with the syntactic property name.

Each node (see Section IV.5) may have a syntactic property list, which is defined in the same way.

Syntactic properties may be included in arc-tests, arc-actions or any of the node-manipulating routines. They are not used by the structural combining rules (except via the property inheritance rules, where these exist), or by the semantic network system.

Section IV.5 : Analysis Procedure

The main aim of the MCHINE project has been to investigate how the constructs outlined in the rest of Chapter IV interact in the course of analysing a sentence. Hence this section, which amplifies the basic outlines to show how the devices are used, is more detailed than the others.

There is a device called an analyser which takes as input a string of words, a context, and an initial state (i.e. one of the "states" of the recognition rules). (There is a slight redundancy here, in that the context could be taken to include the other two inputs, since the context represents a global computational environment. However, it is clearer to phrase it this way, so as to emphasise that these two inputs are essential for the analyser, whereas extraction of information from the context may not always occur). The definition of "word" is not relevant here - we will assume an adequate definition can be given at some stage - but each word has an associated lexical entry. A lexical entry consists of a triple - a semantic item, a syntactic property list and a syntactic feature list (see Section IV.4). Any one of these parts of the triple may have a null entry, but no lexical entry may have all three null.

The analyser associates with each initial segment of a string a set (possibly empty) of partial analyses (sometimes called "partial paths"). If the segment is not a proper initial segment (that is, it comprises the whole string), the set is referred to as the set of complete analyses of the string . The last word in the initial segment concerned is referred to as the current word of the analysis.

An analysis (partial or complete) consists of a state, a context, and a weight. (Again, this is a redundant description, in that both the state and the weight could be subsumed by the context, but the presentation here seems clearer). The weight is a non-negative integer. The context includes the value of all registers (interpreter and grammatical) and the state of the main semantic network, and so can represent any structuring or side-effects. The state is referred to as the current state, and is originally set to be the initial state. That is, the list of partial analyses associated with the empty initial segment is set to be the input context, the initial state, and the weight zero.

The analyser makes one scan through the input string, keeping a list of partial analyses which is altered as the analyser processes gradually larger initial segments. For each initial segment (i.e. for each current word), the analyser performs the following procedure on each partial analysis on the list. It finds all the arcs in the current state for which the arc-test yields true if applied to the current word. For each of these arcs, a new partial analysis is created (from the one being processed at the moment), the arc-action is executed in that analysis, and the state-specification is used to select a new current state for the new analysis (except in the case of NEWLEVEL arcs - see below - where the state is selected in a different manner). The new list of partial analyses is made up of all partial analyses produced in that way.

There is one special type of arc, the NEWLEVEL arc, and one special action, POPUP, which must be described in detail. Processing occurs in the analyser at various levels, only one of which is

current (in any given partial analysis) at one time. Different levels have their own workspace, both in terms of interpreter registers and grammatical registers, and therefore the analysis can use a new level to process a constituent independently, using information distinct from that at a previous level. Levels are created when the analyser encounters a NEWLEVEL arc, in the following way. The arc-test in a NEWLEVEL arc is not really a test, but indicates some state in the recognition rules, and the state-specification may be null. When a NEWLEVEL arc is processed, the state given in the (pseudo) arc-test is set as the new current state, and the registers are modified in the following way. If the state-specification is null, then the structure for the old level is attached to its appropriate destination in the surface structure, and all the interpreter registers (apart from the one which keeps track of the grammatical registers) are cleared. If the state-specification is not null, it is pushed on to the continuation stack (see Section IV.6), the arc-action is pushed on to the action stack and all the interpreter registers are pushed down. Thus there are two ways of creating new levels - destructively, with all processing information from the old level being discarded, and by embedding, with all processing information from the old being stored on stacks.

The POPUP action is, in a sense, the reverse of a NEWLEVEL arc. It causes the current level to be left, and processing to return to the last level which has been saved; if none has been saved, special action is taken (see below). POPUP causes the structure built at the current level to be attached to its destination, the grammatical structure-storing registers currently active are tidied up (see

Section IV.8) and the interpreter registers are popped up. Processing continues from the state given by the state-specification popped off the continuation stack.

If no previous level has been saved on stacks, and there are still words to be processed, the analyser performs a restart (see Section III.9). That is, it sets up new structures to build on, and computes a new state to process from, using the associated state of the syntactic feature list of the next word (see Section IV.4). If no such state can be found, the analysis is terminated (i.e. removed from the partial analysis list). On the other hand, if no words are left, the analyser winds up the analysis by checking that all the structural combining rules so far used have been applied (see below).

The semantic structure produced by the analyser is the result of using structural combining rules to combine the semantic entries for the lexical entries of the input words, and applying further combining rules to combine the semantic items thus formed, and so on in a hierarchical fashion. The analysis process consists of working out which lexical items to use as inputs to which combining rules, and which combining rules to use thereafter. The input-output relationships operate hierarchically (see Section III.1, III.2), and the lexical entries are examined in a left-to-right order, so the whole process can be looked on as building a tree from left to right. The analyser keeps track of this gradual building up of rules and arguments by creating structures called nodes. A node contains an structural combining rule, a list of the nodes which will contain its arguments, and the result produced by applying that rule to those arguments. Each node can have syntactic properties (see Section

IV.4) associated with it, to help the analyser connect nodes correctly. When a node containing a rule is created, nodes are built below it to carry the argument values that will subsequently be inserted. These nodes are constructed using information in the rule (e.g. how many arguments it needs) and are called dummy nodes.

A dummy node contains restrictions for some or all of its entries, delimiting what items can later be filled in on them. These restrictions will come from various sources during the analysis, including the semantic rule on the node dominating the dummy node. (In some ways, dummy nodes are like the complex symbols of Chomsky (1965)). Evaluating a rule-node consists of applying the semantic rule at that node to the semantic items on the daughter nodes. A lexical entry is allocated to an input slot in an SCR (structural combining rule) by creating a rule node for the SCR, creating a node which contains the semantic part of the lexical entry (and has the syntactic properties of the lexical entry), and inserting the latter as one of the "argument nodes" of the former. The syntactic properties of lexical entries can be passed up the nodes if the SCR has a property inheritance rule (see Section IV.1). A property inheritance rule for an n-argument SCR will be of the form (L1, ..., Ln), where each Li is a list of syntactic property names. For i= 1 to n, the property values of the ith node will be entered on the syntactic property list of the SCR node (and should not then be altered). Evaluating a SCR-node consists of applying the SCR to the semantic items contained in the argument (daughter) nodes. If the daughter is also a rule-node, a check is made first to see that it has been evaluated, and the property inheritance rules (for the daughter) are applied at that stage.

At the end of the sentence, the topmost node in the SCR tree is evaluated, and hence the whole tree of rules is evaluated (by the recursive system just outlined). The analyser associates with each complete path a list of result-pairs. A result-pair consists of the semantic item from the root of the rule tree, and the corresponding context. (The latter can then encode, for some higher level process, any alterations to the "world" which have resulted from the process of analysing the sentence). The list of result-pairs is chosen by scanning the complete analyses for the sentence, and selecting those which have the lowest weight. (There should be only one result-pair in this list unless the sentence is "ambiguous").

This description does not describe every detail of the analysis process, but it should be sufficient to explain the descriptions in Chapter V (which may themselves elucidate the mechanisms), and covers most of the important points. Chapter VI describes one particular implementation of this kind of analyser, and may give some indication of how some of the vaguer aspects of the outline here could be realised.

Section IV.6 : Registers

There are certain registers which are used in the sentence-analysis process. The interpreter registers (i.e. those which are part of the framework - see Section III.5) are as follows :

Continuation stack : holds the state to be used on terminating the current level (see Section III.7 and IV.5).

Action stack : holds the action to be taken (if any) on leaving the current level.

Register stack : holds the list of grammatical registers in use at current level.

Holding register : holds the structure being built at a lower level, prior to attachment to the main surface structure.

Temporary register : a single slot for temporary workspace.

Shelf : a slot which can hold input words temporarily, on a last-in-first-out basis.

Current node : pointer to surface structure currently being worked on.

Top node : pointer to topmost node of subtree being worked on at this level.

Top nodes : list of subtrees so far built at this level.

Treetop : the overall result of the analysis.

In addition, the grammar-writer can define any register he wishes, designating them as either "structure-holding" (i.e. to be included in the "tidying-up" process - see Section IV.8) or "pointer-holding". There are constructs available for activating registers at the current level, and for performing arbitrary manipulations on them.

The three stack registers above (continuation, action and register) always stack or unstack together, when level-changes occur, so it would be possible to replace them with a single register (a "control stack") which contains items recording all the information ("stack frames"). This would not be a significant change.

Section IV.7 : Conversation Routines

Conversational structure is not the main focus of the MCHINE project, but it was necessary to implement some dialogue system to test out the MCHINE grammar (Chapter VI). The implemented version is based on Power (1974), but does not follow his notation closely. The main points of the system are as follows.

Dialogues are structured into instances of conversation games. A game is simply a procedure, with various properties, that performs certain tasks.

The special properties are all in terms of control structure, and are very simple. When any game is in progress, another game can be initiated in any of three ways - a "nested" call, where the current game may be continued when the new game is finished; an "exit" call, where the current game is immediately terminated before commencing the new game; an "exit-all" call, where all games currently in progress are immediately terminated before commencing the new game.

The tasks performed by the games are not restricted, but they should include the following. A game initiates sentence-processing by providing the appropriate arguments (string, context and state) to the sentence analyser (see Section IV.5), when an input utterance is required. A game also uses the result-pair from the analyser to modify the world model of the hearer, in a way defined by the details of the particular game. The operations carried out by various games will depend on the illocutionary analyses made by the linguist.

Section IV.8 : Guidelines for Analyses

As observed in I.1, linguistic theories generally have a collection of techniques and concepts which are employed in applying the theory. Many of these are not explicitly stated, and linguists may sometimes not be aware that they are making implicit methodological assumptions. This section outlines those aspects pertaining more to the application of the devices outlined in earlier sections of Chapter IV, rather than to the formal properties of those devices. To some extent, the rules-of-thumb here characterise the notion of "ad hoc solution" for a computational grammar.

One major question that has to be considered is where, in the model, to describe particular patterns. In a description which uses several syntactic and semantic mechanisms, it may not be obvious where a given generalisation should be allocated.

Generally, the role of the SCRs (structural combining rules) is to factor out any syntagmatic regularities. If there are several examples X1Y1, X2Y2, ...XnYn, where the relationship between constituent Xi and constituent Yi is the same in each case, this relationship can be most economically represented by putting it in the SCR used to combine the Xi Yi pairs, rather than trying to build this syntagmatic pattern either into the Xis or the Yis. On the other hand, if we have examples X1Zi, ...X1Zn, where there seems to be some semantic similarity between the pairs, and no examples Y1Zi with this semantic property, it is neater to try to capture this pattern in the representation of X1. Some kind of balance must be struck between these two approaches - an SCR which only ever occurs

with one particular value for its first argument is slightly ad hoc; conversely, a whole battery of semantic items with some common property directed towards syntagmatic combination suggests that a generalisation is being missed. (This principle underlies the arguments in Section V.1.3 concerning modifier-head relationships).

Related to this is a need to avoid multiplying lexical items. We do not want to describe different usages of a particular surface word by producing a different lexical item for each use. As far as possible, we should account for different nuances of meaning by having the same item interact in different ways with the context (both linguistic and situational). This last point is an example of a fairly general principle, which might be termed localised semantic description. The idea is that, for ease of semantic computation in the surface structure tree, all decisions are carried out at an independent, local level as far as possible. If we can arrange our semantic description so that each noun phrase, for example, constructs its own semantic representation without much reference to its surroundings, then rules can be written in a more modular fashion. Different uses of a noun phrase would then have to be described by appropriate differences in the SCRs that combine them with other structures, or in the way that they react with other structures, once combined.

If some aspect of a constituent cannot be processed within that constituent, but has to be held until some more global information (either higher up the SCR tree, or in the conversational context) is available, then the way of representing this aspect may change. When the processing happens locally, it may be possible to describe this

property as a function acting on an argument, for example. If the property is uninterpretable locally, its contribution cannot be expressed in a function-application occurring at the local level, and it must be held in some static form which a higher semantic item can react with later, or which can be manipulated by a higher SCR.

Section V.1 gives some examples of situations where a generalisation can be extracted into an SCR. The assumption of localised semantic description underlies several of the analyses in Chapter V, particularly Section V.6. Sections V.2 and V.7 include examples of information which cannot be integrated at a local level, but must be interpreted by a higher rule.

As stated earlier (I.3 and I.6), computational grammar assumes that sentences or dialogues which sound "odd" must contain (or result from) some anomalous structure or process. This criterion has not generally been used in artificial intelligence language programs. Recognition rules are sometimes given which will accept an endless stream of auxiliary verbs uncritically, for example. The justification is that these rules will work correctly on correct input, and their behaviour on ill-formed strings is completely irrelevant. However, if we adhere to the principle of relating oddity and anomaly (subject at least to a partial specification of what constitutes anomaly), then grammars must be more carefully constructed.

Since the focus of this investigation is on the sentence-analysis process, the guidelines for writing the recognition rules are important. As discussed in Section III.8, grammars should not be allowed unlimited power to revoke all decisions once made.

Unlimited look-ahead should be avoided, for the following reasons. The computational grammar model uses strict left-to-right processing, taking in one word at every stage. If the analyser requires to test a word somewhere "ahead" in the input, it should explicitly store the intervening items somewhere, and process them later. If this can be managed, then the look-ahead is permissible - the analyser is "remembering" the string of words before starting to process them. However, if the look-ahead is a hidden way of parsing a later item before the current word, it is obscuring the true flow of decisions, and should be avoided. (Martin Kay (at the Workshop on Theoretical Issues in Natural Language Processing, Cambridge, Mass., June 1975) commented that look-ahead could often be backtracking in disguise; look-ahead, he suggested, moved the sentence-pointer forward, but left the label "YOU ARE HERE" behind). The exception to this is the "one-word look-ahead" discussed in Section III.8; this is not so much look-ahead as delaying, until the next word is taken in, all the actions that might have to be performed at the current point. Since the actions will be performed (or abandoned) immediately, there is no need to store anything.

As commented in Section III.5, it may be possible to use unnamed, general purpose registers for storage during sentence-analysis ("work registers"). There is no clear criterion for when these can or should be used, although it seems appropriate to use them when the item in question has not been analysed, and so cannot be allocated to a very specifically named register (e.g. if indulging in explicit look-ahead in the way described above). In the interests of seeing how far this concept can be taken, two work registers have been included among the interpreter registers -

TEMPORARY and SHELF.

It seems plausible to suggest that a structure stored in a grammatical register must eventually be used (either by being incorporated into a larger structure, or by being discarded after some information has been extracted from it). Let us assume that the interpreter can distinguish pointer-holding registers from structure-holding registers. An intuitively attractive principle is that all structures built at a particular level should be explicitly used before leaving that level. That is, all structures which have been temporarily stored in structure-holding registers should be removed from these registers and either attached to some larger structure, or else explicitly discarded (perhaps because all their information has been recorded in some way, as with an auxiliary verb). This is not to say that structures cannot be left lying in a register while a lower constituent is processed; it is just that such items should not be lost when the interpreter exits to a higher level and restores all the stacks. The grammar-writer should discipline himself so that any operations which use items from registers to build structure (as opposed to merely examining them) simultaneously remove those items, leaving the registers empty. Then the interpreter, before leaving a given level, can check that all the structure-holding stacks that it is about to pop are empty. If any are not, it should try to use up the left-over items before leaving that level. This "using-up" process will have to be fairly general and based on formal properties of the current surface structure, since it will have to be programmed into the interpreter, not the grammar. A first approximation might be to attempt to attach the spare structure on the bottommost empty node (cf. Kimball (1974)).

Uses of this tidying up principle will be discussed in Chapter V).

As defined in Section III.5, a "flag" is a register with a small fixed range of values, used for recording the presence or absence of some condition . These can be very useful devices, allowing the same analysis network to be used for two similar phenomena, with minor differences recorded in flags (Bobrow and Fraser (1969)). However, they should not be allowed to obscure the real relationships between various conditions and operations in sentence-analysis. For example, the fact that a clause has a passive verb form conveys information about the way that deep semantic roles (or "cases" - see Section V.8) will be arranged in that clause. Using a two-valued flag at various stages during structure-building is one way of handling this, but it may not be the most transparent. If possible, the consequences of a particular construction occurring in a sentence should be recorded or carried out as directly as possible. This complaint about arbitrary flags is analogous to the criticism of using arbitrary syntactic features to record (temporarily) a particular fact which is later decomposed into heterogeneous consequences (Section III.11).

As discussed in Section III.6, an analyser which handles decisions by exhaustive exploration is less interesting than one which manages to postpone decisions until sufficient information is available to resolve them. Ideally, an analyser which only ever maintained one (correct) analysis throughout the parsing process would be extremely elegant. It therefore seems desirable to write grammars in such a way that the analyser has to "branch" as rarely as possible.

So far, all the points mentioned in this subsection have been fairly peripheral to computational grammar. Most of them are not hard rules, but rough guidelines. Some of them suggest ways of "keeping the grammar clean", others are suggestions for ideas to try out. The points which follow may be slightly more important, since they serve to clarify the notion of a "structural combining rule".

In transformational grammar (at least in the "Aspects" model), most phenomena were handled by one mechanism - the transformation (see Section II.1 for a discussion of that approach). In computational grammar there are various ways one could handle these constructions. Descriptions previously formulated in terms of transformations can largely be replaced by descriptions employing two kinds of device - using structure-holding registers, and defining several different SCRs.

For example, there were good syntactic arguments for the transformation of "Subject-Verb Inversion" (Chomsky (1957), Burt (1972)), whereby the auxiliary verb at the start of a "yes-no" question was described as part of the main verb phrase of the sentence. When writing a recognition grammar, this notion again arises. It is rather clumsy to have to specify all the possible auxiliaries at two points in the grammar and use elaborate interconnections to relate the auxiliary at the start of a question to the following verb phrase. It is much simpler to store the question-auxiliary in a register (without recording all the information from it, but noting that the sentence is interrogative), and extract it later when commencing the verb phrase. This seems to be a very direct expression of the Chomskyan transformation.

Although the phenomena which supported this strategy are the same as those used in a transformational argument, the justification comes from the simplification of the parsing rules with no allied complication of any other part of the grammar.

On the other hand, the different surface configurations of deep semantic roles (Section V.8) are not amenable to such obvious re-ordering manipulations. Transformations such as "Dative Movement" and "Passive" were intended to shuffle subjects, objects and indirect objects into a canonical structure, resulting in a simpler statement of cooccurrence relations. However, cooccurrence relations are based on semantic properties (Lakoff and Ross (1967), McCawley (1968)) rather than on surface syntactic form. Hence such re-ordering would not simplify the recognition rules, since these handle the structural options available at surface level; this is not where the generalisation is. It might be, however, that the set of SCRs could be simplified using re-ordering operations; for example, we might need only one two-object rule instead of two separate rules if we replace Dative Movement by a register manipulation at the surface. As will be discussed in Section V.8, there are reasons for including separate semantic rules anyway. Given this, there is no motivation for attempting to include a re-ordering operation.

Generally, it is the overall simplicity of the grammar that decides whether to multiply the set of SCRs or to re-order constituents directly. Re-ordering is usually appropriate where the generalisation is describable in surface syntactic terms, since then (and only then) it should simplify the recognition rules. Where the pattern is semantic, the decision will depend on whether re-ordering

would greatly complicate the recognition rules and whether the SCRs can be significantly reduced or simplified.

This brings up another point concerning the ordering of arguments in SCRs. The SCRs are functions whose arguments must be ordered so as to distinguish them, as with any mathematical function. Complications can arise when the same notational device (namely, left-to-right order) is used to represent this arbitrary ordering as is used to represent temporal ordering in the words of a sentence (see, for example, the chaotic arguments of McCawley (1970)). The crucial point seems to be as follows. Since the SCRs are used to process a temporally-ordered sequence, there is an ordering imposed on the SCR argument-slots, other than just the arbitrary order used to distinguish them - namely, the ordering determined by the order of processing of the individual arguments. "First argument" in an SCR effectively means the "first to be allocated to a slot in the SCR". (This temporal ordering can then be used to distinguish the argument places, since any ordering achieves that). A definition like this restricts SCR surface trees to being built in a particular order (notationally, left-to-right), with no "gaps" being left to be filled later. (In fact, the MCHINE program does use strict left-to-right building, as it makes it much easier to keep track of the structure being built). So far no examples have been found which are difficult to handle in this way. Therefore we can adopt the principle that the ordering of SCR argument-places is significantly related to surface ordering, and that arguments must be inserted in left-to-right order. (See Isard (1974) for some related comments on trees and processing order).

CHAPTER V

SOME AREAS OF ENGLISH GRAMMAR

Section V.0 : Preamble

This section uses the framework of Chapter IV to describe certain aspects of English grammar. The descriptions are phrased informally, with references to the concepts and devices of Chapter IV, since this seems the clearest expository device. Some of the analyses have been implemented, or partially implemented, in the MCHINE program (Chapter VI), but others have to be assessed wholly on whatever merits of generality, logic, elegance and plausibility that they may have. Ideally, a computational model of language should suggest both how to describe language and how to program the description. Chapter VI illustrates an implemented version of the framework, but Chapter V demonstrates how the ideas can be used away from the machine room.

The areas of English examined may seem slightly mundane, and it might be thought that a new framework should test itself on tougher ground. Unfortunately, if the grammar, or a major part of it, is to be implemented on a computer, then there are certain basic areas (e.g. noun phrases, verb phrases, auxiliary verbs) that have to be covered if the program is to function at all. If working with a new framework, there are no existing analyses to be relied upon, since it is not clear (starting from scratch) how even these prosaic regions will look from the new vantage point. Section V.2, for example, supplies the background to the implemented grammar, but does not reach any dramatic conclusions.

The linguistic descriptions are necessarily rather over-simplified, and this is also a consequence of having to develop a new model and implement it at the same time.

Section V.1 shows how structural combining rules (SCRs) provide an appropriate way of describing certain regularities which cannot be properly described simply by allotting words to syntactic categories.

Section V.2 outlines the justification for the way that auxiliary verbs are handled in the MCHINE grammar.

Section V.3 illustrates the need for syntactic properties (in the sense of Chapter IV) and property inheritance rules.

Section V.4 gives a description of relative clauses which demonstrates several facets of computational grammar, particularly the use of registers.

Section V.5 examines how the processing devices of computational grammar can be used to describe "perceptually complex" sentences.

Section V.6 uses "definers" and "sense properties" to describe the processing of referring expressions.

Section V.7 gives an extremely detailed examination of the English tense system, showing how semantic networks and structural combining rules can capture various regularities.

Section V.8 outlines a description of verbs and "cases" which is simple and useful.

Section V.1 : The Internal Structure of Noun Phrases

V.1.1 Possessives and Determiners

As observed in Section III.11, determiners ("the", "that, etc.) are distributionally related to possessives ("my", "Fred's", etc.). If we could arrange our semantic representations for referring expressions so that the combining process for <determiner> + <head noun> was the same as the combining process for <possessive> + < head noun>, then one SCR will do for both. This is probably not possible, for the following reasons. Determiners form a closed class, containing what are traditionally known as "grammatical" formatives (see Lyons (1968)). Their function is to provide more detailed information about the semantic processing of the nominal provided by the head noun, regarding its definiteness, specificity, etc; they do not provide substantive semantic structure. Possessives, on the other hand, are an open class, and arbitrarily complex possessives can be built from noun phrases. These possessives contribute a substantial part of the meaning of the referring expressions, and supply a whole semantic structure which is to be combined in some way with the head nominal. The possessive construction is recursive (as will be discussed in Section V.1.2 below) and structures can be built up within structures, whereas the determiner construction is not itself recursive. Two different SCRs are needed, say "SCR-Possessive" and "SCR-Determiner".

Nevertheless, the distributional generalisation can still be captured in the recognition grammar, where the two categories (with features POSS and DET) appear as parallel options at the appropriate point of the ATN. A grammar which puts both classes of item into one

category for all purposes would force the two levels of processing to be tied together. Here, we have managed to separate the level of SCRs from the level of ATNs.

V.1.2 Restrictive and Non-Restrictive Adjectives

Many adjectives have two uses as modifiers in a noun phrase, usually referred to as "restrictive" and "non-restrictive" (or "defining" and "non-defining" - Fowler and Fowler (1906)). Informally, the restrictive use is where the meaning of an adjective (usually denoting some property) is used to delimit the class of items denoted by a head noun, so that the combination of (adjective + noun) denotes some narrower class of objects, as in (51)(a) and (b).

(51)

(a) Pick up the blue block

(b) The male employees should read this copy of the Sex Discrimination Act, and the female employees should read that one.

The non-restrictive use is where the class denoted by the noun phrase is not narrowed down by the meaning of the adjective, but the adjective conveys some property that the speaker wishes to attribute to that class. These are hard to illustrate, since most adjective + noun combinations can be understood restrictively, and there are few examples where only the non-restrictive interpretation is possible.

(52)

(a) These stupid arguments do not concern me.

(b) Your fickle friends have deserted you.

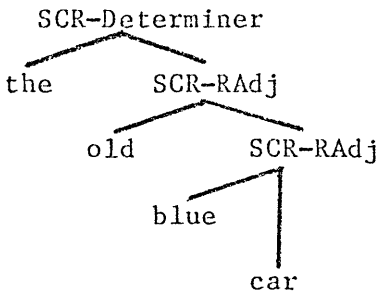
(c) His wealthy parents have bought him a golf club.

Since a wide range of adjectives can have either use, it seems better to describe these as two separate constructions, rather than having separate lexical entries for the two uses. (The latter could be done by having a feature [ATTRIBUTIVE] and a lexical redundacy rule stating that an adjective with that feature had two forms - [RESTRICTIVE] and [NONRESTRICTIVE]. Possible, but inelegant). SCRs provide an obvious mechanism for this, since there can be two SCRs for combining adjectives and nouns. One of these performs some kind of property-intersection on the sense to yield a new class, and the other constructs an assertion which is to form part of the message conveyed by the speaker.

The non-restrictive SCR (call it SCR-NRAdj) has some interesting consequences for building surface structure.

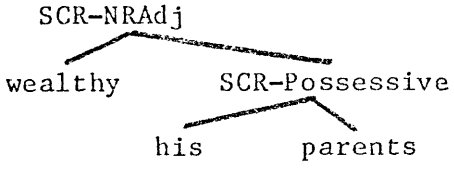
Let us assume that the head noun generally provides some kind of predicate, denoting a class of items, and that a restrictive modifier can be combined with it to form a new, narrower class-predicate. This process can be applied repeatedly, so that the restrictive rule (call it SCR-RAdj) can take part in a right-branching structure like (53).

(53)



In this structure there are no discontinuous constituents - all the surface items which form a subtree are adjacent, and the restrictive adjective is always to the left of the item(s) it has to combine with. The non-restrictive case is not so simple. The non-restrictive adjectives acts not on a class-predicate to form a narrower class predicate, but on the final result of the SCR-Determiner, to form an assertion. In (52)(b) "fickle" makes an assertion about "your friends", and in (52)(c), "wealthy" makes an assertion about "his parents". This seems to call for surface-tree of the form (54).

(54)



(Remember, as outlined in Section III.1, III.2, that the surface tree directly represents how the SCRs act on their arguments. There is no question of temporarily pasting together some "syntactic structure", with different dominance relations, and letting the SCRs sort out their arguments later).

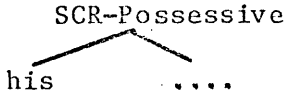
Here there is a discontinuous constituent ("his...parents"). The non-restrictive adjective seems to act on the meaning of the (possessive + noun) and the subtree to the right of the adjective may be arbitrarily large. There are at least two ways of handling this.

Firstly, the analyser could hold the possessive (or determiner - see V.1.1) in a register until it had ascertained what adjectives were present in the noun phrase. Once all the non-restrictive adjectives had been built into a right-branching structure (there may be several adjectives, e.g. "His kind, loving, wealthy parents..."), the possessive could be attached and the rest of the phrase analysed.

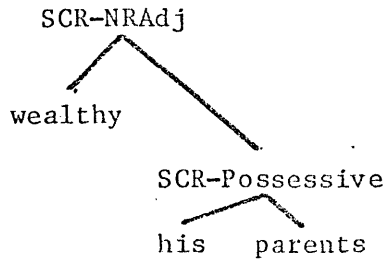
Alternatively, the analyser could build the possessive (plus a node with SCR-Possessive) on to the surface tree at once. On encountering non-restrictive adjectives, it could build branches on to the <u>top</u> of the subtree for the noun phrase. The stages would be as in (55).

(55)

(a)



(b)



The latter approach requires the subtree for the noun phrase to be built as a separate item, before being attached to higher nodes (so that nodes can be interposed above the root of the subtree). It is quite feasible to assume that each processing level (see Section III.7) has a separate register, an interpreter stack called TOPNODE, which holds the root of the subtree being built at that level. (Both the MCHINE and SHRDLU programs do that). The former approach seems equally viable, but has not been tested here.

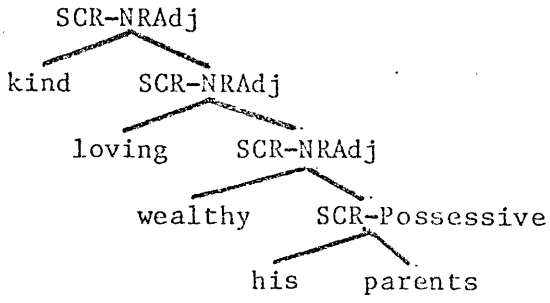
This structural aspect of SCR-NRAdj is related to the recursive aspect of SCR-Possessive, mentioned in V.1.1. Possessives may be arbitrarily large phrases (e.g. "the tired old man's hat"), attached as constituents of other noun phrases. Let us assume that such phrasal possessives are processed initially as noun phrases, then (on encountering the "'s" at the end) attached under a SCR-Possessive node, with processing continuing on the main noun phrase without initiating a new level (see Section V.5 below for further details of the processing levels involved). While processing the phrasal possessive, the analyser may have no indication that this is not the main noun phrase, and so will be building it as the subtree for the current level. On creating the SCR-Possessive node, some re-organisation is required, since the SCR-Possessive node becomes the new root of the main noun phrase, with the structure that formed the previous subtree as left daughter. Here is another situation where it is useful to have the TOPNODE register for storing the root of the current subtree. Notice that these two manipulations of TOPNODE would not interfere with each other, although phrasal possessives and non-restrictive adjectives may occur in the same noun phrase. During the processing of the phrasal possessive, TOPNODE

contains the root of the possessive, and so any non-restrictive adjectives encountered (e.g. "[that careless man]'s motorbike") are inserted correctly above the possessive. Once the phrasal possessive has been completed, and the subtree restructured to allow the rest of the noun phrase to be processed, TOPNODE contains the root of the main noun phrase, and any non-restrictive adjectives will be attached correctly above this phrase, since they always follow the possessive. Even if a recursive left-branching structure occurs (as in the examples in Section V.5 below), this re-structuring will occur in the same way at each embedding. The two potentially recursive constructions (possessive and adjective) will not interfere with each other, despite the fact that both re-allocate the contents of TOPNODE.

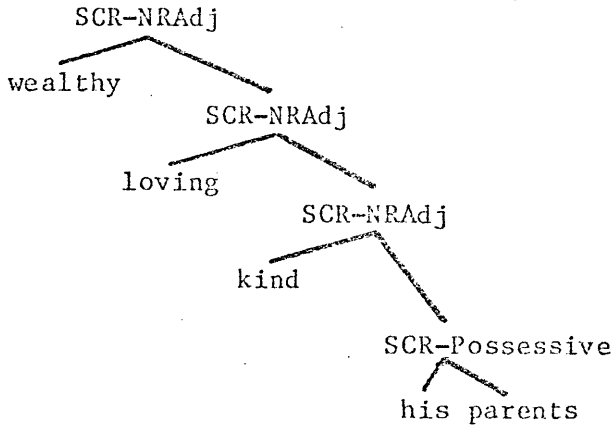
It was observed above that several non-restrictive adjectives may occur in a single noun phrase. Careful examination of the two approaches outlined above (holding the possessive in a register, or building a tree which is then re-rooted) will show that they will result in slightly different trees for such sentences. Holding the possessive in a register will retain the surface order of the adjectives (see (56)(a)), whereas re-rooting re-orders the adjectives (see (b)).

(56)

(a)



(b)



This does not give a way of choosing between the two methods, since both structures are suitable semantically, for the following reason. The inputs to SCR-NRAdj should be an adjective-meaning (say, a property P) and a noun-phrase meaning (say, a set S). The important question is - what is the output of SCR-NRAdj ? The effect of the rule is, informally, to create an assertion that the set S described by the noun-phrase meaning has property P. However, this cannot be the output of the rule, since that would provide the wrong input for the SCRs higher up the tree. The noun phrase subtree as a whole should produce a set description S′, since the constituent which includes the NP will use that set-description in some way; the containing constituent does not require an assertion to operate on.

The best way round this is to have SCR-NRAdj pass up its second argument S unaltered as its output in the rule-tree, and do its work solely by placing the assertion (i.e. that P is true of S) somewhere suitable (e.g. in the "world model" of the speaker). This allows SCR-NRAdj to apply to its own output, as is necessary in (56), since its second argument is a set-description and its output is a set-description. Since the same set S is being passed up trees like (56)(a) and (b) unaltered, with various assertions being made about it on the way, the order of application of the different invocations of SCR-NRAdj is immaterial.

One advantage of separating out the non-restrictive semantic relationship into an SCR (instead of trying to build it into lexical entries), is that certain modifiers other than traditional adjectives sometimes occur with a non-restrictive type of relationship to the noun, e.g. (57).

(57) The many admirers of Shostakovitch will be saddened by this performance.

The word "many" occurs here after the determiner, and seems to express an assertion that there are many admirers of Shostakovitch (it is certainly not restrictively distinguishing the "many admirers" from the "few admirers"). Such usage might be describable by using SCR-NRAdj to combine the meaning of "many" with the meaning of the rest of the phrase.

This discussion has ignored the major question of how the analyser is to decide, while processing a sentence, whether a particular adjective is being used restrictively or non-restrictively. The decision seems to depend on factors which are impossible to formalise within the limited grammatical framework here - intonation, context of utterance, hearer's view of speaker's beliefs, etc., may all contribute. To test that the two surface structures described are practicable, the MCHINE program has been designed to allow either, but the choice of which to build is fudged by having adjectives categorised into two disjoint subsets, with features NRA and RA to distinguish them.

V.1.3 Adjectives and Classifiers

So far the traditional term "adjective" has been used quite freely, and the adjectives discussed in V.1.2 have been regarded as expressing "properties" that may be predicated of sets. These over-simplifications aided the exposition (without affecting the relevant arguments), but they will now be examined in greater detail.

Bolinger (1967) discusses the wide variety of English adjectives, showing that it is not reasonable to describe all adjectives as denoting "properties". Examples like (58) are relevant.

(58)

(a) chief problem

(b) alleged thief

(c) former president

(d) possible problem

Montague (1970b) points out that describing the adjective-noun relationship as the intersection of properties (cf. Winograd (1972)) is valid only for a certain subclass of adjectives, and there is no reason to regard this as the adjective-noun relationship. He suggests that there should be several adjective-noun combining functions, and this proposal is easily re-phrased in terms of computational grammar as the need for different SCRs. If we could make the operations of the SCRs sophisticated enough, that might even allow for constructions like those in (59), where the way that the adjective modifies the noun depends in quite subtle ways on both the adjective and the noun.

(59)

(a) big flea

(b) small elephant

(c) good cook

(d) bad athlete

Further investigation shows that it is not very plausible to regard "adjective" as a clear semantic category (although its pre-nominal usage gives grounds for a syntactic feature [ADJ]). Consider a phrase like (60).

(60) brass candlestick

Traditional grammar has two ways of describing this combination of words. Either it is an adjective-noun pair, like the rather heterogeneous examples in (58) and (59), or it is a noun-noun pair (sometimes known as a classifier-noun pair), like the examples in (61).

(61)

(a) donkey jacket

(b) soup spoon

(c) boiler suit

(d) monkey wrench

(e) gas stove

It is hard to define what criteria are used for sorting an example like (60) into either list. The main criterion seems to be that adjectives can be used to modify a wide class of nouns, whereas classifiers (i.e. nouns used as pre-nominal modifiers) are idiosyncratic. This generalisation is not strong enough to provide any real two-way classification, since some of the classifiers in the phrases in (61) can modify various nouns:

(62)

(a) soup plate

(b) gas fire

(c) soup kitchen

(d) gas mask

There does, nevertheless, appear to be a distinction between systematic modifier-noun pairs and idiosyncratic modifier-noun pairs, but it does not correspond to traditional adjective/classifier boundaries.

The relationship between the meanings of words in (62)(a) is (intuitively) very similar to the relationship between the meanings of the words in (61)(b). The same relationship seems also to occur in pairs like "fish knife", "dessert spoon", where the name of a kind of food is used to classify an eating implement. (62)(c), on the other hand, is different. Similarly, the relationship between the word-meanings in (62)(b) is like the relationship between the word-meanings in (61)(e), and this relationship also appears in phrases like "coal fire", "oil heater", etc., where a fuel is used to classify a fuel-using device. (62)(d) seems to be different. We could perhaps say that there are certain systematic modifier-noun relationships, which manifest themselves in phrases consisting of pairs "X+Y" where both X and Y may vary over more than one item. There are also certain idiosyncratic modifier-noun relationships, which manifest themselves in phrases consisting of pairs "X+Y", where X and Y do not vary (i.e. these are "one-off" items). (61)(a), (c), (d) and (62) (c), (d) seem to be examples of these. Thus it appears that, while the traditional distinction between "systematic" and

"idiosyncratic" modifier noun pairs may well exist, it certainly does not correspond to the usual classification of "adjective" versus "classifier".

It is worth digressing here to point out a simple linguistic test which distinguishes these two kinds of combination. In contrasting noun phrases with different modifiers, the head noun may be replaced by "one" under certain circumstances, as in (63).

(63) This is a red ball and that is a blue one.

In (63) the modifier-noun pairs are traditional property-denoting adjectives, which are highly systematic in the way they modify nouns. If the modifier-noun pairs are totally idiosyncratic, then the "one"-pronominalisation results in oddity.

(64) ??? This is a monkey wrench and that is a pipe one.

This may be because the contrast which is being expressed includes the particular modifier-noun relationship involved, and if the two phrases have different internal relationships, the contrast is difficult:

(65)

(a) This is a wine bottle and that is a whisky bottle.

(b) This is a red bottle and that is a blue one.

(c) ?? This is a wine bottle and that is a red one.

Intuitively, the contrast is being made between different values for a particular property of the items, and so the same property must be referred to in both phrases. Systematic modifier-head relationships may be those where the modifier specifies some value for a particular aspect of the head, whereas idiosyncratic items are more like arbitrary labels. (There are borderline examples like "ulterior motive", where, intuitively, the modifier expresses some aspect of the head, but where there are no related phrases "ulterior X" showing the same modification - ??"This is a worthy motive and that is an ulterior one.")

This discussion has been somewhat vague, with frequent appeals to the reader's intuition. Hopefully it has been sufficient to make two points.

Firstly, although there may be syntactic distributional grounds for grouping pre-nominal modifiers into adjectives and nouns, there is no useful semantic distinction to be made, since there are systematic uses of a wide variety of modifiers. Secondly, the aspect of modifier-head combinations which is semantically relevant is the way that the modifier modifies the meaning of the head; it may be possible to describe this by having many different SCRs for the vast number of possible relationships.

The latter proposal has grave repercussions. Leaving aside the prickly question of whether we want to have a multitude of relationships like "uses as fuel", "is used to eat", etc., built into linguistic rules, there is still the problem of how an analyser is to

choose the right SCR for a particular pair. It is somewhat implausible to have the analyser treat a pre-nominal modifier as n-ways ambiguous (where there are n SCRs which could take it as first argument), so the analyser will have to wait until the head of the phrase is found, and then select a modifier-head SCR which will accept as inputs this modifier and head.

This may sound similar to the process described by Quillian (1969), where a semantic program examines the semantic structures of the modifier and head to work out the relationship between the two, but there is a slight difference. Here we are assuming that certain commonly-occurring semantic relationships are stored in standard modifier-head SCRs (instead of having to be computed each time). Thus, under normal circumstances, only the list of SCRs is considered, rather than all possible relationships. The Quillian process (which starts from scratch, with no standard set of relationships) would be useful for computing possible meanings of a hitherto unencountered pair, but even then it might be wrong if the newly encountered example is in fact idiosyncratic; consider what a productive process like that would have done with "pyramid salesman" or "battery hen" on a first attempt. While a heuristic process like Quillian's would generate guesses for unknown phrases, the suggestion here is to keep a set of standard relationships for systematic purposes, and enter idiosyncratic phrases directly in the lexicon as compound nouns. This also produces a problem for a analyser, since as well as searching the SCR list, it will have to check whether the lexicon has a compound entry for the two items it is examining.

The process of searching a list of SCRs for one which "matches" a pair of arguments is in fact required elsewhere in the grammar (see Sections III.9, V.4) so it is not necessarily a weak point of this description that it needs such a search process, but the search routine is not a well developed concept at present, and cannot be relied upon.

None of the suggestions in V.1.3 have been implemented in the MCHINE grammar, so this discussion can be regarded only as speculation on the possible uses of SCRs.

Section V.2 : Auxiliary Verbs

V.2.1 Avoiding Branching

The part of the ATN in the MCHINE grammar that processes the auxiliary verbs is deterministic, subject to the "one-word-lookahead" provision of Section III.8. That is, no wrong analysis is ever pursued for more than one word, and no markers or structures are altered once they have been set. The network appears to be slightly more complicated than other networks for English auxiliaries that have appeared in the literature, but this is a consequence of three constraints within which the rules were written.

Firstly, the network should not backtrack or follow up several wrong paths. Secondly, detailed information regarding the various auxiliaries should be extracted (see below). Thirdly, the grammar should accept exactly the sequences of auxiliaries that occur in English, and no others (see Sections I.2 and I.3 for justification of this).

The general word-storing register SHELF is used on several occasions to postpone decisions for one word. This may seem unnecessary, if one-word branches are being allowed anyway, but these were cases where it was possible to predict that the information would be forthcoming immediately, and where the options were a small fixed number (here, usually two options). The SHELF register is also used to hold the auxiliary verb which occurs at the start of a question while the subject noun phrase is analysed. This allows the same part of the auxiliary network to process the first auxiliary verb, whether the verb occurs before or after the subject (cf.

Chomsky (1965)).

V.2.2 The Information Conveyed

The information contained in the auxiliary verbs is recorded in various ways. Much of the information cannot be interpreted within the sentence grammar, but must merely be marked on to the semantic representation for the clause, so that some higher interpreter can act on it. The tense, modality, and aspect of a sentence are interpretable only within some context (e.g. a conversation) and so their meanings must be defined outside the sentence (see Section V.7 below). Some of these properties (e.g. progressive, perfective) seem to belong to the verb phrase (since there are perfective and/or progressive verb phrases which do not form surface clauses (see (66) below)), and some are associated with the clause (e.g. tense, modality). The information extracted from the auxiliary sequence is as follows:

Perfect aspect : a two-valued marker is set on the semantic item for the verb phrase.

Progressive aspect : a two-valued marker is set on the semantic item for the verb phrase.

Verb-negation (see V.2.3) : a two-valued marker is set on the semantic item for the verb phrase.

Clause-negation (see V.2.3) : A two-valued marker is set on the semantic item for the clause.

Tense : a two-valued marker is set on the semantic item for the clause.

Illocution : SAY, ASK or ORDER is marked on the representation for the clause.

Modality : the corresponding modal (CAN, WILL, etc) is marked on the semantic item for the clause.

Voice : a restriction is set on the surface node for the verb phrase, so that the SCR used must be of the correct variety (active or passive). See Section V.8 for more details.

V.2.3 Negation

The distinction between verb-negation and clause-negation may seem unusual. This is intended to capture the following patterns.

Firstly, verb phrases (in the infinitival or gerundive form) often contain a negative element separate from the negation (or otherwise) of the containing clause.

(66)

(a) Not to go to the party would be impolite.

(b) Not having been there, he did not want to comment.

Secondly, a negative element in a sentence can sometimes be interpreted in two different ways - compare (67)(a) and (b).

(67)

(a)

A : How can I find time to write this essay ?

B: You can not go to the party tonight, or you can not go skiing at the weekend.

(b)

A: I broke my ankle on Saturday.

B: You cannot go to the party tonight, and you cannot go skiing at the weekend.

In (a), the "not" seems to state that a negative course of action is possible; in (b), the "not" seems to state that a particular course of action is not possible. (This distinction is more easily expressed in the notation of modal logic). One possible way to express this is to associate the negation with the verb phrase in (a) and with the clause as a whole in (b). (The modal "can" is also associated with the clause).

Thirdly, sentences can, under certain circumstances, contain two negative elements, (corresponding to the two usages just described).

(68) ? You can't not go to the party - they will be expecting you.

Such sentences sound more acceptable if the first negative element is conflated with the first auxiliary verb to produce an "n't" form, and

even then a slightly unusual intonation is necessary if the sentence is not to sound odd.

The same patterns regarding double negation hold with other auxiliary verbs, but not all of them provide clearly distinct semantic interpretations for the two types of negation in the way that "can" does. This may be due to the fact that, for other modals, there is no imaginable situation in which one interpretation is true but the other is not. If we follow the patterns of (67), the sentences in (69) should be approximately paraphrased by those in (70) - but what situation would distinguish these cases ?

(69)

(a) He mustn't go to the party.

(b) He must not go to the party.

(70)

(a) It must not be the case that he goes to the party.

(b) It must be the case that he does not go to the party.

Unfortunately, this phenomenon has not been analysed in depth here, as it would need a full description of the semantics of modal verbs. (See Isard (1974) for a partial treatment of modal verbs in a computational framework). Jackendoff (1973) discusses the use of "VP" and "S" negation in connection with certain quantifiers, but his description is phrased in transformational terms.

The network in the MCHINE grammar operates as follows. An "n't" form indicates clause-negation; a "not" following an "n't" form indicates verb-negation; a single "not" is taken as either clause-negation or verb-negation depending on a variable which the user can set (this is because, in a full model, these decisions would not use the kind of grammatical information represented in the current model, but would be based on intonational or contextual information); two "not" elements in succession cause a failure.

V.2.4 "Do"

It has been suggested for some time that the verb "do" in sentences like (71) is an unusual item.

(71)

(a) Do you want a cup of tea ?

(b) You do not like asparagus.

(c) Do not enter that room.

It can be regarded as a verb, since it carries an inflection for tense and "number" agreement. It can be regarded as an auxiliary verb, because (like the aspect and modal verbs) it appears before the subject in questions, and precedes the negative marker if there is one. However, it has the oddity of appearing only when there are no other auxiliary verbs, and not appearing even then in unemphasised, non-negative declarative sentences. This prompted the introduction of the transformation often known as "do"-Support (Chomsky (1957), Burt (1972)). Trying to class this "do" as either a modal or an

aspect verb leads to difficulties, so it seems better to regard it as a separate kind of auxiliary altogether. From the point of view of sentence-processing, the information that can be extracted from the auxiliary "do" is considerably different from that extracted from the other auxiliaries, and is largely based on what its paradigmatic (or distributional) behaviour tells the analyser about the absence of other auxiliaries.

Talking of "aspect" verbs is misleading, since, for all practical purposes, this is a pseudo-class containing only two members ("have" and "be"), both of which may occur in a sentence, and which convey totally different information. "Have", "be" and "do" are separate, one-off auxiliaries which can be distinguished by any consistent feature-marking that the analyser may need. The MCHINE grammar has them marked (redundantly) with the features [ASPECT], [COP] and [DO1] respectively.

Section V.3 : Number Agreement

In Section I.2, it was stated that, since language comprehension was a primarily semantic goal, syntactic mechanisms should be introduced only where necessary to achieve this goal; if some aspect of English could be fully described in semantic terms, then there was no need to postulate any additional syntactic structure. This may seem a clear aim initially, but there are areas of English where it is impossible to give wholly semantic descriptions, and additional devices have to be introduced, despite the methodology of avoiding separate syntax. Sometimes the syntax and semantics are related in such a way that redundacy occurs, but neither can be completely eliminated. A good example of this occurs in the English system of number agreement. (Katz (1972, pp.378ff) gives a good discussion, within a different framework, of this area of grammar, and the arguments here overlap with his to some extent).

V.3.1 Subject-Verb Agreement

Subject-verb agreement is often referred to as "number-agreement", and is described in terms of "singular" and "plural". This is misleading, since it gives the impression that the agreement system is based on plurality of the set of things denoted by the subject (i.e. that it is a semantic phenomenon). The two-way classification into "person" and "number" for verb forms (e.g."3rd person singular") may be helpful for relating English to other languages, but it does not reflect the agreement patterns which actually occur in English.

Consider regular verbs. Most of these have two present-tense forms (e.g."run", "runs") which we can refer to as PRESP (mnemonic for traditional Present Plural) and ES (mnemonic from the inflectional ending) forms. The PRESP form is compatible with subjects "I", "you", "we", "they", or any "plural" noun phrase. That is, it covers what were traditionally known as 1st and 2nd person singular, plus all plural forms. The ES form is compatible with what are traditionally called 3rd person singular subjects - "he", "she", etc. Regular past tense forms (or "remote" tense forms, to use the terminology of Section V.7), are generally compatible with any subject at all, as are the present and past forms of modal verbs (which are "defective" in traditional terminology).

An exception to these agreement patterns is the verb "be". There are four classes of subject for agreement with inflected forms of "be", as follows.

Compatible with "am" : "I"

Compatible with "was" : "I", "he", "she", etc.

Compatible with "were", "are" : "you", "we", "they", etc.

Compatible with "is" : "he", "she", etc.

Although "be" is a solitary case, and so can be treated exceptionally, we still need agreement-classes that are capable of describing this pattern as well as the regular one. The most economical set seems to be :

Agreement Class 1 : "I"

Agreement Class 2 : "you", "we", "they", etc.

Agreement Class 3 : "he", "she", etc.

Other classes can then be formed by taking the union of these minimal classes. Thus the four classes for "be" outlined above are 1, 1+3, 2 and 3 respectively. Regular verbs will generally require 1+2 and 3 to describe the present tense, and 1+2+3 to describe the remote tense. Using unions is of practical use for the way that agreement information can be used during sentence-analysis, as will be described below.

The difference between Class 2 and Class 3 might seem to correspond to a semantic difference in the subjects involved, as in (72), but that is not always the case.

(72)

(a) The dogs run round the garden.

(b) The dog runs round the garden.

There are phrases where the "agreement-number" differs from the "semantic number", (e.g. (73)), so these notions are logically distinct.

(73)

(a) These scissors are blunt, and your trousers have to be trimmed.

(b) Measles are not hard to cure, but cancer is. (Katz, p.379).

Also "mass" nouns (like "snow", "rice", "mist") cannot be given a semantic number, since they do not denote a set of discrete objects, but have a very definite agreement class (namely, Class 3).

(74)

(a) Snow falls softly.

(b) * Snow fall softly.

To emphasise the distinction between agreement classes and semantic plurality, notice that "you" is unmarked for semantic plurality (or else has two lexical entries, one for each plurality), but has a definite agreement class, Class 2.

(75)

(a) You run very fast.

(b) * You runs very fast.

The picture is further confused when we consider that verb agreement classes can (occasionally) have semantic effects in the understanding of sentences. Some subject noun phrases are unmarked for both semantic number and agreement class. In such cases, the verb form used may provide the missing semantic number information:

(76)

(a) The sheep runs very fast.

(b) The sheep run very fast.

Also, agreement classes cannot be marked statically in the lexicon, since they are a property of whole noun phrases. The assignment of phrases to agreement classes is productive (i.e. new examples can be allocated systematically) and seems to be based on semantic number. If someone tells you that a "sib" is a tool for cutting cloth, and that "latt" is a confetti-like substance, then you will probably regard (77)(a) and (b) as acceptable, but (78)(a) and (b) as somewhat odd.

(77)

(a) Twelve sibs are lying on the shelf.

(b) Some latt is pouring out of the bag.

(78)

(a) ???Twelve sibs is lying on the shelf.

(b) ??? Some latt are pouring out of the bag.

Brown (1958, pp.250-253) reports that children sometimes work in the reverse direction; that is, they make systematic guesses about the meaning of a new word on the basis of its verb agreement class in examples like (77).

These phenomena can be described in the following way. Each noun phrase has a syntactic agreement class (1, 2 or 3) and a semantic number (SINGULAR, PLURAL or MASS). Each inflected verb is marked with a list of the agreement classes which are compatible with it. (Listing them takes advantage of the fact that some classes are formed by taking the union of the basic three classes). Nouns like "sheep" (which are unmarked for agreement class or number) have two separate entries, with different agreement and syntactic number for each. (Hence the semantic information provided in cases like (76) is conveyed by elimination of ambiguity using syntactic agreement). Agreement compatibility can be tested in the following way. On finding a "subject" in a clause, a restriction is set on the syntactic property list of the node that the "complement" of the clause will be built on. This restriction states that the verb's agreement class list must contain the agreement class of the subject. The way in which number is treated within noun phrases will be dealt with below.

V.3.2 Determiner - Head Agreement

As mentioned above, agreement classes cannot just be marked on all noun phrases in the lexicon, since most noun phrases are constructed out of other lexical entries. Sometimes the determiner decides the agreement class, as in (79)(a) and (b); sometimes the head noun contributes the information, as in (79)(c) and (d); and sometimes both are marked for agreement class, as in (79)(e).

(79)

(a) A sheep is sinking in the mud.

(b) Many sheep are swimming in the loch.

(c) The dogs cavort in the tree-tops.

(d) Your children are spreading marmalade on our cat.

(e) Those apples are very sour.

(f) * These man is selling pornography.

When both constituents provide an agreement class, it must be the same one, or oddity results, as in (79)(f).

This is one place where the mechanism for combining properties of constituents (see Section IV.1) is useful. The SCR for combining determiners and head nouns can have an associate property inheritance rule, which passes up to the noun phrase node the agreement classes of both the determiner and the noun (if present). Examples like (79)(f) will cause the analyser to attempt to make two different entries for the same property, which is not allowed. Since determiners are often combined with more than a single head noun (for example, if there are adjectives or other modifiers before the head noun), any SCRs for constructing intermediate parts of a noun phrase (e.g. the adjective-noun combining rule) will also have to have a property inheritance rule which passes up the head noun's agreement class.

Section V.4 : Wh-Clauses

V.4.1 The Surface Structure of Wh-clauses

English relative clauses, and certain question-forms, share a common surface structure, consisting of a sentence (or partial sentence) with a "wh-word" ("who", "what", etc.) at the beginning.

(80)

(a) What did you buy ?

(b) The car which is parked outside has a flat tyre.

(c) The man who you sold the book to wants his money back.

(d) When did he leave the house ?

This formal similarity suggests that these clauses ("wh-clauses") may be describable in similar terms. This subsection examines how a recognition grammar can handle the surface form of wh-clauses, and later subsections will examine the semantic regularities involved.

The way that wh-clauses have been described in transformational analysis is as follows. The clause is regarded as a complete sentence, where one constituent (usually regarded as a noun phrase) has been converted to a "wh-phrase" (either a wh-word or a phrase starting with "which", "what", or "how") and moved to the beginning of the clause from some position within the clause. This way of looking at the structure has certain advantages, since the relationships between the wh-phrase and the rest of the clause are generally the same as those which would hold between an NP in the

"original" position and the rest of the clause. Semantically, the wh-phrases in (80) can be regarded as displaced versions of the subject, object, indirect object, and time-adjunct, respectively. Syntactically, subject-verb agreement seems to occur between the wh-phrase and the main verb of the wh-clause, if the "original" position is that of subject (cf.(80)(b)). Fuller arguments for this way of describing wh-clauses can be found in Kuroda (1966), Burt (1972), Chomsky (1964) and Ross (1967).

In the framework being used here, there are two ways to describe wh-clauses. One way would be to accept the spirit of the transformational analysis, and write a recognition grammar which attempts to locate the wh-phrase in its "original" place in the clause. Alternatively, we could postulate a wh-clause SCR which takes the meaning of a wh-phrase as its first argument, and the meaning of a sentence (or partial sentence) as second argument, and produces a suitable semantic form. The former approach has been adopted here, because of the following difficulties with the latter method.

Firstly, the wh-clause SCR would have to duplicate all the semantic intra-sentential relationships (e.g. subject-to-verb, time-adjunct-to-main clause, etc.) since these can all be used between wh-phrases and their associated clauses.

Secondly, the wh-clause SCR would have to include subject-verb agreement as a special case, since it would be treating an "object" wh-phrase and a "subject" wh-phrase as merely two possible options in a list of relationships.

Thirdly, the criteria for deciding which intra-sentential relationships to use are not primarily semantic (i.e. they are not based on the semantic structure of the two arguments for the putative SCR), but are surface syntactic. If the wh-clause sentence lacks a surface subject, then the wh-phrase can be used regardless of what deep roles (in the terms of Section V.8) are filled.

Fourthly, if the sentence-part of the wh-clause is a sentence-fragment (i.e. lacking a subject or object), then this cannot be analysed correctly by the simple sentence recognition procedure. For example, if we assume that "say" must have a surface object, then (81)(a) is an acceptable sentence, but the sentence fragments (b) and (c) are not.

(81)

(a) What did you say ?

(b) * Did you say ?

(c) * You say ?

Hence some recognition procedure is needed for the second part of a wh-clause which is different (albeit in some minor fashion) from the simple sentence grammar, and which takes into account the fact that a wh-clause is being processed.

Both SHRDLU and LSNLIS parsers (see Sections II.9, II.1) adopt this approach, using structure-holding registers to perform the task. On encountering a wh-clause, the opening wh-phrase is processed and stored temporarily in a register. The rest of the clause is then

processed by a procedure which is similar to the sentence procedure, except that it may use the item stored in that register as part of the surface structure. This general strategy has been incorporated in the MCHINE grammar, but some interesting questions still remain. In particular, how do we ensure that the analyser processes the sentence successfully, and uses the stored wh-item in a suitable place ?

Consider a sentence like (80)(d) above. The sentence part of the wh-clause ("did he leave the house") constitutes a complete clause, and so the analyser should find the right surface structure for it without failing. On attempting to leave the level at which the wh-clause has been processed, the tidying up procedure (see Section IV.8) will cause the wh-phrase left in WHSLOT (the register allocated to this purpose) to be found. Suppose we incorporate into the tidying-up procedure the option of performing rule-selection (see Section III.9) to combine the left-over item with the current subtree. If this works, the Time-Adjunct SCR (see Section V.7) should be selected, and the wh-phrase attached as a time-adjunct to the sentence part of the wh-clause. A similar analysis might work for place adjuncts, e.g. "Where did he see you ?". However, wh-clauses in which the wh-phrase has to be treated as an obligatory surface structure constituent (such as subject, object or indirect) object, cannot be handled in this way. Consider (80)(a). If the grammar requires "buy" to have a surface object, and the analyser does not find one after the verb, it may discontinue the analysis (since we are assuming that the analyser will not treat strings like "Did you buy?" as acceptable sentences). The tidying-up process is invoked only on leaving a level (at a "POPUP", in the ATN

terminology), so it would not help here, unless we introduced the additional principle that, before abandoning an analysis, the analyser should first try to leave the current level. Even if this could be made to work, it would be suitable only for situations where the wh-phrase has to be located at the end of the sentence fragment. If the wh-phrase has to be put in subject position (as in (80)(b)), the lack of a consituent will show up long before it is feasible to leave the level of the wh-clause. One solution might be to have another general principle that, if a failure is about to occur, the analyser should try to use any structures in the current set of grammatical registers to fill the current structural requirement, and, if successful, continue. This is unattractive, for two reasons. Firstly, it seems to be introduced simply to patch up this problem, with no other justification. Secondly, it needs a clear definition of what would be regarded as the "current structural requirement" and where (in the recognition rules) to re-commence if the structure-filling process succeeds.

It turns out that there is one quite neat way to ensure that the stored item is available exactly as needed. Notice that the wh-phrase can be regarded as a form of "noun phrase" (cf. the transformational version of this phenomenon) and wh-phrases will always be allocated to a position that a "noun phrase" could have filled. We can therefore write the noun phrase grammar (i.e. an ATN subnetwork) so that one of the options is to look in WHSLOT for a stored structure. Then, at any point where the analyser is looking for a noun phrase, it can fulfil this expectation by using the stored WH-item, if one is present. Using this trick has certain advantages. Firstly, it means that the sentence fragment in a wh-clause can be

processed using exactly the same network as is needed for an ordinary sentence - the insertion of the stored wh-item will happen if and only if it has been stored. Secondly, it allows the wh-item to occupy any position that a noun phrase could have occupied, without separate rules for each case. in particular, since a prepositional phrase includes a noun phrase, dangling prepositions (e.g. (82)(a) and (b)) will be automatically covered.

(82)

(a) Who did you speak to ?

(b) The man who I was addressed by is here.

This will not interfere with the method already outlined above for optional adjuncts. By virtue of being optional, the adjunct wh-phrase will not be sought by the sentence grammar (via the NP network) and so should still be in the WHSLOT at the end of the level.

V.4.2 The Complex Noun Phrase Constraint

If we adopt the description in V.4.1 of how to analyse wh-clauses, there are some interesting consequences. Since WHSLOT is a grammatical register associated with the level processing the wh-clause, it behaves like a "local variable" in a programming language (see Section III.5). That is, each time the analyser initiates a wh-clause, a new version of WHSLOT is set up at the current level, only one incarnation of WHSLOT can exist at each level, and so only the contents of the current WHSLOT are accessible. If one wh-clause is processed inside another (e.g. in (83)), the

inner wh-clause will not be able to use the contents of the WHSLOT register for the outer clause, since that register is at a higher level, for which the processing and register contents are temporarily suspended.

(83) What did the man who you saw say ?

Consequently, the analyser will become confused if two wh-phrases are associated with one wh-clause, both at different levels, as in (84).

(84) * What did the man who bought arrive ?

(Where the intended interpretation is that "what" represents the object of "bought"). This is not to say that two wh-phrases cannot be associated with one relative clause - cf. (85).

(85) What did you say to whom ?

Here, both wh-phrases are located at the one level, and can both be processed by the one use of WHSLOT. The two wh-phrases will occupy the single incarnation of WHSLOT at different times (if in fact the second one ever has to be stored). Also, there is nothing to prevent the wh-phrase appearing in a containing clause being used in a lower clause, provided it does not, as in (84), become entangled in another wh-clause.

(86) What did Mary think Fred liked ?

In (86), the "what" is intended as the object of "liked", but appears at the opening of the clause "did Mary think...", this creates no

problems, since no other use of WHSLOT occurs, and so the one version of WHSLOT established at the start of the sentence will still be untouched when needed at the end. Several wh-clauses may appear in the same sentence, provided the wh-phrases are kept adjacent to their associated sentence parts, allowing the WHSLOT register to be used properly.

(87) Where did the man who you mentioned bury the treasure which he stole ?

This pattern (i.e. the acceptability of sentences like (83), (85), and (86), but the oddity of sentences like (84)) has been documented in the transformational literature. The best known description of the phenomenon is due to Ross (1967), who formulated it as a constraint on alterations to deep structures :

(88) The Complex Noun Phrase Constraint

No element contained in a sentence dominated by a noun phrase with a lexical head noun may be moved out of that noun phrase by a transformation.

The procedure outlined in V.4.1 was based on providing a simple and effective way of locating wh-phrases in wh-clauses. The notion of a register being local to a level was not constructed solely for this analysis, being an obvious computational technique for processing constituents as embedded units. Nevertheless, this procedure (based on surface analysing procedures) appears to embody Ross's constraint insofar as it applies to wh-clauses.

The Ross rule (88) also covered complex noun phrases (hereinafter CNPs) other than wh-clauses, so it is natural to examine these other cases to see how the procedure here relates to them. The other form of CNP was that where an ordinary noun phrase (NP) and a clause (S) were in apposition :

(89)

(a) You believed the claim that he was an agitator for a washing machine firm.

(b) The fact that it was Tuesday was overlooked.

Let us investigate what similarites there are, in surface structure, between relative clauses and appositional CNPs. A sequence of the form "NP + that" may be the start of either of these :

(90)

(a) I didn't believe the claim that he made.

(b) I didn't believe the claim that he made a mess of things.

A possible way for the analyser to handle this potential ambiguity is to consider both possibilites in parallel. This has no particular drawbacks apart from the general unattractiveness of exhaustive searching, as discussed in Section III.6. However, let us consider one way in which the decision could be postponed by using a register. Let us assume that, after getting the intial "NP + that" sequence (or perhaps the relevant sequence is NP + that + NP, since criteria for recognising the beginnings of relative clauses are difficult to

state), the analyser decides that a CNP (of some type) is to be processed but it does not know what the relationship between the head NP and the sentence-part will be, since it is either apposition, or one of the relative-clause possibilities. The analyser therefore does not build the head NP into the surface SCR tree, but holds it in a register, and goes on to process the sentence part of the CNP. This shows the similarity between the appositional CNPs and relative clauses - they consist of a head phrase and a sentence-part, where the semantic relationship between the two points is unknown at the start of the clause; the head phrase can be stored temporarily until the end of the clause. The appositional relationship is not intra-sentential, like subject or object, but is extra-sentential, like the optional time and place adjuncts. If we assume that there is an SCR for apposition, the relationship will be found at the same stage as one of the adjunct relationships would be found in a relative clause - namely, during tidying up. The wh-clause strategy will then serve for both types of CNP, and the limits on registers discussed above will apply equally to both. Hence Ross's constraint (88), in its full form, will be incorporated into a single method of dealing with appositional and wh-clauses, and one more case of breadth-first searching has been avoided using registers.

Unfortunately, there are certain problems with this approach. In some sentences, the CNP constraint appears to be inapplicable :

(91) What did you make the claim that you liked ?

(If (91) is unacceptable in some idiolects, then presumably what follows here does not apply to the grammars of those idiolects).

In (91), a wh-phrase, the object of "liked", has been placed, in surface structure, outside the CNP which contains it. By Ross's constraint, and by the analysing strategy given here, this should result in an odd sentence. Intuitively what seems to be happening is that the sequence "make the claim" is being treated as a single verb (roughly synonymous with "claim"), and the "that + S" sequence is treated as the object of this "verb". (Cf. "What did you claim that you liked ?"). It is not at all clear how this could be handled in either a transformational or a computational account, and it is slightly obscure even how to treat it as an exception. Perhaps the neatest solution would be to enter "make the claim" as an idiom or compound verb in the lexicon. The analyser would then be faced with two possible analyses for sentences beginning like (91). There would be one analysis where "the claim" is part of the verb, and one where it is the head of a CNP. The latter analysis would eventually fail, since the displaced wh-phrase would cause problems; that would leave only the analysis where "made the claim" is a compound verb. Since computational grammar has, at present, no way of treating idioms or compound lexical entries, this suggestion is not very enlightening.

David Kilby (personal communication) has observed that in some examples, the ambiguity between relative clause and appositional CNP is not resolved even at the end of the sentence :

(92) Did you consider the claim that he made an over-simplification ?

The two interpretations of (92) are describable as follows. The relative clause interpretation results from the head NP "the claim" being taken from the WHSLOT and inserted as the object of "made", with the NP "an over-simplification" being processed as the second object of "consider". The appositional CNP interpretation results from "an over-simplification" being inserted as the object of "made", with "the claim" being left in the WHSLOT register until the end of the clause.

Whether a particular grammar would find both these analyses or not would depend on the finer details of the mechanism for getting an item out of the WHSLOT. If we introduce some form of ordering between options (i.e. between ATN arcs in each state), then the analyser will get only one of the analyses. That is, if the grammmar specifies that stored wh-items are to be used in preference to looking for an input noun phrase (where both are available), then only the first interpretation of (92) will be found. This would be a rather awkward ordering, as it would empty the WHSLOT into the subject position of every wh-clause. On the other hand, if input phrases have priority (i.e. WHSLOT is searched only if a gap is found), then only the second interpretation will found. In fact, it seems better not to order these two options, since either ordering will cut out possibilities that we need to include in the grammar. Sentence (92) is a special case of the fact that certain wh-clauses may present the analyser with a choice of using a stored wh-item or analysing the next part of the input string :

(93)

(a) What did you give your father ?

(b) What did you give to your father ?

For example, assuming that the SCRs used within wh-clauses are exactly those used in ordinary clauses for combining verbs and objects, (93)(a) and (b) differ in the order that "what" and "your father" are to be placed in the SCR tree.

The MCHINE grammar places no ordering on the two options (using a wh-phrase and using an incoming NP), and so will find both analyses in cases like (93).

V.4.3 Semantics of Wh-Clauses

It was suggested in V.4.2 above that the best surface structure for a wh-clause was one similar to that of a sentence, but with a wh-phrase as one of the constituents. This assumes that the SCRs for combining the constituents of a sentence will function equally well with either a wh-phrase or some other item (such as an ordinary NP). Such a situation is desirable, since otherwise we would need a duplicate set of SCRs, for all the cases where one of the arguments was a wh-phrase. Now it is necessary to consider what the semantic representation of a wh-clause should be, and whether the representation will allow such assumptions about the surface structure.

Consider the various uses of a wh-clause (exemplified in (94)). In a question, it acts as a pattern to single out some set of items about which the questioner wishes further information . In a relative clause, it acts as a predicate to convey more information about the antecedent noun phrase (apart from the separate construction described in V.4.4 below). In a relative clause without an antecedent, it acts as a descriptive term to refer to some item.

(94)

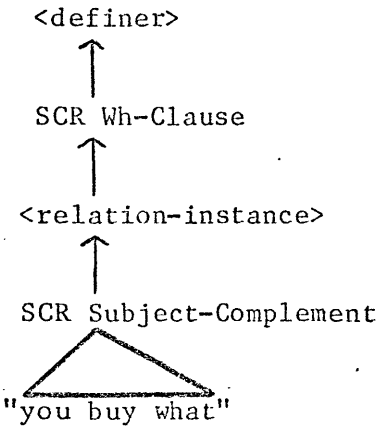
(a) What did you buy ?

(b) The book which you stole is boring.

(c) What you did was wrong.

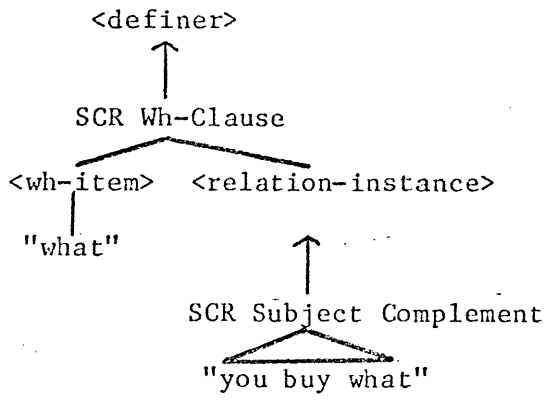
If we simply let a wh-clause meaning be a clause-meaning (i.e. a relation-instance) with one component marked as the "wh" part, all the semantic rules for handling semantic structures would have to check every relation-instance to see if it had a "wh" component, since the behaviour of an ordinary relation-instance is completely different from this multi-purpose structure. On the other hand, one semantic structure which is ideally suited to this multiple role (as a pattern, a predicate or a term) is the "definer" (see Sections III.10, IV.3). If we can arrange our rules so that wh-clauses are a form of definer, we may be able to capture all these uses in one structure. This will require some extra SCRs to construct these wh-clause definers, since at present we have assumed that only the ordinary clause SCRs are necessary. It may seem that what we need is a unary SCR which acts on a wh-clause (represented as a relation-instance) to produce a definer, e.g.

(95)



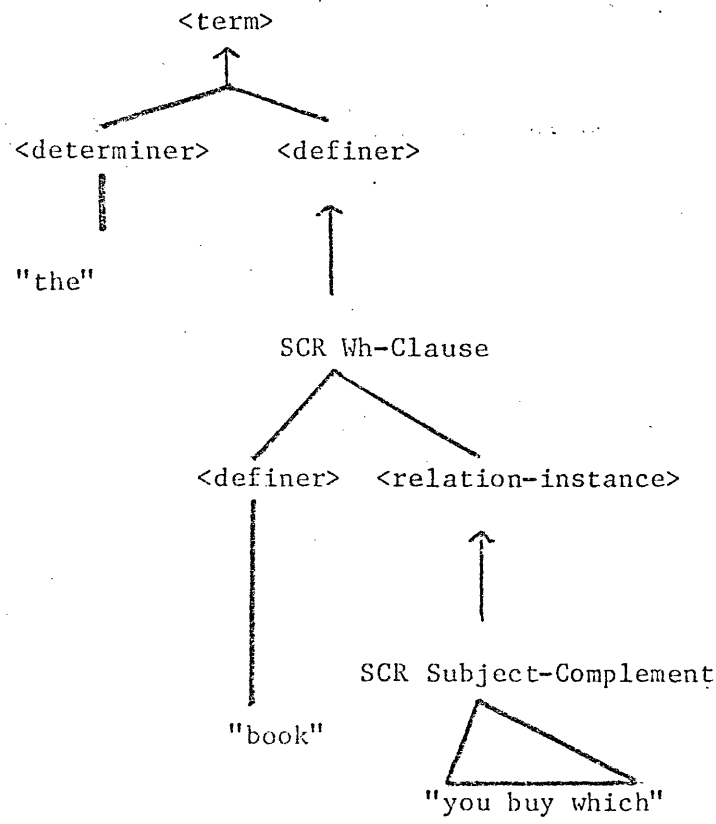
However, if we make the rule binary, with the wh-phrase forming the first argument (e.g.(96)), certain advantages accrue.

(96)



In particular, it allows for the fact that, in ordinary relative clauses, the semantic content of the wh-item is spread between the antecedent nominal and the relative pronoun. For example, the information that the underlined phrase in (94)(b) refers to a book derives from the antecedent "book". Therefore we might want to have some way of incorporating the semantic content of the antecedent into the overall meaning of the wh-clause. The phrase in (93)(b) might have a tree like (97).

(97)



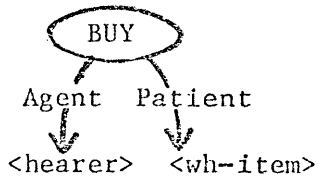
If we have organised the representation of referring expressions so that they are normally represented as definers (see Section V.6), then the two descriptions will connect quite well, since relative clauses (both with and without antecedents) will produce the same sort of structures (definers) as ordinary noun phrases. The only other interface to be arranged is for the question wh-clauses. The conversational routines will have to be able to manipulate a definer (suitably marked) at any stage where a question might occur. (See Chapter VI for a possible way of achieving this).

The way of converting a relation-instance with one component holding a wh-item into a definer is straightforward for simple examples. The wh-clause SCR has to find the component with the wh-item, mark it as "blank", and select it as the selected role when

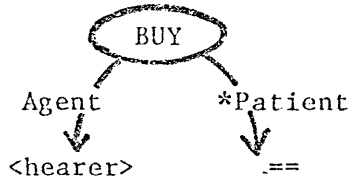
constructing the definer. That is, a relation-instance like (98)(a) would become a definer like (98)(b).

(98)

(a)



(b)



Marking the selected component as blank has one undesirable consequence - the information in the wh-item is lost. This information constrains the class of items that the definer refers to, and may be quite rich, as in (99).

(99) Which large green furry object did you buy ?

The information can be retained, in quite a natural way, by transferring it to the restriction for that component in the definer. Then the definer will denote only items which match the description originally specified in the wh-phrase.

V.4.4 Non-restrictive Relative Clauses

There is yet another way in which wh-clauses can be used. Corresponding to the distinction between restrictive and non-restrictive adjectives (see Section V.1 above) there is a distinction between restrictive and non-restrictive relative clauses.

(100)

(a) The girl who you met last night rang up.

(b) Alice, who I cannot stand, rang up.

Some relative clauses (e.g. (100)(a)) act as modifiers to restrict the scope of the antecedent phrase, thereby constructing a more narrowly referring expression. Other relative clauses (e.g.(100)(b)) do not restrict the range of items specified, but add incidental information about the item(s) that the speaker wishes to convey. (See Smith (1964), Thompson (1971), and Ross (1971) for some comments on these clauses from a transformational standpoint).

The other main difference between the two types of relative clause is that, while the restrictive relative clause combines with the head nominal of the antecedent to create a more complex nominal (which other items such as determiners can interact with), the non-restrictive relative clause makes an assertion about the semantic item produced by the entire antecedent phrase. Hence restrictive relative clauses cannot be appended to proper names unless the name is treated as the head nominal of the noun phrase, e.g. :

(101)

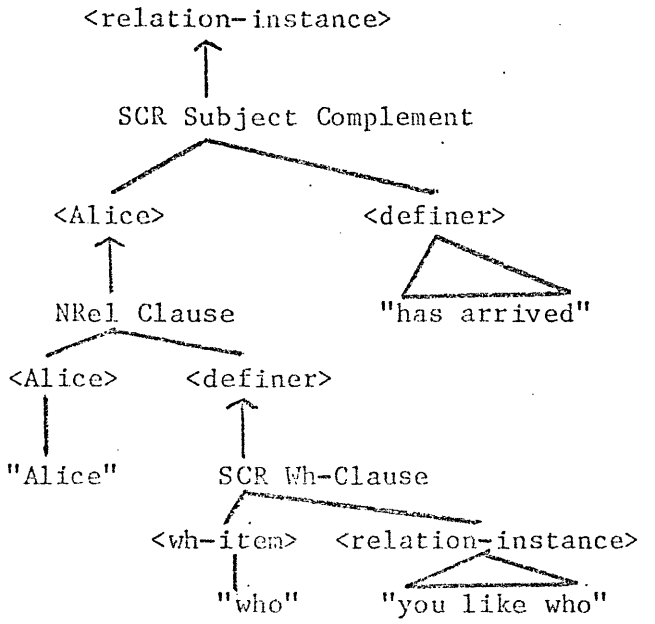
(a) Alice, who you like, has arrived. (Non-restrictive)

(b) The Alice who you like has arrived. (Restrictive)

The only problem in incorporating these clauses into the wh-clause grammar outlined above is finding some way that the analyser can distinguish between the two (cf. comments in Section V.1.2 on adjectives). As Smith (1964) comments, non-restrictive relative clauses often have "comma intonation" surrounding them, and commas are the usual device for indicating the usage in written English. The MCHINE grammar can analyse relative clauses as non-restrictive if and only if they are surrounded by commas, since its ATN includes an explicit test for an opening comma at the start of the network for non-restrictive relatives.

The rest of the grammar is fairly straightforward. There is an SCR "NRel-Clause" which takes any item and a wh-clause meaning as its arguments, makes an assertion about the item, using the wh-clause, and then returns the item unaltered as its result. The latter stage may seem redundant, since the main purpose of the SCR is merely to make the assertion, but the analysis process has been designed on the assumption that the SCRs will form a tree (with results from lower rules forming arguments for higher rules). Hence every rule must return one result. (See similar comments in Section V.1). The trees for non-restrictive relative clauses will be as in (102).

(102)


<relation-instance>
SCR Subject Complement
<Alice>
<definer>
NRel Clause
"has arrived"
<Alice>
<definer>
"Alice"
SCR Wh-Clause
<wh-item>
<relation-instance>
"who"
"you like who"

Section V.5 Limits on Embedding

V.5.1 The Idea of Complex Embedding

It has often been noted (e.g. Yngve (1960, 1961), Miller and Isard (1964), Chomsky (1965)) that certain kinds of English sentence are hard to understand, although their grammatical structure seems to follow the same pattern as other, easily understood, sentences. Examples include centre-embedded clauses (e.g. (103)(a)) and verb-particle constructions containing "heavy" noun phrases (e.g. (103)(b)).

(103)

(a) ? The rat the cat the dog chased bit died.

(b) ? He called almost all the thirty Philipinos he met on his Globtik holiday in France up.

Investigations of these phenomena sometimes seem to assume that such complexity will be accounted for by some single principle, and that the actual structure of the whole sentence will be the relevant factor in determining complexity. Neither of these assumptions is necessarily true - there may be several principles which contribute to "complexity", and the most difficult examples may fall under more than one of these principles. Also, the relevant parameters for determining complexity may be within the processes of perceiving or producing the sentences, and may not be directly measurable in the final structure of the sentence.

One measure of complexity which is possible is the depth of the surface structure tree, in terms of the longest path from the root to a tip. This does not account very adequately for the data, as certain constructions appear to be easily comprehended despite great depth; left-branching or right-branching structures, such as (104)(a) and (b), are the usual examples (Yngve (1960), Chomsky (1965)).

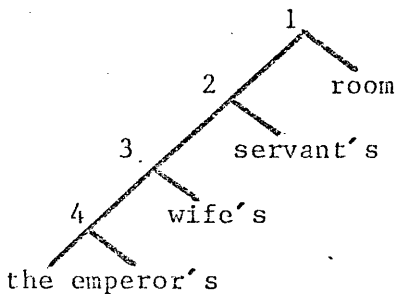
(104)

(a) The emperor's wife's servant's room.

(b) The room of the servant of the wife of the emperor.

Yngve (ibid.) put forward a production model, with a more subtle definition of "depth". His "depth" was measured in terms of partially-processed surface structure nodes. This measure was more satisfactory, in that it did not rate right-branching structures as being complex (i.e. (104)(b)) would not be classed as complex), but it did rate left-branching structures as complex, which is counter-intuitive. This resulted from Yngve's model being for production of sentences, rather than for analysis; the producing mechanism therefore started from the root of a subtree like (105) and began producing nodes down the left of the tree, depth-first. The left-branching construction caused the nodes to be initiated in the order indicated in (105) (1-2-3-4), and completed in reverse order (4-3-2-1).

(105)



Such constructions therefore resulted in a large number of partially completed nodes, and hence had a large "depth".

Let us consider how a recognition mechanism might deal with some of these examples, to see whether the complexity might result from difficulties in the sentence analysis process. We will first examine right-embedding and centre-embedding constructions, before returning to the question of left-embedding constructions. Intuitively, most (or all) of the "difficult" sentences are cases where a large constituent (in some sense) has been inserted in a central position in the sentence. If a small consituent is inserted, or if the large constituent is at the right-hand end of the sentence, oddity does not occur.

(106)

(a) I told my story to the police.

(b) ?? I told the story that I had heard from a crowd of Arabian horse-thieves who were planning to set up a Citizens Advice Bureau in Friockheim to the police.

(c) ? He said that he had an appointment to the secretary.

(d) ?? He said that almost all of his friends were founder members of the First International League of Bad Linguists to the secretary.

(e) He said that almost all of his friends were founder members of the First International League of Bad Linguists.

(f) I said that Bill knew that Harry thought that Fred was a crook.

Informally, what seems to be happening is that the analyser, while processing the embedded item "forgets" some information relevant to the containing clause, and so has difficulties when it attempts to recommence processing at the higher level. As Miller and Isard suggest:

"Let us imagine that anyone who knows English has something corresponding to a relative clause subroutine that can be called to process sentences containing such structures. When this subroutine is called, the main sentence-analysing routine is interrupted and the point at which it must be resumed is stored temporarily until the subroutine has been executed.....suppose that, while the subroutine is being executed, a second such construction is encountered, so the subroutine is required to call itself. If this recursive feature were not available, confusion would result; the temporary memory for the point of re-entry into the main routine might be erased, for example, so that when it is resumed, the main routine would have to treat subsequent words as if they began a new constituent of the sentence."

(Notice that this was written some years before the augmented transition network model was suggested for English). This would be compatible with the fact that larger constituents (which presumably require greater processing resources) cause greater difficulty, and with the fact that no difficulties occur in right-branching structures where no segment of the higher clause is left after the embedded item (e.g. (106)(f)). Notice that it is not plausible to suggest that it is the initial storing of information about the containing clause that goes wrong, since phrases like (106)(d) and (e) show that the trouble derives not from embarking upon an arbitrarily long constituent, but from trying to re-commence the containing clause afterwards. If we examine this in greater detail, it turns out that the subroutine analogy can be described precisely using the computational grammar devices outlined in Section III.7.

V.5.2 Right-Embedding

In the standard ATN or PROGRAMMAR mechanisms, each embedded constituent is processed at a different "level", and these levels are organised in a strict hierarchical fashion, corresponding (usually) to the structure of the surface tree. For example, in PROGRAMMAR, when a parsing subroutine exits normally, it attaches its result to the current node of the syntax tree. Normally a tree with n embedded structures will have been processed at n different levels. Each level is initiated by a "PUSH" arc and terminated by a "POP" arc (in ATN notation), with interpreter information about higher levels being kept on stacks while lower levels are processed. The kind of information that the interpreter needs to retain may depend on the exact details of the theory of surface structure being used, but the

MCHINE program has to maintain separate values in several registers for each level (see Sections IV.6 and VI.3.4).

The information about a particular level has to be retained until it is finished, and the only way of finishing it is to "pop-up"; this, in turn, can happen only when all lower levels have been completed and "popped". Hence information builds up on the interpreter stacks as levels grow downwards. It is quite plausible to suggest that this building-up is what contributes to complexity. However, embedded constituents are all treated similarly, whether embedded in the centre or at the extreme right of a clause. In either case, all the information builds up for each level. Hence it is not simply that overloading stacks causes trouble, for then both centre- and right- embedding would be perceptually difficult. It is more that overloading causes some kind of "forgetting", and this lost information is not needed in right-embedded structures since no further processing occurs at the higher level.

If we were not interested in modelling linguistic behaviour in detail, we could continue with the traditional approach of allowing the parser unlimited space on stacks, to process arbitrarily complex constituents (which might be incomprehensible to humans). However, since the aim is to restrict our model wherever possible and to reflect human linguistic ability as much as possible, it seems reasonable to investigate how the embedding facilities can be limited. The exact bound on storage and/or processing is possibly very subtle, but a crude approximation could be achieved by putting a limit on the interpreter stacks, in such a way that bottommost items are lost if overloading occurs. This still leaves various

possibilities for using such stacks in the ATN interpreter.

We could continue the traditional method of storing information on the stacks even for right-embedded structures, but organise the structure-building mechanism so that the surface structure tree would not be incomplete or ill-formed if certain levels were never re-activated by the parser. Alternatively, we could have two different processes, centre-embedding and right-embedding, where performing the latter does not increment the interpreter stacks and leaves the structure at the current level in good order before "forgetting" its details. The main difference between these two is that the traditional method "forgets" levels only when forced to, and "remembers" information at every embedding; in the second method, the analyser decides at the start of a constituent that it is not going to return to the current level again, and abandons the information immediately.

The latter approach has been incorporated into the MCHINE program. Although there is little to choose between the alternatives, the second method was preferred for the following reasons. Firstly, it fitted in well with the idea (Section III.7) of not needing to specify continuation states on every PUSH arc. Secondly, it seemed easier to organise a structure-building mechanism where levels were abandoned deliberately, rather than being left to survive after being "forgotten". Thirdly, there were certain constructions (see below) where this decision method seemed very suitable. Fourthly, it seemed an interesting idea, whose feasibility should be investigated to see what happened.

The right-embedding device has already been described in Chapters III and IV, as it is the modified PUSH construct referred to as "NEWLEVEL...NIL". The interpreter should, on executing such an arc, create a new processing level without storing the details of the current level, as outlined in Chapters III and IV. (In the MCHINE program, this consists of attaching the current TOPNODE to the current SUBROOT, thereby ensuring that no loose ends are left in the tree, setting the node currently being worked on - CURRNODE - as the new SUBROOT, and setting up a new blank TOPNODE. The continuation stack, the action stack, and the register stack are untouched.)

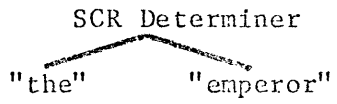
This device can be used quite neatly in analysing right branching noun phrases like (104)(b). Suppose that the analyser has completed the phrase "the room", and reads the word "of". (The traditionally intransigent problem of attaching prepositional adjuncts is ignored here, on the assumption that the argument offered here applies to at least one of the options facing the analyser). If it decides that a right-branching prepositional adjunct is about to be processed, the action it must take is exactly that of a (NEWLEVEL...NIL) arc. It needs a new level to process the next NP, but it has no new information about what will follow, since this second NP forms a structural subpart of the first NP. It has to assume that whatever was predicted to follow the original NP will now follow the total NP including the adjunct. It would just be an embarrassment to have to specify another continuation state, when no further predictions can be made about the grammatical environment surrounding the NP. Since the first NP can be regarded as finished apart from the adjunct now being started, no interpreter information need be kept concerning it. Instead of the stacks building up four

nested levels to process (104)(b), only one level is used at a time. Hence no "complexity" or unintended "forgetting" is caused by processing a multiply-branching phrase like (104)(b) inside a clause. Using the more traditional methods, some continuation state (say, STOPNP) would have to be specified each time, and the continuation stack would pile up three copies of it.

V.5.3 Left-Embedding

Let us now consider left-branching structures like (104)(a). As observed before, Yngve's model predicted a large "depth" for this construction, since his method was based on starting with the base of the complete subtree. The situation is different if we examine the structure in recognition terms. The analyser cannot know in advance that such a deep tree will be part of the input, and so it does not start by building a series of partially processed nodes. After processing the phrase "the emperor's", a whole node (like (107)) will have been completed (see Section V.1 above for more details).

(107)



The "'s" signals that this whole node is to be subordinated to a new node that must now be commenced. So at any given time, only one of the phrase nodes is incomplete and being processed.

Consider how this processing can be decribed using the constructs defined in computational grammar. As discussed in Section V.1, possessive noun phrases fulfil the same role (in the containing structure) as lexical possessives, and are paradigmatically related

to articles. Hence, after the analyser has built a possessive node, it should process the containing phrase from the appropriate point; that is, as if an article or lexical possessive had been found. It should not try to process a whole noun phrase, since another article or possessive should not be present.

(108) * The emperor's a wife lives here.

Since the phrase forming the possessive has been completed, information held in registers for it need not be retained, and some of these registers will be needed for the incoming phrase. The appropriate command to insert in the grammar at this point (as observed by Winograd (1972, p.93)) is a jump to the state following a "determiner" or "possessive", and a re-initialisation of any registers. Notice that the grammatical registers need not be pushed down, since the old values are being discarded. We do not need to save values of the interpreter register for the completed possessive phrase, so these can also be re-allocated without pushing down. Hence no new levels are created, whether by nesting or by overwriting the old one.

Left-branching structures create difficulties for any simple top-down recognition grammar, since there is a danger of infinite recursion. Some special modification is usually needed in a context-free parser which uses recursive left-branching rules. It should be emphasised that the above approach to the problem is just such a special-purpose modification, and there is no claim that the ATN interpreter in the MCHINE program can handle any left-branching structure, as currently written. In order to analyse left-branching

phrases, specific instructions had to be inserted in the ATN grammar in the program. A similar method might be employed for left-branching structures (for example, in another language) but it would have to be included in the recognition grammar of that language.

V.5.4 Relative Clauses

Another classic case of embedded structure which is difficult to perceive is that of relative clauses within relative clauses, as in (103)(a) above. It is interesting to assess what the computational analysis of relative clauses given in Section V.4 suggests regarding the possible processing difficulties presented by sentences like this.

There are several aspects of this construction which lead to a great deal of storage space (in interpreter and grammatical registers) being used :

(109)

(a) The relative clauses are all attached to the subject of the containing clause. In writing the MCHINE grammar, it became apparent that the subject position is one of the few places in English grammar where a NEWLEVEL...NIL arc is not an adequate form of embedding, since some continuation state must be specified (for the start of the complement of the clause), and the new processing level must be nested. Hence, the register values for the containing clause are being saved while the subject phrase is being analysed.

(b) The relative elements are all to be located as objects of their clauses, so that they must be stored from the start of the clause until the end.

(c) The wh-clause uses all the registers of the clause network, plus a structure-holding register (WHSLOT). This is more than any other class of constituent.

Also, a great deal of processing must be carried out at the end of the clauses, since three wh-clauses all terminate within a few words. At the end of each wh-clause, the analyser has to tidy up registers at that level, fit the contents of WHSLOT into surface structure, and pop up stacks to resume the higher level.

Due to the points made in (109), the interpreter and grammatical stacks are more deeply piled with entries during the centre-embedded relative clause construction than in most others. Thus, if there were a limit to how many entries can be made on a stack, with earlier entries being lost, this could also create difficulties.

There are several factors which might be thought to contribute to complexity, e.g. :

(110)

(a) Total storage space required

(b) Total processing required (if this could be quantified)

(c) Depth of stacked information

(d) Loss of information about a higher level (owing to (a), (b) or (c), or some other factor)

It can be seen from the points made here that the description of Section V.4 will predict that multiply-embedded relative clauses are "complex", whichever metric is selected from (110). This is not very interesting, since the same is likely to be true of almost any serious description of English relative clauses. However, there is one point worth noting about the MCHINE grammar version, if our complexity metric is based on (110) or related factors.

A more standard ATN analysis of relative clauses will also predict that any multiply-embedded structure is complex, since information builds up on stacks for every constituent processed, as outlined in V.5.2 above. In the MCHINE grammar, this is not the case, since the NEWLEVEL...NIL construct allows some constituents to be nested without loading up the stacks. Hence the discrepancy in complexity between a right-branching and a centre-embedding structure is much clearer in the MCHINE grammar than it would be in a standard ATN description. The self-embedded relative clause is one of the few cases where genuine nesting (as opposed to NEWLEVEL...NIL processing) has to occur to several layers. (One of the other examples is the embedding of a clause within the first of two objects in a clause, as in (106)(c) and (d)).

After the MCHINE program had been debugged on its target sentences and had carried out several simple dialogues (see Chapter VI), a test was carried out to see how the NEWLEVEL...NIL construct interacted with multiply-embedded relative clauses. There is a

variable DEPTH in the MCHINE program which records the maximum number of entries allowed on stack registers, and the program had been debugged with DEPTH set to 8. This allowed at least four self-embedded relative clauses to be processed successfully. With DEPTH altered to 3, the program could still successfully process non-centre-embedding sentences, but failed on triply-embedded relative clauses.

To summarise, the analysis of relative clauses presented in Section V.4 indicated that multiply embedded relative clauses will be complex, if any of the factors in (110) are taken to determine complexity. The vacuity of this observation is mitigated by the point that a grammar with the NEWLEVEL...NIL device distinguishes more sharply the complexity of centre-embedded relative clauses from that of other embedding constructions. The idea of using a subroutine-like mechanism for embedded constituents is not new, but the NEWLEVEL...NIL construct introduces a modified form of subroutine which seems particularly useful.

Section V.6 : The Semantics of Noun Phrases

The term "noun phrase" can be taken to cover a wide range of forms, including nominalised clauses. For the purposes of this discussion, we will focus primarily on the simpler forms - proper nouns, pronouns, and common nouns with optional modifiers before and after. Various classifications (e.g. "definite/ indefinite", "specific/ non-specific") can be made of such phrases, and these categories are generally treated as syntactic, rather than semantic. The object of this section is to show that these traditional distinctions are better looked on as semantic regularities, and to suggest how the patterns might be incorporated into a computational model.

The SHRDLU program (Section II.9) translated English noun phrases into procedures which, when executed, found the referent(s) of the phrase. As observed in Section III.4, this incorporates some aspects of the "sense/reference" distinction, provided that we take "referent" to mean some internal construct, rather than a concrete object. We will try to refine that approach, to see exactly what kinds of "meaning-procedures" there are, and what kind of operations can be performed on them.

V.6.1 Definiteness

Certain kinds of "definite" noun phrase (using the usual classification) seem to have a fairly straightforward behaviour. They can be regarded as providing descriptions which, in a particular context, will identify a unique set of objects that the hearer knows about.

(111)

(a) The red block

(b) This man

(c) The farmer's wife

Even if we could formalise this way of describing the meaning of NPs, it would be neither a necessary nor a sufficient criterion for traditional "definiteness". It is not clear whether proper names and personal pronouns are traditionally "definite", but they are certainly used to identify a unique set of objects that the hearer knows about. Winograd (1972) points out that it is odd to use a phrase like "the pyramid" if the context does not make it clear to the hearer which pyramid is referred to. Compare this with the comment by McCawley (1971b) that (112) sounds odd if the context does not make it clear who "he" is.

(112) He resembles Mike.

The further observation should be made that (112) also sounds odd if the context does not make it clear who "Mike" is. This alleged similarity between proper names and descriptive phrases may seem dubious at first, but in fact proper names, as used in English conversation, are not "pure names" in the philosophical sense; that is, there is not a one-to-one correspondence between names and objects. Names may be intended as an approximation to pure names, but the way that they are used is analogous to the use of identifiers in programming, where the value of an identifier depends on the context. The expression "Mike" in (112) may have different referents

in different contexts, since different people have different mutual acquaintances.

As Winograd points out (ibid., p.156), "definite" cannot be identified with "known to the hearer". There are various kinds of "definite" noun phrase which can be used to refer to objects not previously mentioned.

(113)

(a) My brother who lives in Chicago.

(b) The title of his new book.

(c) The tallest elephant in Indiana.

(examples from Winograd).

Winograd observes that (113)(b) apparently is allowable because the noun "title" denotes, in some way, a function defined on the set of books. Similarly, (113)(c), by providing a full description, allows the hearer to assume that such an object exists. Sentences like (113)(a) are more interesting - "brother" does not denote a functional relation (since a person can have several brothers), nor is there any guarantee that "who lives in Chicago" provides a unique description in the way that "tallest" does. If a speaker uses (113)(a) in addressing someone who knows nothing of his brother, the assumption is that this description provides all the relevant information (for this particular conversational exchange). If the hearer does know all the details of the speaker's family, the phrase is once again like those in (111), and the sentence may be

inappropriate if it does not sufficiently define a referent.

This raises a further aspect of "definite" noun phrases - the adequacy (and relevance) of the information involved. There seems to be a general principle in using definite noun phrases, that distinctions need not be made if irrelevant in the particular context. When the program of Davey (1974) is constructing referring expressions, it takes into account which distinctions are relevant to the context, using various equivalence classes within the domain of discourse. Oddity can result from not being consistent about what distinctions are relevant. (114)(a) sounds acceptable, on the assumption that which "end" is involved is irrelevant, but (114)(b) sounds odd, with the second clause suddenly introducing a contrast as being pertinent.

(114)

(a) I sat down on the end of a nearby log.

(b)?? I sat down on the end of a nearby log, and Harry sat on the thin end.

Let us try to cover these aspects of "definiteness" by the following generalisation :

(115) A definite noun phrase is used by the speaker to provide the hearer with all the relevant information to produce a set of referents; the information may be inherent in the description, as in the case of "functional" words, superlatives, etc.

The main inadequacy of (115) is that it says almost nothing, as becomes apparent on considering the idea "produce a set of referents" in more detail. In the case of some referring expressions (e.g. proper names, pronouns, certain phrases with "the"), the referents are to be found in the hearer's "world model", since they are things which he should know about already. However, in the case of phrases like those in (113), there seems to be a process of generating a referent on the spot, given a description which it must meet. Hence "produce" in (115) means "find or generate". At this point in the argument, (115) becomes so general as to cover many other kinds of referring expression. "Indefinite" noun phrases can be used to provide a description of an object the hearer is not acquainted with, and the speaker provides only the information he feels is relevant.

(116)

(a) A woman with a collecting can accosted me.

(b) Some vandal has wrecked this phone box.

The difference seems to be that "indefinite" noun phrases can be used only to generate a set of objects - there is no possibility of "finding" one automatically in the hearer's world model. (That is not to say that he could not infer, at some stage, who the subjects of (116)(a) and (b) refer to, but that would not be part of the information that the speaker is imparting in the noun phrase).

This contrasts with certain other noun phrases which we have been classing as "definite", where referents must be "found", but cannot be "generated" if the search fails. In particular, proper

names, personal pronouns, and demonstratives all seem to require that the hearer can actually identify what objects are referred to - the meaning of the phrases allows no construction on the spot of a referent(s) to match the description. If we class these phrases as "deictic", then the classifications so far are as follows.

DEFINITE and DEICTIC : find a referent.

DEFINITE and NON-DEICTIC : find a referent; if none, generate one.

INDEFINITE : generate a referent

(Rumelhart and Norman (1973) give a similar gloss for "definite", but they attribute this behaviour to the article "the", rather than to some more general property of definiteness which is possessed by various kinds of phrase).

V.6.2 Specificity

So far all the noun phrases have been regarded as referring to some set of referents in a given context. There are certain expressions for which this is not a very satisfactory way of describing the meaning. In particular, generic statements often do not have particular referents as their subjects or objects :

(117)

(a) Wolves eat meat.

(b) The wolf is a carnivore.

(c) A wolf is a carnivore.

(d) The monarch is the head of the Church of England.

The subject phrase sometimes does have a referent in a particular context, as in (117)(d), and there may be an interpretation of the sentence in which that is the best way to describe the contribution of that phrase. Nevertheless, in the cases being considered here, there is also a "generic" reading of the sentence, in which the statement conveyed is not about one particular referent in one particular context. Let us refer to the generic interpretation of the noun phrase as "non-specific" and the other interpretation as "specific".

Noun phrases like "the monarch" or "a wolf" can have either a specific or a non-specific reading, depending on the context of use, but certain other kinds of expression are less versatile. For example, it is very difficult to find a generic (non-specific) use of a proper name or personal pronoun. The subject of (119)(a) cannot be interpreted as meaning "anyone named Ed"

(119)

(a) Ed drinks Newcastle Brown.

(b) Anyone who protested was thrown into prison.

Also, some phrases (e.g. the subject of (119)(b)) are wholly non-specific. It might be thought that the set of wholly specific

phrases and the set of deictic phrases might be related, since they clearly overlap. However, some wholly specific phrases are not deictic, e.g. "Someone who protested was thrown in prison".

Intuitively, the non-specific phrases are being used as patterns, to make statements about any object(s) that match the pattern, whereas the specific phrases are being used to produce instances, in a particular context, which do match. It would be useful if we could find some single representation for the meaning of a noun phrase which could function in both these ways.

V.6.3. Predication

Noun phrases can be combined with the verb "be" to produce a verb phrase which can form the complement of a sentence. It would be preferable if whatever semantic representation we can devise for referring expressions will also describe their behaviour when they combine to form predicates, as in (120).

(120)

(a) Bill is the owner of that Mercedes.

(b) Your landlord is a Marxist.

The two previous subsections have indicated that a suitable representation might be some kind of "pattern" which can be either used directly (non-specific), or used to find a set of referents (definite), or used to generate a referent set (indefinite). This suggests that we consider either using the referent set to form the predication (effectively, treating the phrase in the complement as specific), or using the pattern itself to convey the predication

(effectively, treating the phrase as non-specific). If the phrase inside the complement is either wholly specific or wholly non-specific, there will be no choice, but it remains to be seen which is better for those phrases which can take either interpretation.

If we are making the assumption (as suggested in Section IV.8) that semantic processing is carried out locally as far as possible, then the difference between using the specific and non-specific forms would be as follows. The specific form would produce a referent set for its semantic structure, whereas the non-specific form would produce a pattern. In the former case, the subject-complement SCR would have to perform some kind of object identification process between the set of objects referred to by the subject and those produced by the complement. In the non-specific case, the subject-complement SCR would have to assert that the pattern was true of the subject set. There seems no way to choose between these, other than seeing which integrates best with other rules and operations required. It would be extremely useful, for example, if the structure chosen for such complements was similar to that chosen for ordinary verb-phrases, since then there would need to be only one subject-complement rule to cover both cases.

This can be achieved (as indicated in Section III.10) by using a "definer" for the semantic structure for referring expressions. As described in Section III.10, the definer is a piece of semantic network with one role "selected", and can be semantically processed in various ways. Referring expressions seem to need at least 3 semantic operations defined on them :

(121)

(a) Find-referent : use the semantic item as a pattern to get a set of refernts from the world model.

(b) Generate-referent : construct a new semantic item which fits this pattern.

(c) Construct-assertion : combine another item with the pattern to create a relation-instance.

In addition, they need to remain as patterns on some occasions, so that they can indicate a range of items. If we adopt the "definer" as the kind of structure for a referring expression, these operations can be defined in a fairly straightforward way. (121)(a) corresponds to fetching an item from the semantic network to fill the selected role. (121)(b) corresponds to constructing an item to fill the selected role (the role-restriction can provide information for this). (121)(c) corresponds to putting a semantic item into the selected role on a definer.

The phrases underlined in (122) could then provide a definer directly to combine with the subject item to form a relation-instance.

(122)

(a) Fred is <u>a teacher</u>

(b) John is Mary's father

This assumes that the "be" would not affect the overall semantic structure of the phrase, but that is not implausible. (Some transformational treatments of "be" regard it as absent in deep structure, being inserted by a transformation). Since verb SCRs ("role-placement rules" - see Section V.8) produce definers, the modularity of subject-complement SCRs is achieved.

Section V.7 : Tense and Time

V.7.1 The Previous Descriptions

This section discusses and attempts to develop two previous computational descriptions of the English tense system (Isard and Longuet-Higgins (1973), which is based on Reichenbach (1966), and Isard (1974)).

In conversation, we frequently indicate when (approximately) an event occurred by relating it to another event, as in (123).

(123) When I arrived, he had already left.

The "when"-clause can be regarded as setting up a "reference time", with respect to which the main clause is interpreted. We can talk of earlier events either by using past tense or perfect aspect, or both. Isard and Longuet-Higgins distinguish :

(124)

(a) the time at which the sentence is uttered

(b) the time of reference

(c) the times at which the events in the sentence occurred

Thus, in (123) "I arrived" gives the time of reference, which is before the time of utterance and after the time that "he left". That is, there are two event times in this example, one of which supplies the time of reference. Isard and Longuet-Higgins offer the following "rule of thumb" :

(125)

(a) past tense will be used when and only when the time of reference precedes the moment of speech

(b) present tense indicates that the two coincide

(c) the presence of "have" (perfect aspect) signifies that the time of the event in that clause precedes the reference time

(d) the absence of "have" indicates the coincidence of reference time and event time

The idea of distinguishing event time, reference time and utterance time seems intuitively useful for describing tense, but the "rule of thumb" is slightly misleading on points (b) and (d). Present tense can be used (as Isard (1974) points out) to refer to events in the future, and so reference times which succeed the utterance time can be set up using present tense, as in (126).

(126) When he arrives in London, he will have spent four hours on the train.

Regarding (125)(d), the exact relationship between reference time and event time (in the absence of "have") is unclear :

(127)

(a) When they built the fifth bridge, they took several tenders

(b) " " " " " ", they used the best materials.

(c) " " " " " ", there was a gala opening.

"Approximately coincide" seems the best summary of examples like these. When the data include clauses referring to intervals of time (which Isard and Longuet-Higgins were specifically excluding) even "approximate coincidence" is not a very good description. Nevertheless, (125)(a) and (c) seem to be generalisations which are worth retaining.

Isard and Longuet-Higgins also observe that it is not necessary for the reference time to be established within the same utterance as it is required - it may be a previously mentioned reference time :

(128)

Was he there when you arrived ?

No, he had already left.

The reference time for the answer is that set up by the "when"-clause in the question.

McCawley (1971b) suggests that a past tense sentence (such as "the farmer killed the duckling") is ill-formed unless the context provides a time-reference point for it. This is true only as long as "context" is taken to mean the entire knowledge of the hearer of the sentence. Certain historical statements do not need such explicit reference times, as the events themselves are well-known. Similarly, a sentence like "Angus went to school in Edinburgh" does not sound odd if the hearer knows that Angus is educated, and hence probably went to school at some time. This is relevant to the use of "when"-clauses, since it would be very difficult to carry on a

conversation if some events or states were not already self-supporting; the whole point of a "when"-clause is to use such an event or state to set up a reference time.

Isard (1974) develops these ideas further. He proposes two slots, PRESENT and REMOTE, into which reference times can be put. These slots maintain their values between utterances (to account for dialogues like (128)) unless either they are obliterated by some new value being put in the slot or the conversation switches to using some other slot. That is, (128) establishes the reference time referred to by "when you arrived" in the REMOTE slot; later utterances which use remote tense will automatically refer to that reference time unless some new remote reference time has been specified, or some utterance in the present tense has been interposed. Isard suggests that it is the task of the "time-binder" "when" to get the reference time from the subordinate clause and put it into the appropriate slot. The main clause then uses the slot which corresponds to its own tense - the "when"-clause should have filled this. The problem of "approximate coincidence" of events is simplified for the Isard program by interpreting "when e1, e2" as either "e1 is coincident with e2" or "e2 immediately follows e1", both of which are clearly defined in the domain he is using - events are moves in a noughts-and-crosses game.

V.7.2 Nested Tense Settings

The rules suggested by Isard and Longuet-Higgins for maintaining reference times between utterances seem to cover many cases. However, there are examples which seem to escape these constraints.

As commented above, an utterance may contain two event-times, one of which forms the reference time for the utterance. Sometimes this gives the next utterance in the dialogue a choice of two possible reference points - the already-used reference time or the event-time of the other (main) clause :

(129)

A : When I was at university, the Duke of Friockheim visited us.

B : Was there a big reception ?

(130)

A : When I was at university, the Duke of Friockheim visited us.

B : What other exciting events occurred ?

This seems to suggest that the conversational reference points are set up retrospectively, or else that some notion of "potential reference time" is needed, or some combination of these two modifications. In such cases, selecting one of the two candidates for reference time sometimes eliminates the other candidate from later use, and sometimes does not. Consider a dialogue consisting of (129) followed by (131). The other reference point ("I was at university") seems to be still available for use in the last question.

(131)

A : Yes.

B : What other exciting events occurred ?

However, if we append (132) to (130), the resulting dialogue does not flow smoothly.

(132)

A : Nothing much.

B : Was there a big reception ?

The ducal visit seems to have been lost as a possible reference point in this case. This may be describable, like the previous set of examples, in terms of conversational structure divided into subsections. It may be that in the well-formed dialogue (129)+(131), the centre question and answer constitute a nested subsection of conversation using a different reference time; the previously-established reference time is saved for use at the outer level of conversation. The slightly odd dialogue ((130)+(132)) may result from there being no embedded section where a different reference time can be used. These comments are lamentably vague. One possible way to formalise this description might be to have a more precise notion of a subsection of conversation (e.g. as in Sections III.12, IV.7 above), with the slots PRESENT and REMOTE being local to each subsection.

Alternatively, it may be that the tense slots are not single slots, but are more like push-down stores, where items can be heaped up in a "last-in-first-out" basis. Finding the relevant reference time for an utterance may consist of searching back down the stack

for an appropriate reference time, removing all potential reference points above it. That is, every tensed clause in the dialogue causes a potential reference point to be placed on the appropriate tense stack, but once a reference time is adopted, it causes later entries on the stack to be lost.

These two "solutions" are not just alternative re-wordings of the problem, but differ in their effects. If we segment conversations into nested subsections (e.g. as described in Sections III.12, IV.7), then we may wish to associate various characteristics with each section. For example, each subsection might achieve some particular purpose in the dialogue, or there might be a whole range of "local" information (other than just tense settings) which is specific to each subsection. To say that the tense settings are determined by the successive sections (nested or sequential) is to say that the patterns which occur in these other characteristics follow the same behaviour as the tense patterns. On the other hand, if we just say that the tense values are stacked and unstacked, it says nothing about the way that such storing will relate to other aspects of the dialogue.

There seems to be a very low limit on how many reference times can remain accessible, whether the "stack" or the "subsection" approach is used. It is quite difficult to construct a fluent dialogue where the speakers use a succession of reference times, and then re-use previously mentioned reference times. The following exchange has used five different reference times, at the point marked -> . The subsequent utterances attempt to re-use all of these (in a last-in, first-out basis). The last utterance (*) seems to be

slightly odd - a puzzled comment such as (134) seems as appropriate.

(133)

A : While I was a student, I lived for some time in France.

B : Did you visit Paris ?

A : Yes.

B : Did you go to the Eiffel Tower ?

A : Yes, we had a picnic at the top.

->

B : What other towns did you visit ?

A : Bordeaux.

B : Where else did you live ?

A : Italy, for a few months.

(134)

A : What, when I was a student, you mean ?

Although the above discussion has been based on past tense examples, most of these patterns appear in present tense sentences as well. For example, the following dialogue uses two present tense time references, with the "storing" effect described above.

(135)

A : When he is staying in London, his relatives will visit him.

B : Will they annoy him ?

A : No.

B : Will he be working during the day ?

If a retrospective selection of time-references is used, then this provides another analogy between the behaviour of tense markers and that of pronouns (cf. McCawley (1971b)).

V.7.3 The Function of Time-binders

Isard (1974) suggests that most of the systematic allocation of reference times inside a sentence can be performed by the "time-binder" word "when" in the subordinate clause. That is, executing the semantic part of "when" does the following in his program :

(136)

(a) locates the event time of the subordinate clause

(b) sets this event time into the appropriate slot (PRESENT or REMOTE)

The interpretation of the main clause then uses, for its reference time, whatever has been loaded into the slot that its own tense refers to. There are slight problems with this analysis. Firstly, there are other tense manipulations necessary within the sentence; secondly, it may be inappropriate to have all the

manipulations performed by the time-binder.

Compare (137) and (138) :

(137)

(a) Before I arrived he sent off the letter.

(b) When I was opening the window, he left the room.

(138)

(a) At five o'clock, he left the room.

(b) On Tuesday, he sent off the letter.

(c) Yesterday, all my troubles seemed so far away.

Although time adjuncts in the form of a clause contain a time-binder word (which can be given the responsibility of setting up the reference time) there are very similar sentences (e.g.(138)) where there is no single time-binder to which this task can be allotted. These sentences may need a separate SCR for time adjuncts, which takes the meaning of the adjunct (a time structure, whatever that will look like) and slots it into an appropriate tense slot. Thus, the time-binder can be relieved of the task of setting the tense slot up - it has just to operate on the clause (or its meaning) to produce a time structure for the SCR to act on.

In the Isard program, the meaning of "when" operates on the meaning of a clause to produce a reference time - namely, the time of the event referred to by the clause. The difference between "when" and other time-binders like "before" and "after" seems to be in the

structure that this conversion produces. Compare the sentences in (139).

(139)

(a) When the sun was shining, I could see Ben Nevis.

(b) Before the sun was shining, I could see Ben Nevis.

(c) After the sun was shining, I could see Ben Nevis.

The relationship expressed, by these sentences, between the time of one state and another differs as the time-binder is altered. Rather than postulate that the time-binders operate on two clauses, it might be better to have the different time-binders construct different reference times from the subordinate clause. Once this time-structure is inserted in the appropriate slot, the procedure for interpreting the main clause can act differently on the different structures. If this conversion from clause-meaning to time-structure is the task of the time-binder, then "when" may, in some sense, be the most basic example. Whereas other time-binders may operate on the event or state referred to in the clause to produce some time structure not coincident with the event or state, the meaning of "when" generally produces, as reference time, the duration of the event or state (approximately). Hence the "approximate coincidence" relationship mentioned above is a direct result of using "when" as the time-binder. If we set up an event or state as the reference time without using a time-binder - i.e, implicitly in a dialogue - then the effect is very similar to using "when". The time-relationship conveyed in (140) seems closer to that conveyed in (141) than either of those in (142).

(140)

A : Did you cook the meat ?

B : Yes. The smell was overpowering.

(141) When I cooked the meat, the smell was overpowering.

(142)

(a) After I cooked the meat, the smell was overpowering.

(b) Before I cooked the meat, the smell was overpowering.

(This is a very weak point - intuitions vary greatly). This would fit in quite well with the description of "when" as a regular "wh"-word. A clause beginning with "wh-" ("what", "who", etc.) can be regarded as a pattern to describe some item, where the "wh"-word conveys the semantic class of item (time, thing, person, etc) and the clause gives a partial description of it. Thus "when I cooked the meat" is roughly paraphraseable as "(at) the time at which I cooked the meat". This is just how Isard uses "when" clauses - they are patterns describing times. Hence, if we postulate that the meaning of "when" merely converts the meaning of a clause into a pattern for a time, then we are saying no more than that "when" is the "wh"-word for time.

V.7.4 Disembodied Time References

Under the Isard analysis, a "when" clause causes the value of the tense slot to change, but in some sense the "meaning" of the "when"- clause is not itself acted upon by a higher grammatical rule. (In computational terms, the clause has side-effects but leaves no

results). Under the analysis suggested here, the clause passes up a result (a time structure) for a SCR to use. Hence the analysis of partial utterances in exchanges like (143) changes.

(143)

(a) Had you taken two ?

(b) Under what circumstances ?

(c) When you took eight.

Isard's program tries to understand (a), and finds it has nothing in its REMOTE slot; it therefore asks for a time reference in (b). The clause (c) is interpreted and executed, and the program attempts once again to interpret the initial utterance (a); the execution of the "when"-clause should have set the REMOTE slot accordingly. Under the analysis here, (c) would merely present the program with a ready-made reference point - the program would have to "know" what to do with it (i.e.use it to fill the "REMOTE" slot), and it could then continue from where it left off. This distinction may seem fairly minor, but the new approach will allow (144)(b) to be treated similarly to (143)(c).

(144)

(a) When did you see me ?

(b) When you left the building.

That is, uttering an isolated "when"-clause is regarded as offering a time-structure to the hearer - the way that the hearer reacts to this will depend on what he is expecting. An alternative which could be adopted under the Isard analysis is to suggest that the act of setting a value into the tense slot can be regarded as answering a question (e.g. in (144)), but this seems slightly counter-intuitive.

This is not to say that the time-structures created for the "when"-clause does not end up in one of the tense slots; the point is that the clause meaning itself (or the time-binder "when") should not perform this task. If we allocate such decisions to rules outside the "when"-clause, then (143)(c) and (144)(b) can be treated identically at the clause level, but any differences can be described at the conversational level, in that the time structure is being used in different ways by different conversational routines (cf."localised semantic description" in Section IV.8).

V.7.5 Tense Clash

Once a "when"-clause has set up a reference time, the main clause has to use it, or oddity results :

(145) * When he walks in, I greeted him.

This is not a constraint on keeping tenses in different clauses of a sentence the same :

(146)

(a) Although she sells sea shells, she was once a frogperson.

(b) Because she helped Fred, he is very grateful.

(c) Fred said that Gladys knows that Evelyn left.

If the time-binder (under Isard's analysis) or the time-adjunct rule (under this analysis) merely alters the value of the tense slot there is no reason why the main clause should have to use this slot. There seems no neat way of accounting for this constraint, but it may be that within each clause there is some notion of "current tense-slot". That is, one of the two slots is selected, on the basis of time-adjunct and tense information in the sentence, and that is the reference time for that clause. The difficulty (in sentences like (145)) is then not because the two clauses have different "current tense-slots", but because the process of interpreting the time-adjunct selects the tense-slot for the main clause using the reference time of the subordinate clause; the main clause itself tries to select a tense-slot using its tense. If these are different attempted selections (as in (145)), a clash results. This is ad-hoc, but it is not clear what else would fit in with the other suggestions.

V.7.6 A Summary of the Tense-Slot system

Let us summarise the modifications to the Isard/ Longuet-Higgins analysis. The task of the time-binder words is solely to convert a clause-meaning into a time structure. This time structure can then be used as the answer to a question, or may become the argument for a SCR which handles time adjuncts. The basic form of time-adjunct is the "when"-clause, which indicates a moment or interval of time using

an event or state which has roughly that duration. Time adjuncts may occur in forms other than bound clauses (e.g. prepositional phrases) but are handled in a similar way. The time-adjunct SCR rule selects the tense- slot indicated by the time adjunct to be the "current" tense-slot for the main clause, and puts the time-structure supplied by the tense of the time adjunct into that slot. In interpreting the main clause, a selection is made of a "current" tense-slot on the basis of the clause's tense, and the interpretation proceeds using the reference time held in that slot (or perhaps by searching a stack of stored values - see above).

V.7.7 Time Semantic Structures and Relationships

Before discussing the details of the various semantic categories and SCRs needed to describe such structures, let us examine informally the kind of information that will have to be represented.

The way in which an event can be used to convey a time structure can vary. For example, (147) seems to refer to a definite point in time, whereas (148) seems to describe an interval of time :

(147) When I opened the box .....

(148) When I was opening the box .....

This distinction affects the way in which a main clause is interpreted; if we append (149) to (147) the interpretation is that the leaving followed the opening, but if it is appended to (148), the interpretation is that the leaving occurred at some stage during the opening.

(149) ....he left.

If we assume that events and states are located on a simple "time-line", the appropriate primitives seem to be "points" and "intervals". (These are left as unanalysed primitives, whose mnemonic names should be intuitively useful). One way to describe an interval of time is to mention a state which existed during that period :

(150) When I was at school .....

The verb "be", together with a complement is a simple way to achieve this. As illustrated in (148), the use of a progressive verb form can also describe an interval of time. (Some generalisation might be possible about the meaning of copular "be" and progressive "be", regarding its use for describing states of affairs; this will not be attempted here). If a sentence attempts to relate two intervals (in the way described above) the interpretation is often that the state of affairs described in the "when"-clause held for a sub-interval of the duration of the state described in the main clause :

(151) When I was at school, de Gaulle was still alive.

Some examples can be found where there seems to be a slight implication that the two time intervals are co-extensive -

(152) When the sun was shining, the room was quite warm.

However, it is rather difficult to find examples where the main clause describes a sub-interval (proper) of the interval indicated by the "when"-clause - the reverse use, as in (151) seems more natural.

It may be that the suggestion of co-extensiveness that in sentences like (152) derives from some kind of "conversational implication". The sentence just states that the state of affairs described in the main clause existed while that in the "when" clause did - i.e. a sub-interval relationship, as in (151). The feeling that the state of affairs did not hold outside the interval described by the "when"-clause may be a deduction from the fact that there must be some point to saying (152). If we add "...And when the sun went down we turned on the heater to keep it that way" to (152), there is no contradiction; (152) does not assert that the state of affairs described in the main clause failed to hold outside the smaller interval.

Point reference times can be described by the non-progressive forms of certain verbs, as in (148). As observed above, relating two point reference times leads to some interpretation of "approximate coincidence" (see (127)). Here is a case where "world knowledge" is necessary to sort out the exact relationship between the two events - the syntax/semantics can only provide the "approximate coincidence" relationship. This vague relationship seems to be paraphraseable as "just before, during, or just after", where there is no absolute definition of "just after " or of "just before"; these depend on the "scale" of the event, in some sense :

(153)

(a) When the bell rang, I opened the door.

(b) When the Americans dropped the atomic bombs, Japan surrendered.

It may seem rather glib to keep allotting various problems to "world knowledge" and "higher-level inferences", but there are limits to what should be crammed into the grammar. One further non-grammatical aspect of (153) is the slight implication of causality that some people may extract from sentences like (153)(b). However, this nuance does not come directly from the use of "when", but rather from the juxtaposition of the mentions of the events. Compare (154), which conveys a similar sense of causality (or fails to, according to your intuitions).

(154) The Americans dropped the atomic bombs. Japan surrendered.

The use of the "when"-clause does help to draw attention to the temporal proximity of the two events, and hence may spark off some deductions in the hearer's mind, but there is no need to cram these inferences into the grammar of time clauses.

If the sentence describes a point event and an interval the relationship between the two seems to be the same no matter which occurs in the "when"-clause :

(155)

(a) When I was quite young, Fred hit me with a shovel.

(b) When Fred hit me with a shovel, I was quite young.

(There is no suggestion that these two sentences are synonymous - merely that the time relationships expressed in them are the same). As pointed out in (148) and (26), there are two ways of using an event to refer to a reference time - either as a point or as an interval. As in (127)(b), two point events can be related by being coincident :

(127)

(b) When they built the fifth bridge, they used the best materials.

If we wish to state that one event occurred at some stage during another event, describing them as point events is awkward; it seems more natural to describe the "surrounding" event as an interval :

(156)

(a) ? When they built the fifth bridge, a workman drowned in the river.

(b) When they were building the fifth bridge, a workman drowned in the river.

Also, if both clauses describe events as "points", they cannot be interpreted as locating the event described by the when-clause during the event described in the main clause :

(157) ?? When he put in salt, he cooked the meat.

Thus there are certain possible ways of relating time-points and time-intervals using "when"-clauses;

"When"-clause	main clause	relationship
point x	point y	x approx.coincides with y
point x	interval i	x contained in i
interval i	point x	x contained in i
interval i	interval j	i a subinterval of j

The ways in which point and interval time structures can be conveyed by different verb forms is quite complex (or even messy). As commented already, an interval can be described by either

(158)

(a) the copular "be"

(b) the progressive form of a verb

Certain other verbs can also serve to describe states of affairs, and produce interval time structures without the use of progressive verbs. Stative verbs, like "believe", for example :

(159) When I believed in deep structure, life was much simpler.

This does not seem too odd, since stative verbs usually do not have a progressive form and seem to already contain the sense normally associated with progressive verbs. However, there seem to be verbs which are semi-stative, in that they have a progressive form, they describe a state of affairs even in non-progressive form, and they fail some of the standard syntactic tests for stative verbs. "Live", in the sense of "dwell" or "reside" is one such verb. The

oddity of (160)(a) suggests that this is a stative verb, but the acceptability of (b) and (c) suggest that it may not be:

(160)

(a) ?? I lived in London and he did so too.

(b) What he did was to live in London.

(c) Live outside London - the rents are lower.

The two sentences in (161) seem almost synonymous; in particular, the time relationships conveyed are the same.

(161)

(a) When he was living in London, the rents were high.

(b) When he lived in London, the rents were high.

If we try to use the "when"-clauses of (161) to describe an interval within which some event occurred, the two verb forms are not equally suitable :

(162)

(a) When he was Living in London, someone bombed the GPO Tower.

(b) ? When he Lived in London, someone bombed the GPO Tower.

As pointed out ((147) and (148)), the progressive form of a verb is more suitable for describing an interval during which some point event occurred. It may be that (162)(b) sounds odd because of its similarity to the completely non-stative, non-progressive verb form (cf.(156)), whereas (162)(a) sounds better because of some similarity

to the standard form. The whole issue seems very messy.

So far, the only time structure associated with totally non-stative, non-progressive verbs has been the simple point reference time. If we now look at the use of such verbs to describe habitual events, further complications arise. Roughly speaking, a habitual occurrence can be regarded as a series of events which together define an interval (namely, the interval within which they occurred). Thus a clause like:

(163) When I travelled in by tube .....

is ambiguous between (at least) 2 possibilities :

(164)

(a) it refers to a single event; consider adding to (163) the clause : "...I left my briefcase behind"

(b) it refers to an interval during which the event occurred several times; consider adding to (163) the clause "...I was able to claim expenses"

This may be the only way in which a non-progressive, non-stative verb can describe an interval - by describing some habitual action which took place within that interval. An interesting consequence of this is the way in which the ambiguity of (163) is preserved even when a progressive marker is added :

(165) When I was travelling in by tube....

(a)...Someone stole my briefcase. (b)...They raised the fares.

There seems to be no grammatical way of disambiguating these - again some resort is necessary to extra-linguistic knowledge.

V.7.8 Perfect Aspect

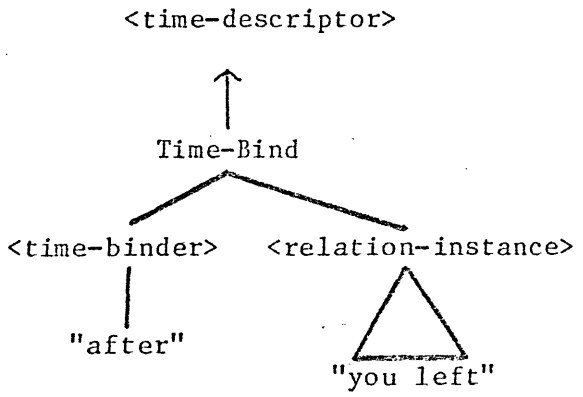
As mentioned previously, the use of perfect aspect generally indicates that the event time of a clause precedes the reference time of the clause. This information can be used in two ways - if the reference time is known, it locates the event time; if the event time is known, it helps to define the reference time. The rules for determining whether to use a verb with perfect aspect, rather than a past tensed verb, (i.e. "I have seen him" rather than "I saw him") are unclear. One relevant factor may be a reluctance (or inability) on the part of the speaker to specify an exact reference time at which the event occurred. This alone would not account for all the nuances that the present perfect seems to convey, but it is outside the scope of this chapter to examine this wider problem.(McCawley (1971b) gives an inconclusive discussion of several examples).

There has not been time to develop a full analysis of perfect aspect here, as it is probably the most difficult part of the English tense/aspect system. The MCHINE program (Chapter VI) includes some ad hoc devices to allow it to respond to questions which use perfect aspect.

V.7.9 Some Rules and Structures

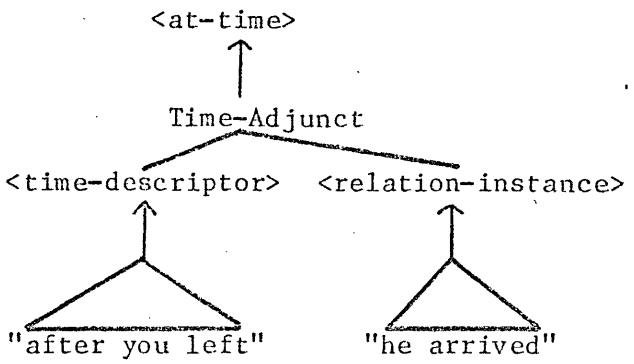
Let us examine in slightly more detail how the rules and structures could operate for time clauses. As pointed out above, it seems plausible to regard time-binder words like "after" and "before" as combining with a clause-meaning to form some kind of "time-descriptor". That is, we could have two semantic categories - "time-binders" and "time-descriptors" - and an SCR (say, "Time-Bind") which performs the combination. Surface trees like (166) would be the result:

(166)



In order to relate these time-descriptors to another clause, there would have to be a further SCR (say, "Time-Adjunct") which relates adjuncts to main clauses. Trees like (167) would follow, where the semantic category of the overall result is yet to be discussed (see below).

(167)



By separating these two functions into separate SCRs, we allow other time-descriptors (e.g. (138) above) to combine with main clauses in the same way, and time-descriptors to be created independently, as in (168)(b).

(168)

(a) When did he arrive ?

(b) After you left.

The overall structure produced by the Time-Adjunct SCR is hard to characterise. It seems to express a relationship between a time and an action, or state of affairs. We can represent this (in a rather ad hoc fashion) with a semantic network relation "AT-TIME".

Let us consider what characteristics a "time-descriptor" will require to represent the time relationships discussed above. (i.e. in the table in V.7.7). The first important point is that a time-descriptor is at the level of intensional semantics (see Section III.11), not referential semantics. The "points" and "intervals" used in V.7.7 above are abstract constructs, which enter into the relationships sketched here; they are distinct from "points" and "intervals" in time. Time-descriptors <u>denote</u> segments of the

traditional "time-line", but they are at a different level of description. The duration of a particular event may define one segment of time, but it may have several different linguistic descriptions, with these different descriptions behaving in different ways (cf. (147) and (148)). It might seem that the minimum amount of information that a time-descriptor would need is the segment of time involved, plus an indication of whether it is to be treated as a "point" or an "interval". If we consider the way that time-clauses are used, it turns out that this is not the appropriate kind of information. Although the point/interval indication is necessary if we are to systematise time relationships as above, the exact times involved may be totally unknown to speaker and/or hearer. Time-clauses serve to locate events or situations relative to one another, not with respect to some absolute time-line. A time-descriptor should therefore contain a representation of the event (or situation) involved, and an indication of how the start and end of the "point" or "interval" are defined by the start and end of the event. For example, in a clause like (169), the end point of the event provides the start of the interval described.

(169) After he arrived.

In all the time clauses examined so far, the "points" and "intervals" can be described directly in terms of the start and finish of the event or situation, with the further option of being "undefined". Hence a time-descriptor can be represented fairly simply, containing just a description of the reference event, a point/interval marker, and an indication of how the event's end-points define the end-points of the time described. The SCR

Time-Bind will act on a clause (which contains an event description) and a time-binder (which will provide some of the information to construct the other markings), to produce a time-descriptor.

V.7.10 "When" Clauses

One obvious way to describe "when" clauses would be to class "when" as a time-binder, along with "after" and "before". That would not capture certain other patterns concerning "when". In particular, "when" is a "wh"-word (see Section V.4 above), and it would be neater if clauses like (168)(a) could be processed by the general wh-clause grammar, as in Section V.4. Also, it might be difficult to process question "when"-clauses (e.g.(168)(a)), answer "when"-clauses (e.g. (143)(c)), and subordinate "when"-clauses (e.g.(147)) all in the same way, if the latter are to be described as time-binder + clause combinations, since (168)(a) does not seem to produce a time-descriptor.

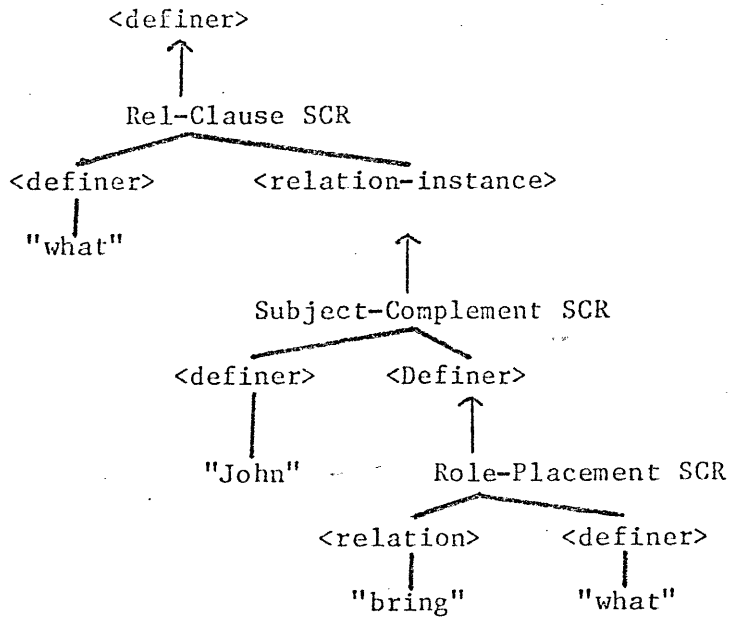
As discussed in V.4, wh-clauses can be looked on as a pattern which identifies some item (or items), and this pattern can be used to insert a reference to that item into a sentence (via a relative clause) or to seek more information about that item (via a wh-question). "When"-clauses fit into this description, since most subordinate "when"-clauses can be regarded as relative clauses, with a paraphrase using "at the time at which" instead of "when". Questions (e.g.(168)(a)) also fit the general "wh"-clause system, since they indicate an item (in this case, a time) which the speaker wishes more information about. As pointed out in Section V.4, the meaning of a wh-clause is a semantic structure with one component marked as "blank" in some way. If we are to treat "when"-clauses

similarly, some suitable semantic structure of this general form will have to be devised.

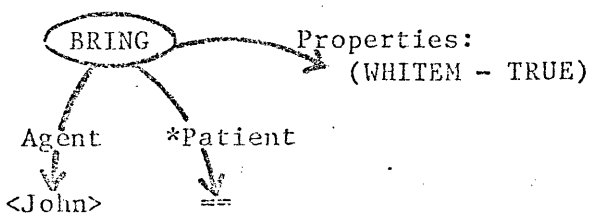
This can be achieved as follows. An independent "what"-clause like (170) is given a surface tree like (171), with a semantic network representation as in (172).

(170) What did John bring ?

(171)



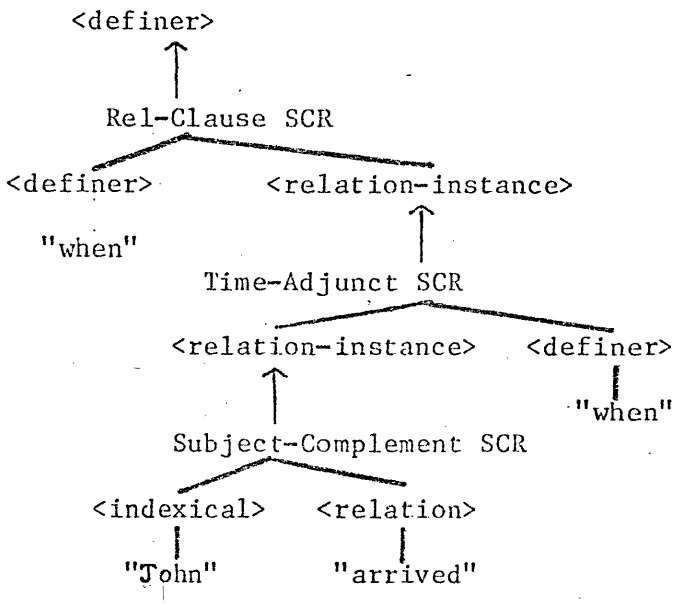
(172)



Similarly, a "when"-clause (173) can be given the structure in (174) for its surface tree.

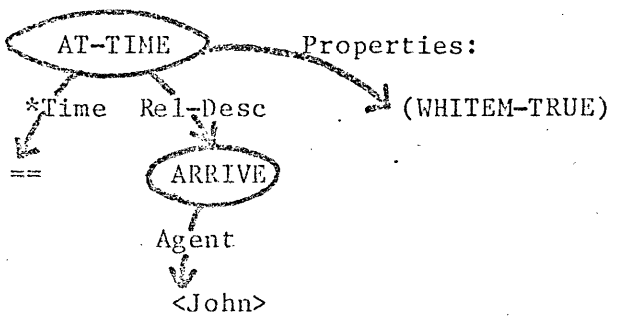
(173) When John arrived.

(174)



That is, "when" is regarded as a "wh" form of time-descriptor, which can thus fill the second argument of the Time-Adjunct SCR, in the same way that the meaning of "what" can fill an argument in a role-placement rule. The resulting AT-TIME relation-instance will have a component marked as a "wh"-form, and the Rel-Clause SCR will work on this as usual to produce a network like (175).

(175)



This means that a "when"-clause will be a definer which singles out the time associated with the state or event described in the main

part of the clause. Thus, this item describes a time, but is not a "time-descriptor" as created by the Time-Bind SCR described above.

V.7.11 Summary of Rules

Let us summarise the representation of time-clauses. There is an SCR Time-Bind which combines a "time-binder" with a relation-instance to form a "time-descriptor". The time descriptor contains the original relation-instance, a "point" or "interval" indicator, and an indication of how its end-points are to be used to delimit the time period. There is an SCR Time-Adjunct which inserts the tense of the adjunct as the tense of the meaning of the main clause, sets this tense as the "current" tense in the current conversational section, and creates an "AT-TIME" relation-instance as its result. "When" clauses are treated as wh-clauses which will produce a definer (in the normal wh-clause way) which indicates the time period of some event. This structure may also act as input to the Time-Adjunct SCR, as may any time-adjunct (whether created by an SCR or already formed as a lexical entry). (Section VI.4.7 discusses a detailed implementation of these suggestions).

Section V.8 : Verbs and Cases

Winograd's syntactic system uses the features SUBJ, OBJ, and AGENT to classify the groups which are directly related to the main verb of a clause. "Subject" and "object" are useful classifications for describing English surface structure, but their semantic relevance might be queried; Lakoff and Ross (1967) (in attacking Chomsky (1965)) claimed that "subject" and "object" cover no semantic generalisations, and are purely a surface classification.

Different surface configurations may be used to express similar semantic relationships; for example, (176)(a) and (b) are very similar in meaning, and (177)(a), (b), (c), and (d) are also.

(176)

(a) John hates Mary.

(b) Mary is hated by John.

(177)

(a) Fred gave Mary the book.

(b) Fred gave the book to Mary.

(c) The book was given to Mary by Fred.

(d) Mary was given the book by Fred.

This could be expressed by associating with each verb a canonical set of object and subject slots, and classing together those sentences which are similar in terms of the items filling these places. That is effectively what Chomsky (1965) did, using

"subcategorisation rules" to describe the slots for main verbs, and having canonical deep structures (to which the subcategorisation rules applied) for superficially different sentences. The "deep structures" for (176)(a) and (b) differed only by the presence or absence of a passive marker - they had the same deep subject and the same deep object. Similarly, all the sentences in (177) had a deep structure in which the deep subject, object and indirect object correspond to the surface subject, object and indirect object of (177)(b). Although Chomsky's subcategorisation rules did not use relations such as "subject" and "object", he pointed out that it might be possible to define terms like these at the level of deep structure. Chomsky's method is sufficient to capture the similarities in the sentences in (176) and (177). We could re-formulate the information by stating that verbs can have an "AGENT", a "PATIENT", and a "GOAL" at some "deep" level, and these may appear in various orderings at the surface, but this is more or less a notational variant of the "Aspects" system. Let us call these deep canonical slots around the verb "roles".

A further elaboration of this description of verb meanings was produced by Fillmore (1968) who made the suggestion that there was a small fixed set of roles (more than just the three corresponding to "subject", "object" and "indirect object") and that certain of these deep roles were semantically similar. For example, there is a role AGENT, which various verbs have, which has some independent semantic content. He hypothesized that there was a small, universal set of roles which are used in deep classification of verbs, such that whenever a certain role appears it always expresses the same relationship between the verb and the term involved. If a term X has

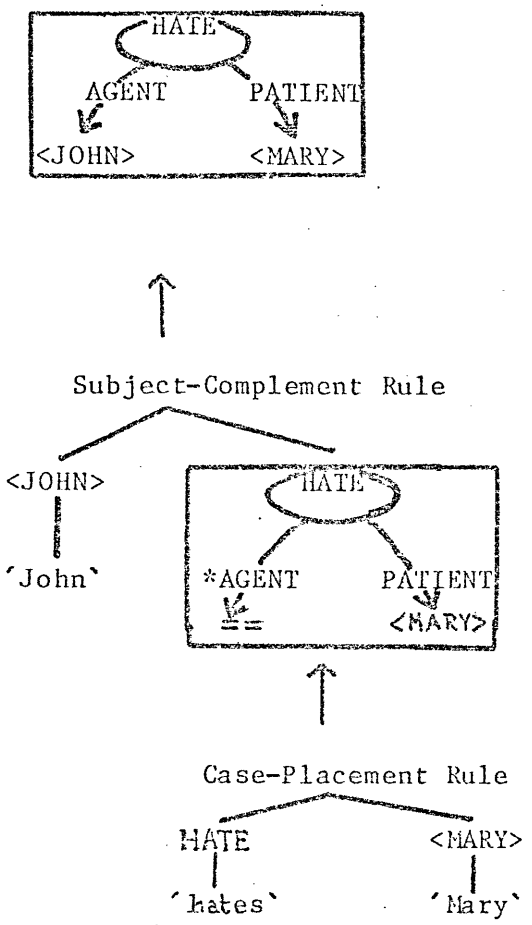
the role of INSTRUMENT in a verb V1 and a term Y has the role of INSTRUMENT in a verb V2, then Fillmore's hypothesis is that, in some sense, X and Y are serving the same semantic function in the structures containing V1 and V2. Fillmore called these semantically relevant roles "cases".

To account for the semantic similarities in sentences like (176) and (177), it is necessary to have some description of the main verb other than its surface behaviour. The most obvious way is to look on the verb as being a relation with a certain number of arguments, where these arguments may appear in a variety of surface configurations. Some rules for showing how these surface configurations are related to the argument places in the relation would also be necessary in a complete description. This is very similar to what Chomsky proposed in 1965, if we avoid the confusion caused by the use of different terminology. Fillmore's main extension was to suggest that there could be some semantically interesting classification of the kind of roles that verbs can have. Some natural language systems in artificial intelligence (see Bruce (1975) for a review) claim to be using a "case grammar" where in fact they are using a notational variant of Chomsky's system. A program does not really use a Fillmore-style system unless it includes some non-trivial categorisation of the roles used in verbs, and uses this categorisation in some way, for example to perform inferences. At least some of the inferences would have to be phrased solely in terms of the "cases" involved, without reference to the particular verbs. If the verbs (or relations) concerned are mentioned at all times, then the semantic content of the role has not been made independent of the verb, and the inference is really referring to "the nth

argument of the relation R" as could be done without cases. "Cases" in mostartificial intelligence language systems are simply mnemonic labels for argument slots.

Verb constructions are handled in the following way in computational grammar. Each verb has an associated semantic relation, and the set of roles for that relation fulfils the purpose of a "case-frame" for that verb (see Section III.10 for more details of the semantic representation). Some of the SCRs (which will be referred to informally as "role-placement rules") take a relation as first argument, and build a "definer" based on a relation-instance using that relation (see Sections III.10 and IV.3). A role-placement rule puts its other arguments (which will come from the object and/or indirect object of the verb into the roles of the relation-instance. The subject-complement rule creates a relation-instance by putting the semantic item from the subject of the clause into the selected role of the definer. Diagrammatically, the SCR tree is like (178) (where the semantic item produced by each SCR in the tree has been sketched in).

(178)



This has several advantages. Firstly, the case frame for the verb need not try to express, in one frame, the necessary deep semantic pattern and the necessary surface syntactic configuration. All the roles are obligatory, in the sense that, in any given instance of a relation, each role will be present, even though the sentence analyser or semantic network processor may not have an item to fill it. This captures the point that if two relations have different sets of possible participants, they are different relations. Fillmore's case frames allowed some cases to be "optional" or to be classed in pairs one of which must occur. This attempted to record, in the deep semantic structure for the verb, information which was concerned with the surface realisation of the

items. In computational grammar, such optionality is covered by the recognition grammar, using the syntactic properties of the verb and the various role-placement rules, as will be described below.

Secondly, if we wish to represent the fact that two verbs have "similar" meanings, there are two ways of doing this. One crude way is to give the two verbs the same relation (with, consequently, the same role-list). Thus two verbs can have the same "meaning", and still differ in their surface behaviours (since that is specified separately). In the MCHINE grammar, "address" and "speak" are handled in this way. If we do not wish to identify the two meanings, we may still represent some degree of similarity by using the facility for re-expressing a relation in expanded (primitive) form. Two verbs can have different non-primitive relations, but have the same primitive relation in their expanded forms. Fillmore tried to capture such similarities (e.g. between "kill" and "die") by simply comparing case frames. As Charniak (quoted in Wilks (1976)) has pointed out, this will show the similarity only if it is assumed that there is some common semantic "core" of meaning paired with the two different case frames. The method here achieves exactly that.

The sentence analyser in the MCHINE grammar processes verbs as follows. On encountering the main verb, it accesses various syntactic properties in the lexical entry, which tell it how many objects there may be, and how to process them. ("Object" is used loosely here to mean "a post-verb constituent whose meaning fills one of the roles in the relation associated with the verb"). This information is conveyed by three lists (any of which may be empty). One list indicates the possible SCRs if no object is present, the

second list contains pairs of SCR and ATN state-name (for single object processing), and the third list contains triples of SCR and two state-names (for double-object configurations). The analyser uses the state-names, if given, to process the object(s), if any, and uses the corresponding SCR given to build a structure round the meaning of the verb. This has the advantage of allowing the individual behaviour of verbs (in terms of how their objects are expressed and how they fit into the verb roles) to be specified separately in the lexicon, rather than itemising every possibility in the grammar (see Section III.9 for further comments). New verbs, with idiosyncratic surface constraints, can be added simply by writing an appropriate lexical entry. Case-placement SCRs do little (in the MCHINE grammar) apart from permuting objects into the deep roles and setting up the selected role (which the subject of the clause, if any, will fill). Prepositions can act as clues to the analyser in processing the objects, since a verb can be marked (using the state-names in its lexical entry) as having a particular kind of prepositional phrase as an object (e.g. "speak" takes an (indirect) object starting with "to"). This illustrates the case-marking function of prepositions to some extent (but see Section VII.8.2 for some difficulties).

Passive constructions are handled quite naturally within this system. The role-placement SCRs are subdivided into "active" and "passive" (no good criterion has been found to characterise these two classes of rules, so at present the grammar writer has to mark the SCRs explicitly). When the analyser encounters a "passive" verb configuration (auxiliary "be" + perfect participle), it sets a restriction on the SCR entry for the verb phrase node, disallowing anything other than a "passive" SCR to be put there. "Passive" SCRs

have to be marked in the lexical entries for verbs, like any other object specification; the grammar writer has to bear in mind that the "agent" phrase (if it is included) has to be treated as an "object" marked with the preposition "by". The construction illustrated in (179) can be included for the verb "address" by including (180) among its single-object specifications in the lexicon, where SCR4 is some suitable role-placement rule marked as "passive", and STPRPBY is the ATN state which "expects" "by".

(179) I was addressed by the chairman.

(180) [SCR4 STPRPBY]

There are several role-placement rules (SCRs) for single-object configurations, and several for two-object configurations. For example, the surface order of the items for the verb-roles is different in (181)(a) and (b).

(181)

(a) He said something to me.

(b) He said to me that you had left.

It might be argued that these could be handled by one SCR, with the analyser re-ordering the surface objects before putting them into the nodes for the SCR. This would have certain drawbacks. Firstly, we could no longer have such a simple, extendible method of specifying the object configurations for a verb, since the grammar would have to incorporate some register manipulation specifically for that verb. Secondly, it may be that the different orderings of the objects has

some semantic content (e.g. concerning "topic" or "focus"); if so, we would want to be able to describe this difference by incorporating it into the different SCRs.

Charniak (1975) has advanced the thesis (also presented, briefly, in this chapter, above) that most artificial intelligence language programs do not at the moment use "case" in the Fillmore sense, and Wilks (1976) has taken up some of the issues in this area. Studying this debate reveals how difficult it is to state exactly what it would mean to be "using case", so it is not very informative (in the present confusion) to state that a particular system does or does not "use case". Charniak would probably regard the MCHINE grammar as not being a case system, but it is worth noting two respects in which the verb-roles are used independently of the verbs.

Firstly, the role-placement SCRs are totally defined (as mentioned above) by the way that they assign items to roles, and are not verb specific in any way. Different verbs can often use the same SCR, if their role-lists are similar. There is, for example, a one-object SCR which puts the object item into the PATIENT role, which can be used in the object-specification for any verb that has PATIENT among its roles. Nevertheless, this does not give the role any independent <u>semantic</u> content, and so could be said not to be a case facility in the strongest sense.

Secondly, in the MCHINE program (Chapter VI), whenever an "event" occurs in the program (i.e. an input or output utterance), the program "remembers" the event by storing a relation-instance representation of it. The "person" who is deemed to have "performed" the action (either "MCHINE" or the one currently marked as

interlocutor) is entered under the "AGENT" role for the event. This is done as a simplified way of setting up the relation-instance, but it could be regarded as attaching some content to the role of "AGENT".

To summarise, the description of verbs given here has several advantages. By having a deep representation of verb roles, it can (as in case grammar) capture semantic similarities of diverse surface forms. By separating the surface specification of a verb from the deep roles, it avoids confusing the possible surface configurations with the set of relationships available at the level of meaning (unlike Fillmore's approach). By allowing decomposition of verb-relations, it allows a further level to show notions of similarity, while retaining the identity of individual verbs. By specifying surface configurations in the lexical entries, it allows the set of verbs to be extended easily. Passive and active constructions are treated in a uniform, general fashion. The structure built around the verb (a definer) is general enough to provide a good interface with the subject-complement SCR (see Section III.10). Checking of selectional restrictions occurs automatically when the semantic items are fitted into the roles around the verb, since semantic relations have role-restrictions as part of the general semantic system.

CHAPTER VI

THE MCHINE PROGRAM

Section VI.1 : Implementation Details

The MCHINE program implements some of the ideas discussed in Chapters III, IV, and V, and has been used largely as a means of investigating the details of these ideas (see Section I.4 for a discussion of this technique). The program can run in two modes - isolated sentence analysis or conversation. In the former mode, the user can type in a single sentence or phrase and the program will try to construct a data structure representing the semantic network (see Section VI.2.7) associated with that sentence or phrase. This structure (which will be the word "NIL" if the analysis has been completely unsuccessful) can then be printed out and examined by the user. In conversational mode, the program can carry on a series of typed exchanges with the user, answering simple questions and responding to statements. (See Section VI.3.10 for more details).

The program is written in POP-2 (Burstall, Popplestone and Collins (1971)) as implemented at the Department of Artificial Intelligence, University of Edinburgh, and runs on the standard DEC System 10 operating system. The compiled code occupies approximately 45K of 36-bit words, above the 11K used by the POP-2 compiler. While running in conversational mode, the program uses additional storage (to "remember" the conversation) and can perform a dialogue of about 20 exchanges in 109K (+11K) of core. The processing time used to analyse an input sentence varies from about 5 seconds for a short simple sentence to about 1 minute for a two-clause sentence. The real response time varies depending on how heavily loaded the DEC System 10 is, but under optimal conditions (typically, 3.00am with no

other users on the machine), conversational replies can vary from 8 seconds to 2 minutes (real time). That is, the program is very slow (see Section VI.5 for further comments).

There is not space here to summarise the POP-2 language, but it should be pointed out that the MCHINE program makes extensive use of certain facilities in the Edinburgh implementation which are not part of the language definition. This is not crucial, since most (if not all) of these facilities can be implemented (and were, previously) using the basic language, albeit more cumbersomely. The most notable facility is the NEWPROPERTY mechanism; the POP-2 system provides a general hash-table which allows the programmer (optionally) to associate arbitrary information with any data item (i.e. a generalisation of the LISP property list).

Section VI.2 : Data-structures and Data-bases

Most of the ideas used to represent information in the MCHINE program are not novel, and have been fully discussed in the artificial intelligence literature. This section merely summarises the methods used, for the purpose of completeness.

VI.2.1 Property Lists

The association-list is a widely-used structure. This consists of a list of pairs, where one member of each pair (the "indicator") represents some field or slot and the other member of the pair (the "value") represents the information present in that slot. This association list, coupled with the NEWPROPERTY system (VI.1.1) provides a completely general property-list system. Incidental pieces of information about some item can be stored on its property-list. For example, the lexical entry for "you" might have the property-list

(DEFINITE.TRUE)(SPECIFIC.TRUE)

to represent the fact that it is definite and specific. (cf.Burstall, Popplestone and Collins (1971, pp.127-132)).

Any entry on an association list can have a "restriction" set on it. When values are entered in an association list, the restriction is examined. If it is non-empty, it is used to test the value being entered. If the test yields false, the value is not entered. Testing the restriction constitutes applying the restriction to the incoming value, if the restriction is a function, and treating the

restriction as a description otherwise (see VI.2.4). Since functions are applied, this facility allows the programmer not only to filter entries to the association list, but also to set "demons" which will be triggered by specific entries being updated. The latter possibility has not been required in the MCHINE program, but the notion of filtering entries to association lists has been very useful.

VI.2.2 Pseudo-Records

Although POP-2 provides a record-definition facility, it incorporates strict run-time type-checking for any record-class that the programmer may define. Hence it is not easy to have record-updating or record-accessing functions which can operate on several classes of records (polymorphic functions). One simple way to allow such sharing of functions is to use association lists to store all information that would normally be kept in records. In the MCHINE program, there is a POP-2 record-class of 3 components, called an "SITEM", and all semantic items in the MCHINE program are represented as SITEMS. The three components are :

Class-word

Serial number

Association list

The serial number is merely a debugging aid. The association list holds all the values that would otherwise be stored in the components of a record. (Cf. Burstall, Popplestone and Collins (1971, pp.218-219). The class-word is a POP-2 word indicating which

pseudo-record class the SITEM belongs to. The class-word acts as an index (via NEWPROPERTY) to any class-specific information that the program may need to associate with the SITEMs. Hence each class can have its own constructing function, its own printing format, etc., accessible by the class-word.

VI.2.3 Contexts

It often happens that the programmer wants to let the values of a particular data-structure vary according to circumstances, maintaining several values without corrupting them. For example, a planning program may have to make hypothetical deductions in some "imaginary" state of the world, without altering any values corresponding to the "real" state of the world. Similarly, a sentence-analyser may have to pursue separate possibilities without letting structure-building associated with one possibility interfere with that of other possibilities. Various mechanisms have been implemented to allow such switching back and forth between values, such as the CONNIVER "context" (McDermott and Sussman, 1972). POP-2 has a counterpart (the "saved-state"), but this is cumbersome and inconvenient for various reasons. A much simpler facility is provided by the POP-2 library program, "CONTXT", designed by Harry Barrow. This mechanism does not store "control" information, and affects exactly those variables and data-structures which the programmer specifies. Also changes in values are recorded only as they occur, so extra space is consumed only as changes occur in a new context, not with the creation of every context.

This facility has been widely used in the MCHINE program - all pseudo-record entries, property-list values, and data-base entries are context-dependent, and the sentence-analyser distinguishes different analysis paths by contexts.

VI.2.4 Matching

The semantic representation system (see VI.3.7) makes extensive use of matching, so the POP-2 library program "ACTOR" (designed by Harry Barrow) has been used to provide simple facilities. Artificial intelligence programs sometimes treat the notion of "matching" in two distinct ways, apparently without realising it. "Matching" usually refers to comparing two items for similarity of structure in some way. For example, (182)(a) matches (182)(b) (where == is a special blank item which matches anything) because they have a similar form. Similarly, (a) matches (c).

(182)

(a) [FRED LIKES BANANAS]

(b) [FRED LIKES ==]

(c) [== LIKES ==]

Artificial intelligence data-bases, in representing information about specific items, often used "matching" as a mechanism to access the information. Many of the pieces of information being represented were relational, analogous to predicates applied to arguments; (183)(a) might represent the predicate logic assertion (183)(b).

(183)

(a) [CLEVER FRED]

(b) |- CLEVER(FRED)

In matching, two items are operated on to produce a truth value; this is vaguely similar to the process of applying a predicate to an argument (e.g.APPLY(FRED, CLEVER) yields TRUE). Somehow, this similarity has led to a blurring of the distinction between the two questions "do X and Y have similar structure?" and "does Y describe the structure of X?". These are different questions - if X and Y are as in (184)(a), the answers are "YES" and "NO", but if X and Y are as in (184)(b), the answers are "NO" and "YES" respectively.

(184)

(a) X : [HAS LENGTH 7], Y: [HAS LENGTH 7]

(b)

X: [ALPHA BETA GAMMA DELTA EPSILON OMEGA ZETA],

Y: [HAS LENGTH 7]

The MCHINE program contains two different procedures EMATCH (for testing similarity of structure) and TRUEOF (for testing whether a description holds).

VI.2.5 Data-base

The MCHINE program uses a semantic network system to represent information about the "state of the world" (see VI.3.10). This is implemented using a very simple data-base system, which is derived from concepts in PLANNER, CONNIVER and the POP-2 library program HBASE.

There is a data-base which can be thought of as a set of "base-slots". Any POP-2 item can be "indexed" by being given a base-slot; a pointer to the base-slot will be stored with the item (using the NEWPROPERTY mechanism), and the base-slot will contain a pointer back to the item. Each base-slot has a "value" which is context-dependent. In any context, the item can be "present" (the value = the item) or absent (the value = UNDEF). Since the base-slot is directly associated with the item, certain operations can be carried out directly without searching the data-base. However, more general queries (e.g. "is there an item present which matches this one ?") do require some form of searching. This is simplified by sectioning the base into sub-bases, each with an associated "index-word". There is a procedure INDEXWD which, for any item, computes the appropriate index word, and hence determines which sub-base is relevant. The class-words of pseudo-records function as index-words for some items, so that some pseudo-record classes have a corresponding sub-base (the exceptions are mentioned in VI.3.7). If INDEXWD produces NIL, this indicates that there is no information to constrain the search, and the whole data-base must be used; this could occur for a general pattern which might match various classes of item. The contents of sub-bases are held in two-way linked lists, to facilitate removal (and garbage collection) of unwanted base-slots, but there is no master-list of the whole base.

Exhaustive searches make use of a POP-2 dynamic list which generates all the base-slots as required.

The data-base imposes no interpretation on "present" or "absent"; in particular, these concepts do not necessarily correspond to "true", "false" or "unknown". It is the task of the semantic network mechanism (VI.3.7) to represent "truth" and "falsity", and the data-base merely provides certain primitive indexing devices for the network program to use.

Section VI.3 : Representing Linguistic Information

VI.3.1 Lexicon

The simplest form that a lexical entry can take is a triple - (<pointer to semantic item>, <list of features>, <property list>).

As there is a small fixed range of features and properties used in the lexicon, both the feature list and the property-list can be coded into a reduced bit-string representation, so that a simple lexical entry takes between 6 and 10 words (excluding any space for the semantic item used). In order to economise further, compound lexical entries are used to take advantage of certain recurring patterns in the entries. One form that a compound lexical entry can take is a pair :

(<lexical entry>, <add-list>)

The add-list specifies properties to be added to the basic lexical entry. Thus, the various forms of a verb can be stored as compound entries, all with the same <lexical entry>, but differing in the <add-list>. The other form for a compound entry is a POP-2 closure function - i.e. a procedure with some of its arguments already fixed. If some entries can be summarised by a rule for constructing them, with all differences captured by a few simple parameters, then it is more economical to store the rule and parameter list than to keep fully constructed entries around all the time.

All these devices were incorporated because of the urgent practical need to save space on the machine, not for any theoretical reasons. No attempt has been made to construct linguistically interesting redundancy rules, although some of the implemented devices might be useful if such a study were undertaken. In particular, some of the generalisations used to save space (and programming time) are expressed in the procedures for putting entries into the lexicon and for accessing them. For example, there is a procedure DEFVB which takes three arguments (semantic item, list of surface forms, property-list of idiosyncratic information) and constructs the various lexical entries for the forms of a verb. The various regularities (e.g. that the perfect participle is the same as the past participle unless otherwise specified) are embodied in this procedure. On the other hand, some generalisations are expressed in the procedures which examine the lexical entries during sentence analysis. For example, each verb entry has a property INFLECTION which (for regular verbs) takes one of 6 values; there are no TENSE or AGREEMENT entries, but the tense and agreement can be computed from the solitary INFLECTION entry.

Accessing of the lexicon is done by a rudimentary hashing system using the POP-2 "MEANING" facility.

VI.3.2 Recognition Rules

The recognition rules are most easily described using the ATN notation (Section II.9) although (as observed in Chapter II) the ATN notation simply provides a graphical representation of a procedure.

States are represented as lists (or strips) of arcs. Arcs are pairs consisting of an arc-pair and a continuation pair. The continuation-pair is either NIL, or else it is a list of either "TO" or "JUMP" followed by a state-name. An arc-pair consists of an arc-head and an action-list. The arc-head specifies the input test for the arc, and an action-list is a list of operations to be performed if the test yields TRUE. Some simplifications are incorporated, both to save space and to make the grammar easier to read. The arc-tests are given in a standard form which displays its internal structure :

<full test list> ::= (NOT)<test list>

<test list> ::= <full test>*

<full test> ::= (NOT)<test>

<test> ::= <test word> <test information>

<test information> ::= <POP-2 list>

<test-word> ::= FEATURE | NEWLEVEL | CATEGORY | PROPERTY | SEMANTIC | WORD | RUN | TEST

This allows an arbitrary logical combination of tests - the optional NOT at the front covers the whole remaining <test list>, and the items of the <test list> are treated as a conjunction, each of which may be negated individually. For example,

[NOT [FEATURE VB] [NOT PROPERTY PREPOS OF]]

yields TRUE if the input word either has not the feature VB or is the

preposition "of". The approximate meanings of the various <test words> are :

FEATURE : <test information> gives a list of features which must be present on lexical entry for input word.

CATEGORY : <test information> gives a POP-2 predicate which must be true of lexical entry for input word.

PROPERTY : <test information> gives a property name and value that must be present on lexical entry for input word

SEMANTIC : <test information> gives a predicate that must be true of the semantic item in the entry for input word.

WORD : <test information> specifies what the input word must be.

RUN : <test information> is a piece of code to be executed, leaving no result, but having side-effects.

TEST : <test information> is a piece of code to test global aspects of the environment.

NEWLEVEL : <test information> gives a state-name to activate at the new level, and a place to put the item found at the new level.

Obviously this set of tests is redundant - all possible testing could be done with just TEST. However, this subdivision makes the content of the tests more obvious to the grammar-writer, and highlights what different types of tests are being used - RUN is a particularly undesirable trick, and should not be allowed to hide.

Similarly, action-lists are organised into a "forward Polish" notation for readability. Although all the ATN structures are packed as economically as possible into records, they are read in as lists to allow easy reading and writing of grammars.

Some states and arcs display common patterns - for example, the state which "expects" the preposition "to" is very similar to the state which "expects" the preposition "by". Such patterns are used to save space by having a general constructing function for such states, with enough parameters to distinguish the states; a closure of this function can be stored instead of the full state. Patterns among arcs can be handled similarly.

VI.3.3 Structural Combining Rules

The surface structure rules are represented in a very straightforward fashion. There is a POP-2 record class "SRULE" which has components for the rule body, input specification, output type specification and (optionally) a list of properties (including a property inheritance rule, if the SCR has one).

VI.3.4 Registers

ATN registers are implemented as data-structures with one context-dependent component. Various operations are provided to perform all the manipulations such as pushing down, clearing, etc.

There are 14 interpreter registers. Some might be termed "control registers", since they keep track of the path through the grammar :

CODETRAC : points in ATN visited so far.

Debugging aid only.

PSTOREGS : Pointer-storing registers active at current level. A stack.

SSTOREGS : Structure-storing registers active at current level. A stack.

CONTLINK : The ATN state to be used on leaving the current level. A stack.

STORACTS : Postponed actions to be done on leaving current level. A stack.

HELDACTS : A slightly underhand device for postponing actions at the current level.

Other registers might be termed "value" registers, since they keep track of the surface structure being built :

TOPNODE : Top node of subtree being worked on at current level. A stack.

TOPNODES : Subtrees so far constructed at this level but not yet joined together. A stack.

SUBROOT : The node to which the subtree being worked on at this level is to be attached. A stack.

CURRNODE : Node currently being worked on.

TREETOP : The overall result of the analysis.

The remaining registers are general-purpose structure-holding registers that any part of the grammar can make use of :

HELDSLOT : Stores the result of a NEWLEVEL temporarily, if the destination of that structure cannot be predicted at the start of the NEWLEVEL. A stack.

TEMPSLOT : General temporary storage, holding one item at a time.

SHELF : General temporary storage, storing items in a last in first-out basis. Not classed as a stack becuse the "push/pop" actions are independent of the NEWLEVEL system.

VI.3.5 The ATN Interpreter

The part of the analyser that scans the ATN grammar (and builds the surface tree) is rather cumbersome, owing to a wish to allow freedom to experiment with various control structures, etc., in the course of developing the program. The interpreter could probably be rewritten more efficiently to perform just its current tasks, eliminating some of the obesity along with the general flexibility.

The ATN is interpreted in the following way. A partial analysis can be represented by a pair (<state-name>, <context>), where <context> provides all the other information via register values. (The state information could be held similarly, but this has not been done). The interpreter maintains a list of such analysis paths, initially comprising one pair - STARSTAT and CUCTXT (both being variables global to the analyser).

The interpreter takes in the next word, expands the current state-name into a list of arcs (see VI.3.2), loops down this list testing each arc. A new context is created from the current one for each arc, so that any side-effects of the arc-tests (an undesirable but occasional occurrence) affect only that path. Each arc which yields TRUE is then "developed". This consists of performing the action-list of the arc, and processing the state-specification at the

end of the arc. If the arc was not a NEWLEVEL arc, the state-specification will either be [TO <state-name>] or [JUMP <state-name>]. If it is the former, this indicates that the given state-name is appropriate for processing the next word, and development of the current arc ceases, after storing the current context and state-name on the analysis list. If the specification is [JUMP <state-name>], the state-name supplies a new current state, and processing continues on the current word.

NEWLEVEL arcs are special in that they directly affect several interpreter registers and pass on the analyser to a new state. The head of a NEWLEVEL arc is of the form

[NEWLEVEL <destination> <state-name>]

e.g.

[NEWLEVEL IN SSBNP0]

This is interpreted to mean : create a newlevel, setting up the SUBROOT as indicated by <destination>, and commence processing in the state given. The destination can be either IN, HOLD or a register name; IN means "item to be attached where CURRNODE now points", HOLD means "item to be stored in HELDSLOT", and a register name means "item to be stored in the given register". (The latter option has not been used anywhere in the implemented grammar). If the state specification at the end of the NEWLEVEL arc is NIL, creating a NEWLEVEL entails attaching the TOPNODE for the current level to the SUBROOT (so that further structure-building at this level can be forgotten) and clearing SUBROOT, CURRNODE, TOPNODE, HELDSLOT and TEMPSLOT. (It might seem that clearing HELDSLOT will be unsafe if

HOLD has been indicated in the arc-head, but it would be unwise to specify HOLD on a NEWLEVEL..NIL arc, as no action-list is added to STORACTS in the NIL case, and so the stored item could not be retrieved from the HELDSLOT. Similarly, attempting to provide an action for storage in a NEWLEVEL...NIL arc is pointless since it will be discarded. These are two places where the ATN interpreter can spot a badly-written grammar and warn the user; there are very few other checkable constraints, unfortunately). If the NEWLEVEL arc has non-null state-specification, then the following operations are performed. The state-specification is pushed on to CONTLINK, the action-list is pushed on to STORACTS, and SSTOREGS, PSTOREGS, HELDACT, SUBROOT, CURRNODE, TOPNODE, TEMPSLOT, and HELDSLOT are all pushed down.

Execution of arcs continues in this way until either the interpreter detects the end of the input string before starting a pass along the analysis list or a POPUP action is encountered. If the interpreter reaches the end of the input string, it sets a variable to indicate this fact and does one more pass along the analysis list. This is because some arcs may indicate options which do not require a word, and which can succeed in the absence of input (e.g. extracting a "wh"-phrase from a register). All arcs which do require an input word automatically yield FALSE on this final path.

Encountering a POPUP action initiates various actions. If processing is already at top level (i.e. there is only one item on the CONTLINK stack), then CONTLINK is popped to provide a new state, and the tree-structure reviewed.

If processing is not at top-level, the interpreter stacks are popped, the registers tidied up, and CONTLINK provides the new state for processing. If no words are left in the input, the interpreter attempts to finish up the analysis path by checking if TOPNODES has accumulated more than one subtree (in which case rule-selection is necessary - see Section III.9) and then putting the current context on the analysis list with state-name STOP. If a POPUP occurs at top level when there are more words in the input, a function called BOTTOMUP is called to find a new state for processing; this implements, very crudely, the "restart" system of Section III.9.

Each analysis path has a TENSION value which holds a value between 0 and 100. Since this variable is global within the analysis-path, it can be altered by any operation, but in fact has been used only in the operation of the SCRs (see VI.3.6). At the end of the analysis, the analysis paths whose TENSION value is the lowest are selected as the final versions. This provides a "weighting"(Woods (1970)) or "preference"(Wilks (1973)), but allows ambiguity in that more than one analysis path may have the same TENSION value.

There is a variable FAIL, which is initially set to IDENTFN, the POP-2 null function. Many of the procedures in the ATN interpreter and the MCHINE grammar call FAIL if something goes wrong, and FAIL can be reset locally (using the POP-2 dynamic binding/ local variable regime) to be some appropriate failure action. When arcs are being processed, FAIL is set to be a JUMPOUT function which will abort the processing of the current arc.

VI.3.6 Surface Structure

The tree of rules and arguments is represented using a pseudo-record class SNODE, which is used for nodes. As well as the information which holds the tree-structuring, each SNODE has the following components :

SRULE : The SCR associated with the node

INTVAL : Semantic item prior to reference evaluation.

EXTVAL : Semantic item after reference evaluation

STRINGFM : Words associated with this structure (debugging aid)

The property-list of each SNODE can hold any optional information, but in fact has been used only for certain syntactic properties (e.g. verb-agreement markings).

There is a range of procedures for tree-building and manipulating, the exact details of which are not relevant here.

Each processing level has a node-pointer in TOPNODE, indicating the subtree being worked on, and a node pointer in SUBROOT, indicating where on the main tree this node is to be attached. These two pointers indicate different data-structures (so that temporary modifications may be made to the TOPNODE before attaching the final version) but are logically the same point in the tree. On leaving a level (either through a POPUP or a NEWLEVEL...NIL arc) the TOPNODE is always merged with the SUBROOT.

The analyser builds the tree in strict left-to-right order, depth-first fashion ("depth" of tree, not of network path search). Any alteration of this order must be achieved by holding sub-structures in registers until they are ready for attaching. The current node (in CURRNODE) can therefore always be recomputed (e.g. after popping up from a level) since it is the leftmost blank or "dummy" (see below) node.

There are certain operations which insert an SCR as a node and use the SCR details to fill out the node. This is done by constructing "dummy" daughter nodes, one for each input place in the SCR. A "dummy" node contains no values, but has various restrictions imposed on the components, using the input specifications from the dominating SCR. Restrictions can also be set explicitly on the components and properties of a node by the action-list of an ATN arc, and this is one way that syntagmatic information (e.g. verb-agreement) is conveyed. If the analyser ever attempts to enter a value which does not meet the restriction specified (if any), the procedure FAIL is executed (see VI.3.5).

Application of the SCRs works in the following way. At any intermediate stage of the analysis, the analyser can attempt to apply an SCR to its arguments (i.e. the values on the daughter nodes). If the daughter nodes have SCRs, these are also applied, and so on down the tree. If a "dummy" or blank node is encountered, the SCR cannot be applied at present and neither can any of the SCRs in dominating nodes, so an "unsuccessful" signal is passed back up the tree. If a node is found, during the evaluation process, whose SCR has already been applied, that SCR is not re-rerun. At the end of the analysis,

the topmost node's SCR is applied, and the same recursive application occurs down the tree; in this case, an "unsuccessful" signal will cause FAIL to be executed, since this is the last opportunity that there will be to run the SCRs. When applying SCRs, reference-evaluating and making entries in EXTVAL (all of which occur at the same point), FAIL is locally set to be a procedure which increments TENSION, rather than one which aborts the analysis completely. (This crude measure allows "semantic anomaly" to cause less havoc than "syntactic anomaly", which may achieve a similar effect to Wilks' "preference semantics" (see Sections II.6, III.6).

When an SCR is applied, its result is first stored in the INTVAL of the node, and then examined to see whether it may require reference-evaluation. If so, this is performed and the result is put in EXTVAL; otherwise, the item is merely inserted in EXTVAL itself. The SCR in the mother node will then take the EXTVAL entry as an argument.

If the analyser finds it is at top-level (i.e. its SUBROOT stack has only one entry left), with no dummy nodes left to build on, and more words to process, it calls a procedure NEWTREE. This stores the current topmost node (in SUBROOT) on the TOPNODES list, and creates a new set of nodes (TOPNODE, CURRNODE, and SUBROOT) to start building a new subtree. (This normally occurs in conjunction with the BOTTOMUP process - see VI.3.5). When the analyser completes a sentence, it checks to see if TOPNODES is non-NIL; if so, it tries to select an SCR which could combine the EXTVAL entries of the nodes on TOPNODES (see Section III.9). If successful, it forms a new tree by joining the list of TOPNODES as daughters to a node with this SCR.

VI.3.7 Semantic Networks

The ideas discussed in III.10 are implemented using POP-2 records and pseudo-records. Each "relation" is a record of 5 components :

RELNAME : print-name of relation

RELROLES : list of names of roles

RELBODY : expanded form of relation

RELELAB : elaborated form of relation

RELRESTS : restrictions on values for roles

There is a POP-2 pseudo-record class of type RELINST ("relation-instance") for representing pieces of semantic network; its components include COREREL (whose value should be a RELATION record) and the various rolenames of its COREREL. That is, a RELINST has a relation plus an association list which binds other items to the roles for that relation, e.g.

(RELINST 30 (COREREL.LIKE) (AGENT.JOHN) (PATIENT.MARY))

The property list of a RELINST can hold miscellaneous information e.g. indexing for the data-base. These structures can be used to form a semantic network which represents a "world model", using two mechanisms. Firstly, a RELINST can be given a component TVALUE which can be filled in with TRUE or FALSE; secondly, the RELINST can be made PRESENT or ABSENT in the data-base. Since both the value of TVALUE and presence/absence are context-dependent, this allows a very flexible system. There are certain standard procedures such as ASSERT, DENY, FINDIF for updating and examining the network.

The RELBODY of a RELATION is a RELINST containing some other COREREL, with markers to show how the arguments of the main relation should fit into the roles in the associated RELINST. This is to allow the expression of "non-primitive" RELATIONs as a configuration of some "primitive" relations. "Primitive" relations are those which have NIL as the RELBODY. When doing an ASSERT, DENY, or FINDIF, this expanded form can be used as well as the main relation. This expansion is optional, and can be controlled by the programmer setting certain variables to 0 or 1; expansion can be used in ASSERT and DENY, and/or in FINDIF, or in none of them.

The RELELAB of a relation is a list of triples of the form (<expression> <operator name> <expression>) and is used to give a procedural version of the relation (if needed). When the relation (with some or all of its roles filled) is "elaborated", all the triples are evaluated in one of two possible modes. In testing mode, the evaluation is intended to yield a truth-value (thus providing an elaborate FINDIF); in updating mode, the evaluation is intended to affect the semantic network (thus providing an elaborated ASSERT or DENY). Each <operator name> is associated with 2 operations - one to be used in testing mode, the other to be used in updating mode. In addition, there is a variable TRUTH which will contain the value TRUE when ASSERT is being executed, and FALSE when DENY is being executed. Thus update mode can cover two separate actions if the elaborated form makes appropriate use of TRUTH in its manipulations. Hence the same triple can, if necessary, be used differently in three different cases, while still representing the same "item of meaning". For example, the relation "BELIEVE" is defined in the program by an entry

```
RELATION BELIEVE [ENTITY ENTITY] AGENT PATIENT;

[ ]

[

[PERSVIEW($:AGENT) ->-> [SRCHCTXT]

[TRUE] =-> [FACTVAL($:PATIENT) ]

] ;
```

which indicates the following. The relation has two roles ("AGENT" and "PATIENT"), with restrictions to ENTITY for both roles. There is no expanded form (indicated by the empty brackets [ ]), but there is an elaborated form, with two operations listed in it. The operator (->->) in the first line indicates that, whether testing or updating, the AGENT's personal context is to be set as the SRCHCTXT ("search context"). The operator in the second line (=->) indicates that when testing, the two arguments should be tested for equality; when updating, the first should be assigned to the second. FACTVAL is a function which accesses the "truth-value" of a statement in a PERSON's context.

Elaboration can also be turned on and off by the programmer setting various variables to 0 or 1.

"Definers", as described in Section III.10, are simply pseudo-records with components RISTRUCT and SLOTNAME, ("relation-instance-structure" and "slot-name"). In addition, a component ROLEGAPS (a list of the roles in the relinst which are not yet filled) may be included for management purposes. As mentioned in

VI.2.5, pseudo-records are generally indexed in the data-base under their class-name. However, this approach would not give a very fine categorisation for the semantic network, as most of the structures are in the form of RELINSTS. Therefore, RELINSTs are indexed under their RELNAME, and DEFINERS are indexed under the RELNAME of the RISTRUCT.

Referring expressions are more neatly described if we look on relations as holding between sets of elements, where a set may be characterised other than by listing its elements. This facility has been included in the semantic network system, in that there is a pseudo-record class SET, and entries in the roles of a RELINST are always SETs. A set can be characterised by listing its members or by including a definer which acts as a characteristic predicate. There are various manipulative functions defined on these SETs to allow the network to use them. Unfortunately, the scope of the MCHINE program did not reach a stage where the SET mechanism was fully utilised, and it cannot be regarded as validated. In most, if not all, of the examples that the program handled, some much simpler representation could have been used equally successfully. It seems very likely that the SET system as implemented contains logical defi ciencies.

VI.3.8 Output Translation

There is no production grammar in the MCHINE program. In its conversations, the range of output utterances is extremely limited, and these are handled by a small routine which replaces semantic network structures by surface strings in an ad hoc fashion. The only items which are not translated in a one-to-one lookup are definers (which are translated using the RELNAME of the RISTRUCT) and PERSONS

(see VI.3.10) which are translated using the PERSNAME. The other translations are as follows :

Semantic Item	Surface String
FALSE	'No`
TRUE	'Yes`
UNDEF	'I dont know`
<the speaker>	'I`
<the hearer>	'you`

Some other output formulae are provided by certain conversation games (see VI.3.9) supplying the strings directly (for example, CGQUERY supplying the 'which` directly in utterances like 'which man`).

VI.3.9 Conversation Games

The overall flow of the MCHINE program, when running in conversational mode, is controlled by a set of 9 POP-2 procedures referred to as "conversation games" (the terminology and the idea are borrowed from Power (1974); cf. also Levin and Moore (1976)). Each game is supposed to be a stereotyped sequence of conversational actions and reactions, with each game being associated with some purpose or task (cf. Schank (1975)). This association is wholly in the mind of the programmer and in the way that the games are used; there is no "goal-directed invocation" as in, for example, Micro-Planner (Sussman et. al. (1972)). For example, there is a game CGANS for replying to a question, and CGBAFFLE for notifying the interlocutor of a failure to understand an input string. As one game can call other games, the structure of the dialogue proceeds in a sequence of sections, either nested or consecutive, corresponding to the various invocations of games. In standard POP-2 procedure call, a function cannot exit and specify which function is to be called

next - either it exits or it calls another function nested within itself. This would be rather restrictive for conversation games, since it is desirable for one game to be able to specify what is to be done next, without having to build up a deeply nested hierarchical structure for the whole conversation. This difficulty has been avoided by introducing three game-calling routines :

RCALL(x) : call x nested within the current game, returning to this point afterwards.

ECALL(x) : exit from current game and then call x.

EACALL(x) : exit from all active games and then call x.

The games are responsible for taking in input from the teletype (including lexical lookup), passing the string to the analyser, and accepting the result of the analysis (a piece of semantic network). The game can then decide, on the basis of the structure received, to perform any action whatsoever, such as updating or examining the "world-model", or initiating another game.

Each game has certain local variables, so that information global to the analyser can be controlled during the conversation. These include REMOTE, PRESENT (the two "tense" locations - see Section V.7), CURRTENS (which indicates which tense location is currently appropriate) and STARSTAT. The latter is the variable which indicates to the analyser where it is to start in the ATN, so a conversation game can influence the initial expectations of the analyser, using its predictions about whether a question, statement or command is imminent. Initially, the program RCALLs a game CGREET, which expects a ritual greeting string. It does not call the

analyser, but compares the input string with a pre-set list of greetings. If no match is found, it ECALLs CGBAFFLE to tell the user; if a match is found, the appropriate action (entered in the greetings table) is performed - this is usually repeating the greeting as output. The general game CGREADY is then ECALLed.

CGREADY applies the analyser to the next input string, starting from a neutral start-state which will allow any sentence or phrase to be analysed. On the basis of the ILLOCUTION of the analysed form, a more specialised game (CGANS, CGOBEY, or CGABSORB) is ECALLed. CGBAFFLE, which is applied when the analyser fails to produce a result, attempts to match the input string against a pre-set list of "farewell" utterances. If this fails, it utters ['Pardon ?`] and ECALLs CGREADY; if it succeeds in finding a "farewell", the appropriate action from the farewell list (again, usually repetition) is taken, and CGREET is EACALLed, since a new interlocutor is expected. The whole conversation can be terminated by terminating an utterance by the character $ instead of punctuation, or by a general System 10 POP-2 interrupt.

Punctuation is used to guide the analyser, but not forcibly. The setting of STARSTAT (made by the current game) can be overridden by the input routines as follows : a fullstop sets up a declarative-expecting state, and a question-mark sets up a question-expecting state. Even these STARSTAT settings should not cause the analyser to be tricked by wrong punctuation, since all start-states have a default arc which jumps back to the neutral start-state. Hence "Have you spoken to Mary." would be successfully analysed as a question, after some thrashing around at the start.

This is not very elegant, but it allows punctuation to provide some of the information sometimes gained from intonation.

In most dialogues, the games will be invoked consecutively, using ECALL, with EACALL being used for greetings and farewells. RCALL is used only when the current game is to be temporarily suspended while some conversational exchange takes place. This is a useful facility since it allows the program to seek information from the user at any stage without corrupting the current exchange. The only place that this has been tried is in the reference-evaluation of definite expressions (see Section VI.6). If no referent set can be computed for a definite expression, the reference-evaluation routine RCALLs CGQUERY to ask 'which <string>' where <string> is a "translation" (see VI.3.8) of the structure being reference-evaluated. The user can then reply with a noun phrase, which will be analysed (CGQUERY having set STARSTAT to a suitable value) and passed up to the routine which is evaluating the definite expression, with CGQUERY exiting normally. This exchange would take place while the analyser was running on some input for a higher game, and, in principle, such nesting could go on indefinitely.

The following games are used:

CGREET : match and reply to greeting.

CGADIEU : match and reply to farewell

CGREADY : expect any analyseable utterance

CGANS : find answer to question

CGTELL : output reply to a question

CGOBEY : obey a command

CGQUERY : get information from user.

CGAWAIT : expect an imperative

CGBAFFLE : complain about input failure

VI.3.10 World Model

The semantic network system is supposed to be general, in the sense that it can be used to represent any relational structure (see Section III.10). The particular toy world that has been used for testing the conversational system is simpler than the SHRDLU BLOCKS world.

The world contains pseudo-records with dataclass PERSON. Each PERSON has components PERSNAME (a POP-2 word), PERSVIEW (a POP-2 context) and a PERSCRED ("credibility" - 0 or 1). There are RELATIONs which can hold between SETs of PERSONs - FATHER, MOTHER, BROTHER, SON, etc. and some RELATIONs which can be used to attribute properties to the PERSONs - MAN, WOMAN, etc. The reason for choosing this world rather than, say, a BLOCKS world, was that two areas of grammar that were originally to be examined were indirect speech and tense. The world of PERSONs seemed to allow a natural-sounding dialogue in which questions like "Did Harry say that Fred likes Mary ?" could be posed, for example. Unfortunately, these long term aims did not come to fruition, so the choice of subject matter may seem slightly arbitrary.

There are four variables - SPEAKER, HEARER, SELF, INTERLOC - which keep track of how the conversational roles are being filled. SELF always holds a pointer to a PERSON with PERSNAME "MCHINE" and INTERLOC holds a pointer to whichever PERSON is regarded as "talking to" MCHINE. The values of SPEAKER and HEARER are set up in the

obvious way (by the conversation games) whenever input or output is to occur. Initially, SPEAKER = INTERLOC and HEARER = SELF.

Imperatives from the interlocutor are regarded as requests to carry out an operation stored under the TASK property of the RELATION in the imperative, using the RELATION's role-fillers as arguments (there is a notational device for keeping track of which roles correspond to which argument-places). Although an earlier version of the program included a simple table-top world in which the imperative system operated successfully, it was hard to fit plausible commands into the PERSON world. The final version of MCHINE would therefore react to an imperative by entering CGOBEY, getting the TASK corresponding to the meaning of the imperative, then giving a POP-2 error, since TASKAPPLY (the execution routine) is not defined in this version.

Questions are treated as requests for information, and the semantic network is searched using the semantic structure of the utterance as a pattern. Since the semantic network system allows the explicit representation of truth-values, a distinction is possible between TRUE, FALSE and unknown relations; these will produce the answers 'Yes`, 'No`, and 'I dont know` respectively. If the question is a WH-question, any item found in the network-search will be returned as the answer, via the translation routine.

Statements are taken as assertions which are TRUE in the speaker's world model, and this fact will be recorded (storing a POP-2 context with each PERSON makes this straightforward). The speaker's PERSCRED is then examined, and, if this is 1, MCHINE attempts to assimilate the assertion. Depending on the state of

MCHINE's own semantic network, the reply will be either 'I know', 'I disagree', or 'Really ?', depending on whether the assertion is already recorded as TRUE, FALSE or not known.

VI.3.11 Semantic Hierarchy

The system of antonyms and sub-classes mentioned in Section III.10 is implemented fairly crudely. Each semantic category (usually represented by a definer) has on its property list a HIERINFO record. This provides a pointer into a 3-dimensional array which represents the necessary hierarchy. Each category has one super-category and a list of sub-categories; the categories are also clustered into antonym sets. Using numerical indexes into an array is merely a fairly efficient way of implementing this multi-dimensional classification, as it allows quicker category-compatibility checking and uses less space.

Section VI.4 : Implemented Grammar

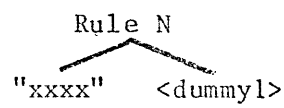
Although earlier chapters made various suggestions concerning points of English grammar, it has not been possible (for a variety of reasons) to incorporate all of these ideas into the MCHINE program. This sections summarises the grammatical rules which are coded in the working program.

VI.4.1 Noun phrases

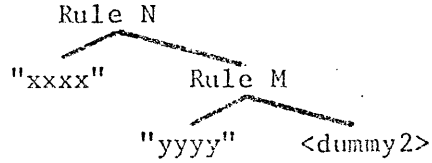
Only very simple noun phrases are covered (unlike, for example, the TESSA program - see Section VI.5). The main reason for this is that it seemed uninteresting to write an ATN which would allow a whole range of modifiers before the noun, unless the extra options introduced a need for new processing facilities. In the grammar as a whole, there were certain structure-building operations which often recurred; for example, going from a structure like (185)(a) to one like (185)(b), where "yyyy" was the input word and the <dummy> node is the "current" node in each case.

(185)

(a)



(b)



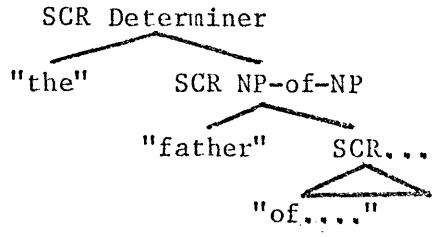
Hence, any repetitive structure-building which merely accumulated a tree in some simple fashion was not of any great syntactic interest. (The various modifiers are all radically different semantically, of course). Possessive noun phrases and non-restrictive adjectives both require special building operations, and it was important to check that the implemented ATN does not rule out such constructions.

The noun phrase part of the grammar allows a determiner or possessive (optionally), one or more non-restrictive adjectives (optionally), one or more restrictive adjectives (optionally) and a head noun (obligatory). Alternatively, a single proper name or personal pronoun can function as a noun phrase. The distinction between a restrictive and a non-restrictive adjective is a matter of use (i.e. how it relates to the rest of the context), rather than an inherent aspect of the lexical item. However, for an analyser to distinguish between the two usages of a single lexical item would require a very sophisticated use of contextual information, which is beyond the capability of the MCHINE program. In order to test that MCHINE's grammatical system nevertheless allows the requisite structure-building to take place, the adjectives used were marked in the lexicon as either restrictive or non-restrictive, and this feature was tested explicitly in the ATN.

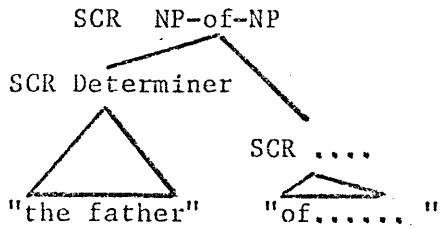
After the analyser encounters a head-noun, it looks to see if the current word could start an adjunct to the noun phrase. The head noun is temporarily stored in the named grammatical stack register HEADNOUN, so that its attachment can have the adjunct incorporated into it or not. This allows a structure like (186)(a), instead of (186)(b).

(186)

(a)



(b)



There are five possible post-modifiers allowed :

(187)

(a) an "of + NP" phrase

(b) a verb phrase starting with an "ing" form

(c) a verb phrase starting with an "ed" form

(d) a restrictive relative clause (with or without a "wh-word" to start it)

(e) a non-restrictive relative clause

In case (187)(a), a node is erected with the SCR NP-of-NP, and processing continues. The NP-of-NP rule attempts to generalise the "possessive" relationship, (the SCR Possessive differs only in minor details from NP-of-NP). It is fairly well established that there is

no constant referential semantic relationship between the items in a "possessive" construction (see, for example, Stockwell, Schachter and Partee (1972, Chapter 11)).

(188)

(a) John's book :

the book John wrote ?

the book John owns ?

the book John is holding ?

(b) Bill's present :

the present Bill gave ?

the present Bill received ?

The way that the two SCRs Possessive and NP-of-NP operate is as follows. The head noun's semantic item contains a "relation-instance" with some slots unfilled. The SCR scans these slots in order (the order being specified when the relation is defined) to find the first unfilled slot, and then inserts the meaning of the "possessing" item in that slot. This works on simple examples, but so would some simpler, cruder trick. The extra complication of searching for free slots has been used in the hope that it will be the prototype for some more general device.

Cases (187)(b) and (c) are handled in a very similar way to (d). The incoming verb phrase is analysed by the VP network, with the inflection of the opening verb determining whether an "active" or "passive" role-placement rule is required. The resulting structure (which is a definer) is handed to the SCR NP Modifier, which takes the head-noun meaning (also a definer) as its other argument. The SCR constructs a new definer identical to that from the verb-phrase, but with the head noun meaning set as a restriction on the selected slot. This is exactly the structure that would result from a corresponding relative clause (see Section VI.4.6) so the semantic similarities between (187) (b), (c) and (d) are captured without incorporating them into the syntax (as, for example, in Smith (1964), Burt (1972)). (Cf. (189)(a) and (b)).

(189)

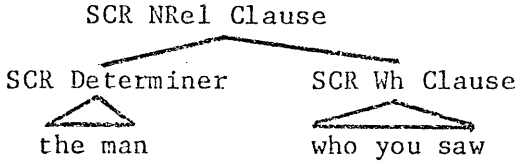
(a) The man who is speaking to you.

(b) The man speaking to you.

Restrictive and non-restrictive relative clauses are distinguished by the absence or presence, respectively, of a comma following the head noun. This is similar to the trick of marking adjectives in the lexicon as restrictive or not, in that the approximation is justified by the aim of trying out the various grammatical processes involved.

Restrictive relative clauses are treated as forming a further modification of the meaning of the head noun, and are incorporated into a structure somewhat similar to (186) above. Non-restrictive relative clauses are treated as making an assertion about the set of

things referred to by the whole preceding noun phrase. This necessitates attaching the contents of HEADNOUN (to complete the NP), and altering the contents of TOPNODE (the current task, which should have been the NP) to be a rule node combining NP and relative clause :



The NRel Clause rule, when executed, makes the appropriate assertion, with due consideration for speaker/hearer context.

Almost every semantic item in the system is a SET, a RELINST or a DEFINER. (See Section VI.3). All these classifications can be refined further by the addition of sense-properties (e.g. DEFINITE) and new items (other than RELATIONS, which are atomic) can be built up from existing ones. A common noun is a DEFINER, with an added component NUMBER, and the Determiner SCR merely adds all the sense-properties of the determiner to the head noun (i.e. the meaning of a determiner is defined entirely by its list of sense properties). All the other SCRs involved in building referring expressions combine definers to build a more complex definer, either by filling in roles or by setting restrictions on selected slots.

The general problems regarding reference-evaluation (see Section V.6 and Ritchie (1976)) have not been solved in the MCHINE program. Complex referring expressions are not reference-evaluated when built, but are manipulated as they are in question-answering, etc. (Even here, the matching routines have not been tested on very complex examples, and so are suspect). However, to simplify this cumbersome

generality, a sense property INDEXICAL has been included, which can be marked on a referring expression in the lexicon. (So far it has not been used on non-lexical phrases). An INDEXICAL expression (roughly corresponding to the class of "deictic" expressions discussed in Section V.6) is reference-evaluated during the SCR-evaluation process, as soon as it is built into the tree. From there, the definer is replaced by a set of "referents", which simplifies later processing. The only items which it seemed safe to class as INDEXICAL were the personal pronouns ("I", "you") and proper nouns ("Fred", etc.).

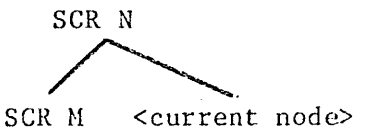
The sense-properties used on determiners are DEFINITE and SPECIFIC, each of which can be TRUE or FALSE. These guide reference-evaluation in a way which approximates the scheme of Section V.6. DEFINITE = TRUE causes the program to search its world model for a RELINST to match the one in the definer it is processing, and to use that RELINST if it exists; if it fails to find one, it examines the definer (which should have certain ad hoc markings to allow this routine to operate) to see if it uniquely defines a set of items, in which case the evaluation continues. If both parts of this fail, the analyser complains (see VI.3.9 for the means of "complaining"). SPECIFIC = TRUE causes the program, when reference evaluating, to replace the definer with its referent set. SPECIFIC = FALSE causes the definer itself to become the result of reference evaluation. Although this algorithm is written in to the program, it has not been exercised on a full range of test cases, so the ideas stand or fall by the arguments in Section V.6

VI.4.2 Verb phrases, Auxiliaries and Predicates

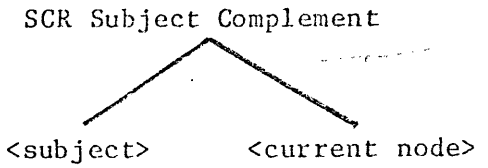
The auxiliary verbs are handled in a comprehensive but fairly straightforward way by a detailed, non-embedding network, based on the notes in Section V.2. At the stage when the auxiliaries are being analysed, the tree structure will always include a structure like (190)(a), (often as part of a structure like (190)(b)), where the <current node> will eventually hold the role-placement SCR for the verb phrase.

(190)

(a)



(b)



The information in the auxiliary sequence is used to set properties and restrictions on the current node and its entries. The properties PERFECT, PROGRESSIVE, NEGATIVE are all set on the semantic item, and the SRULE component is given a restriction to disallow either Passive or Active role-placement rules, as appropriate. The properties MODALITY, POLARITY and TENSE may also be set on the semantic item in the Subject-Complement node if the auxiliary sequence is within a clause, rather than just a verb phrase. By the time the analyser reaches the main verb, all these properties should have been set up, and only the verb and its objects, if present, need to be processed.

Some other parts of the grammar also use the verb phrase network, but do not enter it right at the beginning of the auxiliary network (e.g. the NP-modifier construction - see VI.4.1). In such cases, the other part of the grammar must ensure that the right properties are set up before joining the VP network.

The main VP network is very simple, due to the lexicon-driven strategy described in Section III.9. The network branches into 3 paths (corresponding to two objects, one object, or no objects), and on each path the various possibilities are provided by the object-information properties on the lexical entry for the main verb; this gives a list of lists, each containing an SCR and some state-names. The analysis branches further, depending on how many options are given in the lexicon. On each path, however, the operations are of the same general form. A rule-node is erected using the given SCR, and the verb is inserted as the first argument (i.e. as the leftmost daughter). A piece of ATN is constructed (from the given state-names) which will try to fill the remaining argument places with the appropriate number of constituents, and the analysis proceeds through this network.

This device makes it very easy to add new verbs, with different object configurations, to the grammar. For example, a verb "grunk" which combined with two objects using SCR 4, the first object being a prepositional phrase with "to", and the second being a "wh" clause, could be entered in the lexicon with the property

[OBJ2 [SCR4 STPREPTO STWHCL]]

The grammar would not need to be written. This may seem a trivial point, but the TESSA grammar(and presumably the SHRDLU one, (see VI.5 below)) searches explicitly in its recognition routines (the equivalent of MCHINE's ATN) for any object configuration that it handles, and would thus need a new piece of grammar for each new verb.

The SCRs that act as role placement rules are almost all of the same form. The main verb's semantic item is a RELATION, and this is used to construct a RELINST. The object meanings are slotted into various roles in this RELINST. This RELINST should have at least one role unfilled, and it becomes part of a DEFINER, with one of the unfilled roles being the selected role. This definer forms the meaning of the verb phrase. Different role-placement rules (of the same object number) differ in the way they permute the surface constituents into the RELINST roles, and in the role that they select for the definer.

The verb "be" is treated differently. It has no object information in the lexicon, and is not regarded as a RELATION from which a RELINST and a DEFINER can be built. On encountering "be" in the main verb position (and making sure it is not just one of the auxiliaries) the analyser discards the "be" word (i.e. does not attach it to the tree) and begins to build the complement (that is, whatever follows the "be" verb) directly on to the node which would have held the role-placement SCR if a major verb had been encountered (i.e. the <current node> in trees like that represented in

(190) above). Since noun phrases, adjectives and wh-clauses are all represented as definers, this provides a standard interface for the SCR Subject Complement.

This means that there is only one Subject Complement rule for all constructions, including those in (191).

(191)

(a) John hates Mary.

(b) John is clever.

(c) John is a doctor.

(d) Mary is hated by John.

In each case, the subject meaning is placed in the selected part of the definer, provided by the complement, to form a relation-instance.

More subject complement rules would be needed if more complicated constructions like (192) were included.

(192)

(a) There are lions at the bottom of my window-box.

(b) What I want to do is vomit.

(c) It's bread pudding that he likes.

(See also VI.4.5 for some complex subject-complement constructions).

Some SCRs are marked as "active", and some others as "passive". to cover a sentence like (191)(d), the following are needed :

(193)

(a) An SCR, say SCRP1, marked as "passive", which takes two arguments (i.e. one verb + one object), putting the object into the AGENT role and selecting the GOAL role.

(b) the entry for the verb "hate" must include the syntactic property [OBJ1 SCRP1 STPREPBY] where STPREPBY is the name of an ATN state which expects "by + NP" to follow.

That is, the object-phrase is treated as an "object" prepositionally marked with "by". This fails to use the fact that getting a "by" phrase and using it to fill the AGENT slot is a recurring pattern; otherwise, it is fairly neat.

The object-information of a verb allows the "object" of a verb to be any constituent, such as an infinitive phrase, for example. Hence "I believe him to be clever" is handled by giving "believe" an object-information property specifying two objects (NP and "to"+VP), and a slightly more complex role-placement rule than usual. This SCR has to form the two object meanings into a RELINST (in much the same way as the Subject-Complement SCR does), then put this structure into one of the roles in the RELINST being constructed for "believe".

VI.4.3 Prepositions

For each preposition, there is a corresponding arc which tests for exactly that preposition, and a state which contains just that arc. These states are used in specifying object information for verbs. The preposition arcs all connect to the initial state of the noun phrase network. Since one of the arcs of the NP network searches the wh-register for its result (see Sections V.4 and VI.4.6), "dangling" prepositions are automatically included in the grammar. That is, the recognition rules for (194)(a) and (b) will handle (194)(c) without further elaboration.

(194)

(a) You spoke to Mary.

(b) Who did you address ?

(c) Who did you speak to ?

VI.4.4 Imperatives

Imperatives are treated simply. The options at the start of the sentence include looking for an untensed main verb, with an entry under the TASK property on its meaning; other options include certain combinations of "do", "don't", "do not", followed by such a verb. The incoming verb phrase is then processed by the ordinary verb phrase ATN, and its meaning handed up as the meaning of the sentence, with ORDER entered as its ILLOCUTION.

Although embedded imperatives (e.g. "He told me to speak") could be handled at the surface level by treating the infinitival phrase as an "object" of the verb of ordering (as with "believe" - see VI.4.2 above), this has not been tried, as the semantics of embedded

commands has not been explored.

VI.4.5 Embedded clauses

Sentences like (195)(a) are automatically covered by incorporating the "that"+S construction as an option in the NP network. Such incorporation is desirable if we are to avoid having to treat sentences like (196)(a) and (b) as separate cases).

(195)

(a) That you like Mary surprises me.

(b) It surprises me that you like Mary.

(c) It surprises me.

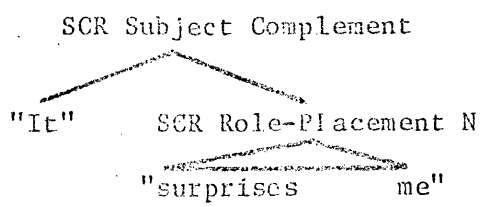
(196)

(a) I said that you were leaving.

(b) I said something.

The related sentence (195)(b) is assigned the same semantic structure as (195)(a), in the following way. The initial clause "It surprises me" is analysed, resulting in an SCR tree of the form

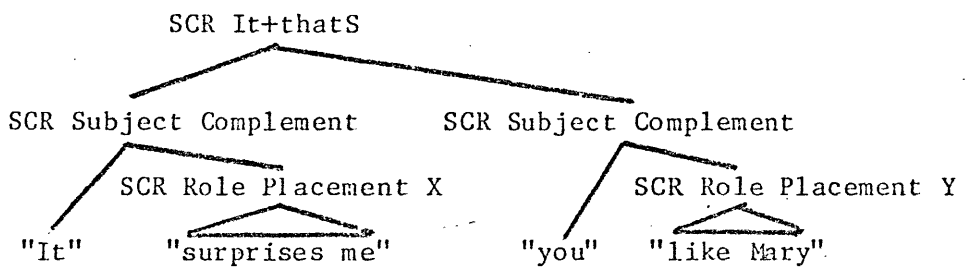
(197)



After "me", the analyser finds that there are more words left in the

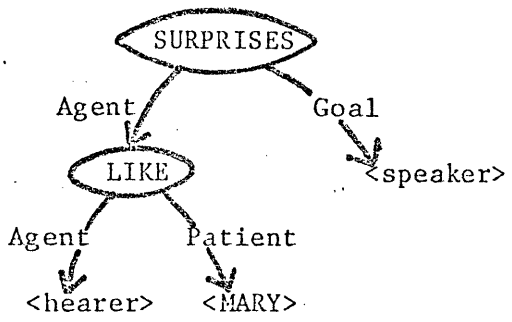
input, and so uses the BOTTOMUP mechanism (see Section VI.3.5) to find a new state. The lexical entry for "that" gives it the syntactic feature "THATS", and there is a pointer from that feature to the ATN state SSBTHS, which "expects" an embedded clause. The analyser stores the old TREETOP (the Subject-Complement node) on TOPNODES, establishes new nodes (see Section VI.3.5) and continues. At the end of the sentence, TOPNODES is found to be non-empty so the nodes on it are evaluated, and an SCR sought which might relate the various EXTVALs found. The rule "SCR It+thatS" is selected (as a result of certain underhand tricks - see below) and a tree like (198) constructed.

(198)



The semantic item for the left-hand Subject Complement node will be a relation-instance with the semantic item for "it" entered in one role. The "It+thatS" rule searches its first argument for the "it" meaning, and replaces it with the second argument. That is, the RELINST produced by the right-hand Subject Complement rule node is slotted into the indicated role in the RELINST from the left-hand node. This gives a structure like (199).

(199)



It is worth digressing here to describe one problem that arose while debugging this part of the grammar, as it illustrates how computer implementation can draw attention to inadequacies in a superficially attractive scheme. When the analyser was first tried on sentences like (195)(b), it produced a structure like that for (195)(c). The grammar had been written so that sentences like (195)(b) would be analysed by first analysing a clause like (195)(c), then doing a "restart" (see above) when the word "that" was encountered, and using the rule-selection technique to find a two-argument rule to combine the two clauses. The analyser did not produce the expected result, but instead returned a structure corresponding to the meaning of (195)(c). At first, this error suggested that the restart mechanism had malfunctioned, and discarded the second clause (either in its input form or after processing it). Further investigation revealed that something else had happened. The whole mechanism had functioned exactly as intended (an a priori incredible occurrence, as any programmer will attest), but the input specifications of the structural combining rules had not been sufficiently narrow, and the "wrong" rule had been selected. The intended rule ("It+that S") had specifications

((RELINST) (RELINST))

for two clause-meanings (see Sections III.10, IV.1, VI.3.7 for explanations of the semantic notation). The "wrong" rule found was that for non-restrictive relative clauses (see Sections V.4 and VI.4.6), which had specifications

((ENTITY) (RELINST))

for a term meaning and a clause meaning. Since a RELINST is also an ENTITY (anything is an ENTITY), the arguments had matched the specification for the NRelClause rule, which happened to be examined earlier in the selection routine. Since NRelClause returns its first argument as its result, the first clause-meaning became the overall result of the analysis. In order to patch up this inadequacy, the NRelClause specification was made more specific :

((ENTITY) (WHCLAUSE))

This hid the problem temporarily, but the difficulty reappeared when time-adjuncts were incorporated into the grammar. A two-clause sentence like (200) was also to be analysed by using the restart and rule-selection technique.

(200) I spoke to Gladys when you were speaking to Boris.

Here, the second clause is a WHCLAUSE, so once again the list of two clause-meanings will match the input specification for NRelClause. It might seem that the solution is to have the time-adjunct rule input specification require something like

((RELINST) (TIMEREF))

or something similar, but that would not avoid the overlap. Since "when"-clauses can indeed form non-restrictive relative clauses (cf. (201)), they must match the input specifications of both rules, no matter what they are classed as.

(201) Half an hour ago, when you were speaking to Boris, I spoke to Gladys.

The MCHINE program has an ad-hoc solution to this problem. There is a short list of "clause-rules" and it is this list which is searched by the rule-selection routine. Since the rule-selection routine is used only in two parts of the MCHINE grammar (after restarts and in certain wh-clauses), and clause + modifier pairs are involved in each case, this range of rules is sufficient.

VI.4.6 Wh-Clauses

One of the most complex parts of the MCHINE grammar is the part for processing wh-clauses. The grammar is an implementation of that discussed in Section V.4, but the general facility for analysing other forms of Complex Noun Phrases (e.g. Appositional Phrases) is not included. To be more accurate, no "SCR Apposition" has been designed, so the program would not be able to build the right structure for the appositional cases, even though it would follow the correct path through the transition network (since, as pointed out in Section V.4, appositional phrases are merely a special case of the wh-clause path).

There are various slightly different kinds of structures involved in wh-clauses, and it is worth describing them in detail here.

Wh-words are cross-classified using four syntactic features : WH, WHDET, WHREL, and WHFULL. All the wh-words ("who", "which", "that", "what") are marked WH. Those which can form the determiner of a wh-phrase (i.e. "which", "what") are marked WHDET. Those which can act as relative pronouns with an antecedent ("which", "who", "that") are marked WHREL. Those which can act as relative pronouns without an antecedent ("which", "who", "what") are marked WHFULL.

A wh-phrase can be either a WHFULL word, or a WHDET word followed by a head phrase :

(202)

(a) Who did you see ?

(b) Which large policeman arrested you ?

A wh-clause is a clause which starts with a wh-phrase, whether embedded (as in (201)(c) and (d)) or forming a sentence (as in (202)(a) and (b)).

A wh-question is a question in which the topmost clause is a wh-clause (e.g. (202)(a) and (b)).

A relative clause, with antecedent, is a noun phrase followed by a wh-clause.

An independent embedded wh-clause is a wh-clause starting with a wh-phrase and without an antecedent.

(203)

(a) What you did was wrong.

(b) Which book you choose doesn't interest me.

(c) I know which book you were reading.

(d) I eat what I see.

Independent embedded wh-clauses can be classed as either independent relative clauses (where the opening wh-phrase is just a single WHFULL word), as in (203)(a) and (d), or embedded questions (where the opening wh-phrase can be more than one word), as in (203)(b) and (c). These differ semantically in that the independent relative clause is used to single out some item(s), in the same way that the corresponding ordinary relative clause is, whereas the embedded question represents the question expressed by the corresponding wh-question. One form refers to some thing(s), the other refers to some proposition or query.

These are all informal descriptions, rather than precise definitions, but they should make the later exposition somewhat simpler.

As observed in Ritchie (1977), the notion of a "noun phrase" is hard to define, and the things that are traditionally referred to as "noun phrases" are a heterogeneous collection whose common characteristic is that they can act as terms in a relationship. The

"noun phrase" network in the MCHINE grammar therefore has to include such un-phrase-like items as independent wh-clauses, amongst others. A great deal of effort went into merging the networks for the various kinds of wh-clauses, and they all use the same network with only minor variations in how it is entered. The difference between the two forms of independent embedded wh-clauses is expressed by the ILLOCUTION property of the structure built; an independent relative clause is marked "SAY" and a question is marked "ASK". As can be seen from the above description, some strings will be ambiguous between the two interpretations, if the opening wh-phrase is only one WHFULL word. This ambiguity should often be resolved by the rest of the sentence, since the verb-frame into which the independent clause fits may have a narrow enough semantic restriction on the relevant role to eliminate the unwanted interpretation. The necessary semantic classifications to separate "question-meanings" from "referring expressions" are fairly subtle, and are beyond the MCHINE program. Hence the analyser treats as ambiguous not only (204)(a) and (c) (which are ambiguous) but also (204)(b) and (d) (which should be resolved within the sentence).

(204)

(a) I know what you saw.

(b) I like what you saw.

(c) What is on the noticeboard does not interest me.

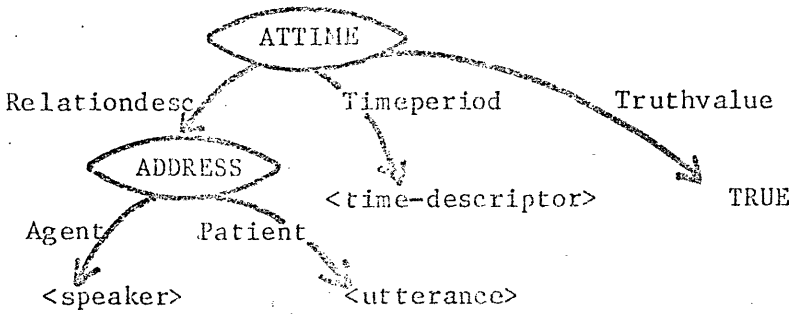
(d) What you broke was very valuable.

VI.4.7 Time Adjuncts

The grammar allows time-adjunct clauses either before or after the main clause. One of the options at the start of a sentence is a time-adjunct, after which the ordinary sentence-initial state is re-entered, thus allowing sentence-initial constructions (e.g. subject-verb inversion) to occur after a time-adjunct. Time adjuncts at the end of the main clause are found by the BOTTOMUP/ RESTART system, together with the mechanism for searching for an SCR (see Section III.9).

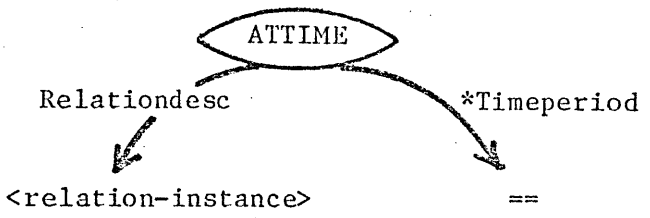
There are two time-adjunct SCRs, which differ only in the order of their arguments, for combining the meaning of a time-adjunct with the meaning of a main clause to create a RELINST with the relation "ATTIME". Events (i.e. utterances in a dialogue with the MCHINE program) are "remembered" in a semantic network of the form (205).

(205)



The tense-manipulation described in Section V.7 is implemented in the time-adjunct rules, but no complicated dialogues have been tried to test it.

(206)



A time-descriptor is implemented as a definer in the form (206), with an entry on the property list under "TIMEPATT" which provides the necessary information, in the following way.

The TIMEPATT is a triple, the first component being the point/interval indicator, and the other two components indicating how the end-points of the time period relate to the end-points of the event described by the relation-description. The second component of the triple corresponds to the start of the denoted time period, and the third component corresponds to its end; these two components contain distinct values (actually, 0, 1 and 2) depending on whether the end-point in question is represented by the start of the event, the end of the event or is undefined.

As discussed in Section V.7, "when" clauses can be handled by the ordinary wh-clause grammar, and this is how the implemented grammar operates. However, the grammar for time-adjuncts at the beginning of the sentence has to make the distinction between a "when" clause and other "wh" clauses, because only in the former case will an SCR tree with the rule SCR Time Adjunct need to be built. This distinction can be made by having a syntactic feature "TBIND", which is present on time-binders ("after", "before", etc.) and on "when". The recognition rules for time adjuncts check for this feature before trying to process the time-clause and erect the SCR

Time Adjunct node if appropriate. This does not interfere with the way that the "when" clause is analysed, and so the wh-clause grammar can still be used.

Section VI.5 : Comparison with Other Programs

Although Chapters II and III discussed the ideas of the MCHINE program and their relationship to other frameworks, no comparison was offered of the more mundane details of actual programs. It is difficult to make fair or accurate comparisons between natural language programs in artificial intelligence, since there is a dearth of real statistics about the performance of the programs. It is good that such prosaic details have not assumed undue prominence at the expense of the principles involved, but it makes it difficult to summarise what practical work is being done.

Three programs have been documented in sufficient detail to allow some comparison, so this section will provide a discussion of the relative merits and deficiencies of the MCHINE program with respect to the Thorne-Bratley-Dewar parser (referred to here as the TBD program for short), Winograd's SHRDLU and Soul's TESSA (Thorne et al (1968), Winograd (1972), Soul (1975)). Unfortunately, these programs are very similar in theoretical orientation, and so the survey here will be somewhat narrow.

VI.5.1 Technical Details

The TBD program produced a labelled bracketing of the input string, with special markings for indicating any constituents that were not in "deep structure position", and where these constituents had been "transformed" from. That is, it was a wholly syntactic parser, operating on single sentences. It occupied 16K of 48-bit words on the English Electric KDF9, and took between 0.5 and 2.2 seconds to analyse sentences, depending on complexity.

SHRDLU was an entire dialogue system, but it contained as a (separable) subpart a syntactic analyser. This analyser could be run autonomously on isolated sentences, and could produce analyses for all the sentences in the list published for the TBD program. The SHRDLU system as a whole could deal with a narrower range of sentences, but its performance was not trivial. The analyser alone occupied about 25k of 36-bit words on the PDP-10, and the whole system took about 80k (both figures include the underlying LISP system). The system responded to sentences in between 5 and 20 seconds.

TESSA was intended to be an improved implementation of the grammar described in Winograd (1972). (The SHRDLU program did not include all the grammatical constructions exactly as described). It ran on isolated sentences, performing no semantic processing, and included some elaborations of the SHRDLU grammar. The program occupied about 30k of 36-bit words on the PDP-10, plus an 11k POP-2 system. The time taken to analyse a sentence is about 2 to 3 seconds.

As stated in Section VI.0, the MCHINE program can run either in isolated sentence mode (when it builds a semantic network for each phrase or sentence) or in conversational mode. It can therefore be compared with the performance of all three programs on isolated sentences, and with the conversational ability of SHRDLU. In isolated sentence mode, MCHINE's performance is spectacularly cumbersome. Not only does it occupy 56k total core space, it is by far the slowest of the programs (by a factor of about 15). However, it must be borne in mind that the MCHINE program constructs a

semantic network as it proceeds, and so is performing what many other systems would treat as two stages. The conversational ability of the MCHINE program also seems to be inferior to that of SHRDLU, assuming that Winograd's "sample dialogue" is a typical sample (as opposed to a one-off performance). The only mitigating factor for MCHINE is that it is fairly robust (by the standards of such programs) in conversational mode, and several sample dialogues have been produced without error.

Although this summary has been included for completeness, it must be admitted that certain details are almost completely irrelevant to artificial intelligence and linguistics. The processing time consumed by a program is of little interest, as it depends so directly on implementation details and the particular programming language used. For example, POP-2 subroutines can be defined as "functions" (held in general purpose identifiers) or as "operations" (with their own syntactic type). Since functions therefore require run-time checks that are redundant for operations, the call of an operation is about 25% faster. Most of the routines in the MCHINE program are defined as "functions", and hence the program could be speeded up simply by redefining them as "operations". Such an improvement is hardly of theoretical importance. Thorne, Bratley and Dewar (1968) included in their data the number of ATN states visited in each analysis. This is a somewhat more interesting statistic, but unfortunately similar figures are not available for the other programs.

The point made in Section I.4 could be emphasised here. It is not the superficial performance of the program that is important, but the principles embodied in it. MCHINE's pedestrian performance can perhaps be attributed to the aim of developing general mechanisms, rather than achieving a virtuoso performance.

VI.5.2 Grammatical Coverage

The coverage of the programs is difficult to compare, since different workers tend to use slightly differing vocabulary and constructions. However, it is possible to produce an imaginary "master sample set" by combining the published data from all the programs and making some reasonable assumptions about "equivalent" examples. One can only speculate about how the programs would fare on these standardised examples, but some reasoned estimates, tempered with charity, are possible. Appendix C contains such a hypothetical standard sample set, and an estimate of the performance of each program. The summary concerns only the ability of the programs to analyse sentences, so SHRDLU's mechanisms for producing sentences have been ignored. It is not clear how punctuation will affect the various programs. MCHINE is guided by punctuation only slightly (as noted previously), and the TESSA list (Soul (1975)) includes a few examples in both punctuated and unpunctuated form. SHRDLU depends on question marks to indicate questions. Since the SHRDLU parser successfully analysed all the sentences on the TBD list, the information in Appendix C regarding SHRDLU is based on that for TBD. Soul (1975) has also provided a list of complicated noun phrases that are within the capability of TESSA, so that program can probably do better than the performance table might indicate.

CHAPTER VII

CONCLUSIONS, PROBLEMS AND SPECULATIONS

Section VII.0 : Preamble

This project has explored the English language in various ways.

Firstly, it has indicated how the artificial intelligence devices embodied in computational grammar can be used for linguistic description.

Secondly, it has used this framework to analyse a few fragments of English.

Thirdly, it has made some proposals concerning the processing mechanisms needed in a model of English sentence understanding.

However, this work has merely started to attack some of the problems, and has not really provided full solutions. Chapter VII suggests some of the points that it might be interesting to follow up, ranging from fundamental alterations in the processing mechanism to small points of detail concerning English grammar.

Section VII.1 : Structural Combining Rules

VII.1.1 Present Version

The main advantage of structural combining rules, as currently defined, is that they provide an interface between what are traditionally known as syntax and semantics. The hierarchical grouping of surface constituents which has been so much part of previous linguistic theories has often been regarded as syntactic, necessitating the development of syntactically-motivated rules (e.g. phrase-structure grammars). The compromise represented by SCRs is to accept that there are tree-like patterns in surface structure, but to stipulate that the rules which express these groupings must be devised on semantic grounds, and that the operations that these rules perform should be on semantic structures.

VII.1.2 Bidirectional Rules

One obvious shortcoming of the entire computational grammar framework is the lack of any means of sentence production, and the one-directional rules which have resulted from this bias. A possible refinement is to see how much of the information used by the analyser is also useful in sentence production, and to try to factor out those parts into rules which are interpretable in both directions. (Cf. Section I.2). It is not obvious whether this would be best tackled by writing a separate production grammar, or whether the current grammar should be modified. The first step would probably be to make SCRs bidirectional, since they will be needed in production. The experimental form of SCRs, described in Section III.3, where the operation of the rule was subdivided into separate components, one

for each argument, would be more easily reversed than the form in which all computation is done in one unanalyseable block. Much of the recognition grammar might remain one-directional, since it is aimed at a strongly directional task, but even there some information regarding constituent-ordering might be extracted into the dual-purpose part of the grammar. One major problem is that the recognition rules often test for some sufficient condition before taking some action (not always a necessary condition) and some rules test simply for the presence of some particular marking or structure, without testing all the details of that item. Hence these rules might not be fully reversible.

VII.1.3 Left-Right Ordering

As discussed in Sections IV.5, and IV.8, SCR trees are built in a strict left-to-right fashion, with any re-ordering of constituents being performed by the ATN part of the grammar before the re-ordered items are built into the tree. This approach is particularly suitable for English, where much of the linguistic information is conveyed by word-order, but it raises doubts about the generality and flexibility of computational grammar. If grammars are to be written for languages which make much less use of sequential arrangement as a communicative device (using, for example, inflections instead), new problems arise. It is far from clear what criteria would be used for such grammars for choosing an ordering for the places in SCRs. This whole issue is very complex, and some of the arguments have already been raised within transformational linguistics, in the discussion of "underlying word order" and "universal bases" (cf. McCawley (1970), Peters and Ritchie (1969)).

VII.1.4 Focus and Topic

One of the many areas of linguistic communication which computational grammar has not touched is that of "topic" and "focus". Different arrangements of semantically similar sentences can be regarded as differing in the emphasis that they put on the various items of information in the sentence, and in the way that they separate "new" information from "given" information (cf. Halliday (1967a,b, 1968)). This is an obvious point for further investigation, since very little work has been done in this area computationally (but see Davey (1974) and Kay (1975) for some suggestions). Within computational grammar as defined here, the best way to describe such a notion would be to incorporate these aspects into the SCRs, so that, for example, "passive" rules would have different consequences, in terms of "topic", "focus" and "new/given" characteristics, from "active" rules.

Section VII.2 : The Analysis Procedure

VII.2.1 Present Version

Most of the ATN system used in computational grammar is fairly standard. The most interesting modification is the NEWLEVEL...NIL construct, which allows the grammar writer to specify a new processing level without having to state what will follow the constituent for which the new level is needed. Beyond that, the main advantage of writing the ATN rules for the MCHINE grammar has been to highlight some of the inadequacies of the ATN formalism as it stands.

VII.2.2 Top-down and Bottom-up

One disadvantage of the ATN formalism is that it has been developed mainly as a way of expressing a top-down search strategy. As discussed in Section III.9, the decomposition of networks into subnetworks promotes a classic top-down approach, and the analyser can do little with an input item that is not explicitly specified as a possible option in the current state. Many English constructions allow items to occur at any one of several points in the sentence (e.g. adverbs), and grammars should be a little more flexible in the way that they handle such constructions. There are other circumstances where it might be neater to let the input items guide the analyser, rather than having the analyser search single-mindedly for all possible options. For example, some verbs can take various configurations of objects, as in (207).

(207)

(a) We gave Veronica a teaset.

(b) We gave a teaset.

(c) He never gives to charity.

(d) We gave a teaset to Veronica.

(e) The chef is cooking.

(f) The chef is cooking something exotic.

At the moment, these are covered by having the analyser explore an exhaustive network representing all the possibilities for the given verb (see Section III.9 and V.8). It would be neater if the analyser could in some way process whatever objects are present and then allocate them to nodes in the surface tree.

Also, a relative clause may, under certain circumstances, be separated from its antecedent, as in (208).

(208) A man came in who had been at the party.

The relative clause is generally understood as modifying the most recent noun phrase in the sentence (cf. Grosu (1972)), so this seems to require a procedure which searches back for an antecedent, on encountering the relative clause. However, that would necessitate a grammar capable of analysing a detached relative clause, even if it was not anticipated, without assuming that it was an "independent embedded wh-clause" in the sense of Section VI.4.6.

Whether these kinds of "optional" constituent need a new formalism, or merely a more imaginative use of the ATN mechanism, remains to be seen.

VII.2.3 Predictions and Procedures

One of the difficulties in trying to construct a semantically motivated processing model is the problem of relating predictions made during the analysis of a sentence (possibly in some semantic terms) to methods of processing actual words. This has been discussed elsewhere, (see Sections II.7 and Ritchie (1977)), but no solution to the problem has emerged. As pointed out in Ritchie (1977), the ATN mechanism, as currently used, is an implicit (and rather unprincipled) way of effecting the conversion from high-level semantic predictions to low-level processing procedure. The first step in tackling this awkward area is to make this conversion explicit, in order to see what relationships do exist between semantic and syntactic categories. If we introduced a set of "category conversion rules", we might at least see what was going on, as a preliminary to systematising the process.

VII.2.4 : Demons and Packets

Over recent years, a particular form of control structure has been used for various processing models in artificial intelligence. It is based on the concept of a "demon" (Charniak (1972)), and has been used for English language programming by Marcus (1975) and Riesbeck (1974). (See Sections II.7 and II.10 for a fuller exposition). As observed in Section III.6, there are various formal similarities between the ATN "arcs" and "states" and Marcus' "demons"

and "packets".

The main differences between the two formalisms are in the possible ways of activating and de-activating the units. A packet is usually a smaller unit than the ATN states, and so several packets may be active at one time, whereas each path in an ATN analysis has only one active state. This extra flexibility in the packet method might avoid the unsuitablility of the ATN states for expressing alternative groupings of arcs, and the unions of states (see Section III.7). As mentioned in Section III.7, the ATN interpreter in the MCHINE program allows the NEWLEVEL arcs to specify a list of ATN states, which together make up the new active state. This is a covert way of gaining an advantage of the packet method.

This leaves unresolved the question of de-activating demons and packets. Since an ATN analysis path has only one currently active state, the de-activation procedure is simple - once all the arcs in the state have been processed (i.e. tested against the input word, and appropriate action taken) that state is no longer active. Each arc has to supply a new state for activation on the path it creates. If a demon-and-packet analyser is maintaining a list of active demons, there is not such a simple way of deciding when to remove a given demon from the list. Since a demon represents a possible option for the input word, and such options are generally plentiful, the active demons may accumulate somewhat. One possible method of de-activation suggested by Marcus (personal communication) is to class certain demons as "one-shot". These units would be automatically de-activated as soon as they were triggered, and would be, in this respect, similar to ATN arcs. Notice that a "one-shot"

demon is not exactly like an ATN arc, since the demon is not de-activated unless it is first triggered (this corresponds to an ATN arc "matching" an input word). If "one-shot" demons are never triggered, they will still hang around. If a demon were a "one-word" demon (i.e. it only stayed active for one word of input), it would be very much like an ATN arc.

Apparently, some compromise between the standard ATN and the demon-and-packet approach is needed. The flexibility of the latter, combined with some well-defined procedure for de-activating demons, should lead to a grammar-writing formalism that is easier to use.

Section VII.3 : Semantic Representation

VII.3.1 Present Version

The semantic network system adopted for computational grammar is general, and includes some useful facilities (such as the "expanded" and "elaborated" forms for relations), but in most respects it adds very little to the proposals of Rumelhart and Norman (1973). The only novel item is the discovery of a particular kind of network structure (the "definer") which is especially useful in specifying semantic structures which can undergo a variety of operations and which interface neatly with the syntactic constructs.

VII.3.2 Semantic Well-formedness

The concept of "semantic anomaly" has been used, rather uncomfortably, for some time within language research (e.g Katz and Fodor (1963), McCawley (1968), Winograd (1972), Wilks (1975)), but there is still no clear way of characterising the distinction between semantically well-formed and ill-formed meanings. If a sentence-analyser is to use referential semantic considerations to guide its processing (e.g. choosing between two possible analyses by selecting the one which offers the "better" meaning), then this whole area will have to be examined much more thoroughly. The over-simplified notion of "selectional restrictions" has been regarded as theoretically inadequate for some time, but computational models still resort to it, for want of anything else. Computational grammar includes the improvement of not treating violation of selectional restrictions as an all-or-nothing matter (as does the Wilks system - see Section II.6), but that just scratches the

surface, particularly as there are no principles developed yet concerning how semantic combinations are to be graded for ill-formedness.

There are a few preliminary proposals concerning ways of "choosing better interpretations" (e.g. Wilks (1973), McDermott (1973), Charniak (1972)) but the theoretical side has not been developed.

VII.3.3 Contexts and Referring Expressions

If we are to have structures for certain noun phrases that allows them to describe different objects in different contexts, then the question arises of which context should be used when a particular expression is evaluated. Using the analogy of Section III.4, let us identify a "state of the world" with an "environment", where "environment" is used in the programming sense (cf. Davies (1973), Stansfield (1975), Moses (1970)), meaning a set of values for variables. When a pattern is used to produce an item from the database (as has been suggested here for the semantics of specific, definite noun phrases), the item(s) produced will depend on the current values in the data-base - different environments will produce different "referent sets". The appropriate environment may be the hearer's model, the speaker's model, or some other state of affairs, possibly hypothetical or associated with some third person.

This means that the linguistic model will need some way of keeping track of which "environment" is to be used for evaluating the various referring expressions in a sentence or dialogue. (See Ritchie (1976) for some of the problems involved here). This device

will have to operate within the sentence analyser (not just in some global conversational routine) as the relevant context may vary within a sentence.

It is perhaps worth including some speculation concerning the possible connection between this problem and the notions of "specific" and "generic" discussed in Section V.6. For example, compare the two uses of "someone" in (209)(a) and (b).

(209)

(a) Did someone attack that man ?

(b) Someone attacked that man.

The "someone" in (a) seems to be non-specific, in that it is being offered to the hearer as a pattern; in this respect, (a) is similar to "Did anyone attack that man ?", where "anyone" is non-specific. In order to respond to the question, the hearer uses the pattern to search his world model for any appropriate items. The "someone" in (b) seems to be specific, in that it represents some item that the speaker does not expect the hearer to be able to identify further. There is a certain symmetry here. In (a), the speaker has no information about the referent, but assumes that the hearer has; in (b), the hearer has no information about the referent, but assumes that the speaker has. Possibly notions like "specific" could be re-defined in terms of the way that referring expressions are handled in the two contexts - speaker's and hearer's (cf. Hintikka (1973)).

Section VII.4 : Syntactic Markings

VII.4.1 Present Version

Computational grammar has tried to clarify the role that syntactic information plays in the sentence-analyser, by separating those devices that seem to be wholly non-semantic, without constructing a full "syntactic component". If syntactic features, properties and rules are allowed to be used in the grammar, but there is no committment to giving a syntactic explanation to every phenomenon, the amount of syntax needed can be assessed more accurately.

If we were to disallow any device that resembles "traditional syntax" (as Schank (1972) and Riesbeck (1974) seem to wish) we would bias the investigation from the outset. Conversely, trying to provide a full "syntactic analysis" for every sentence may introduce pseudo-problems, in that there may not be an appropriate way of describing some aspects of a sentence in a syntactic structure.

The MCHINE grammar has shown the need for twenty-two syntactic features, three syntactic properties, and about eight property-inheritance rules (see Appendix A). This is interesting, but not spectacular. It remains to be seen how this view of the role of syntax can be refined by further grammar-writing.

Section VII.5 : Guidelines for Descriptions

VII.5.1 Present Version

The guidelines used in devising linguistic descriptions are very rarely made explicit, and some of the intuitive meta-rules are so widely accepted that a linguist may not realise that he is tacitly using certain methodological axioms. Section VI.8 attempted to articulate some of the more important criteria that were employed in writing the MCHINE grammar, since some of the reasoning used in the linguistic descriptions would seem unfounded otherwise. This setting out of methodological axioms for grammar-writing is an important step, since lack of acknowledgement of such assumptions could lead to confusing disputes between workers following different schools of thought. (cf. Householder (1965, 1966), Chomsky and Halle (1965)). Although many of the principles are very general, and might be adopted by many linguistic frameworks, some of the guidelines are specific to this particular framework. It is an advantage of computational grammar that it makes even its peripheral assumptions explicit.

VII.5.2 Dynamic and Static Elegance

Linguists place great stress on the "elegance" or "neatness" of a particular grammatical description, and, as commented in Chapter I, this is one of the main means of assessing descriptions when there is more than one possible solution. In a processing grammar, the elegance may be assessed at more than one stage. Firstly, there is the "static" elegance of the grammatical rules as stated - are they obviously ad hoc or redundant, or are they general and simple ?

Secondly, there is the "dynamic" elegance of the grammar in operation - does the analyser thrash round exploring many dead ends, or is it efficient enough to constrain its searching ? (In programming terms, static elegance concerns the source code, but dynamic elegance concerns the run-structure). Only Marcus (1974, 1975) seems to have considered the latter as a topic, but it may be an increasingly interesting area in future. There is often a trade-off between the two forms of elegance, since a completely general static description of the options in a grammar may give no guidance on how these constructs are to be found. On the other hand, a more complicated description (containing more information to guide the analyser) may produce a shorter, neater processing stage. Anyone attempting to write recognition grammars should be conscious of this distinction, and perhaps consider what his methodological priorities are. The MCHINE grammar slid confusingly between these two criteria, trying to gain the best of both forms of elegance, and the project might have benefitted from an earlier realisation of this trade-off.

Section VII.6 : Registers

VII.6.1 Current Version

This project has achieved two useful results concerning "registers". The first is the classification of the kinds of registers and their uses, which will hopefully lay the foundations of a deeper investigation of this area (see VII.6.2). The second, more substantive, achievement has been to show some of the ways that careful use of registers can contribute to more adequate descriptions of English grammar. The analysis of relative clauses in Section V.4 is a particular example of this.

VII.6.2 Constraints on Registers

One major metatheoretical deficiency of computational grammar is the lack of any constraints imposed on any of its devices. For example, to state that registers are used for temporary information storage during analysis is to say almost nothing, unless some further claims are made concerning the various ways that these registers can and cannot be used.

Some of the register principles used in the MCHINE grammar have been developed independently by other ATN users (see Woods (1973)), which reinforces the case in favour of these constraints. Woods (1973) uses a general work register called the HOLD list for temporary structure storage, and imposes the following general principle:

"the constituent saved on the HOLD list must be used by some virtual arc, either at the current level or at some embedded level before a POP from the level at which it was saved can be taken." (p.119).

This is exactly the "tidying-up principle" which was found useful in writing the MCHINE grammar (see Section IV.8). Woods comments:

"The presence of such a facility in the model raises a number of questions of theoretical linguistic interest: for example, should the hold list be constrained so that constituents can be taken off only in the reverse of the order in which they are put on....or can constituents be taken off in any order? Our experience in writing grammars within this model has not turned up any examples which would resolve this question."(p.120).

The constraint that Woods postulates here would have a similar effect to the use of stack registers in the MCHINE grammar. In particular, the patterns described in Section V.4.2 would be partly covered by such a principle.

This area appears to be a promising one for further exploration.

Section VII.7 : Conversational Rules

VII.7.1 Present Version

Computational grammar has included a level of conversational description in its model of sentence-processing, since the "conversation games" of Section IV.7 provided a systematic way of structuring a dialogue in such a way that the conversational information could interact with the sentence analyser. This is yet another area where the barest skeleton has been constructed, but at least the step has been taken.

VII.7.2 Greater Interaction

One area for major improvement is the interaction between the conversational games and the sentence analyser. The interface implemented in the MCHINE program allows arbitrary information to pass in either direction, since the games call the analyser as a subroutine, any part of the program may call a game if required, and the analyser returns the semantic structure of the sentence as a result to the game which called it. In practice, the interactions are limited, and there are very few places where either of the two levels directs the flow of control at the other level.

In some games, the ILLOCUTION of the "result" passed up by the analyser determines which game is to be initiated next. In one or two places, the semantic routines can call CGQUERY if they lack information. The current conversation game always selects the initial state for the analyser to use. Beyond these, the analyser and the games plod forward without consulting each other much.

It should be emphasised that the practical facility for arbitrary interaction is already present in the MCHINE program (being quite trivial - both levels are progammed in the same programming language), but this is not the point. What would be interesting would be to find exactly what intractions are needed, and to try to formalise these within the given framework (or modify the framework to allow it, if this proved necessary).

Section VII.8 : Points of English Grammar

VII.8.1 Present Version

Although much of the grammar-writing that went into the MCHINE program was, as commented in Section V.0, fairly mundane, there are one or two areas where the descriptions themselves are of interest. The unified treatment of wh-clauses, the computational approach to referring expressions, the detailed investigation of tense and the description of verb phrases are the main areas where some innovations have been made.

There are obviously many areas of English as yet unexplored computationally; this Section outlines two which seem particularly crucial, and extremely difficult.

VII.8.2 Prepositions

As described in Section V.8, prepositions can be used as clues to the analyser concerning which constituents fulfil which roles in the verb-frame. This approach has certain limitations, and will not work as a general description of English prepositions, let alone as a general system of case markings. This approach is based on several assumptions about prepositional phrases, including the following :

(210)

(a) The phrase is to fill a role in the verb-meaning, and is not an optional adjunct.

(b) The syntactic properties of the verb predict that a particular preposition will be used to indicate the filler for that role.

(c) The prepositional phrase occurs after the main verb that creates the prediction.

(d) The preposition causes no change in the semantic structure of the noun phrase involved (so that the representation of "at the station" is the same as that of "the station").

All the prepositional usage in the sentences that the MCHINE program was tested on conformed to the assumptions in (210); sentences like those in (211), for example, could be analysed quite straightforwardly.

(211)

(a) You spoke to Mary.

(b) John is liked by Mary.

(c) Who did you speak to ?

It is easy to construct examples of prepositional phrases which violate these assumptions. For example, (212) violates all four conditions.

(212) Near what town did you see the monument ?

Perhaps the most serious error is (210)(d), which renders all prepositions semantically vacuous. Since we want to distinguish the meanings of (213)(a) - (c), this raises problems.

(213)

(a) Under the table.

(b) On the table.

(c) Near the table.

This difficulty can arise even if (210)(a)-(c) are fulfilled, since we want to distinguish (214)(a) and (b), even if we describe the prepositional phrase as fulfilling a role in the verb frame.

(214)

(a) I put it under the table.

(b) I put it on the table.

Perhaps what is needed is a "Prepositional SCR", which combines the meaning of a preposition with the meaning of a phrase to produce some slightly modified structure - typically, a location or time meaning. These new semantic structures could then act as arguments to higher SCRs (e.g. locative- or time- adjunct rules) which required such items. For less obvious examples (like the "to you") in (211)(a)), we would have to either create suitable semantic types (e.g. a "goal-structure") or give an entry for "to" which had vacuous effect. Prepositions would then have multiple entries, since a single preposition (e.g. "by", "to", "in") can have many semantic

effects. There appears to be no simple solution.

VII.8.3 : Conjunction

One of the most difficult areas of English grammar is that of conjunction, and there is as yet no good computational treatment of it. Probably the best known approach is that of Winograd (1972), but a more detailed treatment was included in the Thorne, Bratley and Dewar program (as described by Hamish Dewar (personal communication)). The strategy was as follows. On encountering "and", find some previous state where the set of predictions (ATN arcs) has a non-zero intersection with the current arcs; process from that state until another state is reached which has arcs in common with the state which was current when "and" was encountered; start analysing the next item using the arcs which the last two states have in common.

Even this technique might not be adequate to cover the more awkward examples, such as (215).

(215)

(a) The bulldozer drove into and completely demolished the shed.

(b) The burglar climbed up and over the wall.

Even if some improved scheme could be worked out for the surface structure of conjoined structures, it still leaves the very difficult task of defining the semantics of these constructions.

APPENDIX A

DETAILS OF IMPLEMENTED GRAMMAR

Syntactic Features

DETERM : determiner

CN : common noun

NP : noun phrase

PPN : personal pronoun

STARTNP : possible initial word in noun phrase

ADJ : adjective

RA : restrictive adjective

NRA : non-restrictive adjective

VB : verb

MAJOR : major verb

AUX : auxiliary

NTAUX : negative auxiliary

COP : part of "be"

MODAL : modal verb

ASPECT : part of "have"

DO : part of "do"

WH : wh-word

WHDET : can start a wh-phrase

WHFULL : can start an independent embedded wh-clause

WHREL : relative pronoun

TBIND : time-binder

THATS : introduces embedded question or statement

Syntactic Properties

AGREEMENT : three values, Marked on noun phrases.

INFLECTION : eleven values, marked on verbs.

OBJECT-INFORMATION : an SCR and a list of states, marked on major verbs.

Structural Combining Rules

Determiner-Head : carries sense properties from determiner to head phrase.

NP-of-NP : combines head phrase with "of" phrase.

Possessive : combines possessive with noun phrase.

R-adjective : combines adjective and head-phrase restrictively

NR-adjective : combines adjective and noun phrase to make assertion about things referred to by the phrase.

NP-Modifier : combines a verb-phrase post-modifier with a head phrase.

Subject-Complement : inserts the subject into the slot of the predicate

It+that+S : inserts an embedded clause in the meaning of another.

Time-Adjunctl : relates a time adjunct to a following clause.

Time-adjunct2 : relates a time adjunct to a preceding clause.

Time-Binder : combines a time-binder and a clause to form a time-descriptor

WH-Phrase : carries the wh-marking from a wh-determiner to a head phrase.

Relative Clause : combines a head phrase and a wh-clause to form a wh-item

NR-Relative Clause : given a term and a wh-item, makes an assertion about the term.

Role-Placement Rules : there are ten rules for constructing meanings for verb phrases from the meaning of its verb and its objects, as follows :

Active : 1 intransitive, 2 single-object, 3 double-object

Passive : 2 intransitive, 2 single-object, no double object.

APPENDIX B

SAMPLE DIALOGUES

The following conversations are based on a ludicrously simple "world", inhabited by five "PERSONS". GLADYS is the MOTHER of HIRAM and of BORIS. BORIS is the FATHER of DELIA. Both GLADYS and DELIA have "FAIR" marked on the property "HAIR", and so can be referred to as BLONDE (using the elaborated form of the latter). HIRAM is a DOCTOR. GLADYS, HIRAM and DELIA have PERSCRED 1; BORIS and EGBERT have PERSCRED 0.

All the dialogues are short, as the program soon runs out of space and becomes inconveniently slow.

MCHINE's stupidity in Example 5 results from the unsophisticated set-matching routines, which match the first example found; in the case of GLADYS SON, this happened to be BORIS, not HIRAM.

The meaning given to SURPRISE is that P SURPRISES X if X BELIEVES not-X.

In Example 3, MCHINE's failure to spot that GLADYS does not believe BORIS to be clever (i.e. that STUPID(X) => NOT(CLEVER(X))) is because the relationship between the two relations is via their elaborated forms, and the elaboration system does not make the distinction between FALSE and UNKNOWN properly when the proposition is embedded in a BELIEVE relation.

Example 1

```
:.CONVERSE;
GLADYS : HAVE YOU SPOKEN TO THE DOCTOR ?
MCHINE : PARDON ?
GLADYS : HAVE YOU SPOKEN TO THE DOCTOR ?
MCHINE : NO
GLADYS : CHEERIO
MCHINE : CHEERIO
```

* * * * *

```
BORIS : GOOD AFTERNOON !
MCHINE : GOOD AFTERNOON
BORIS : HAS THE BLONDE SPOKEN TO YOU ?
MCHINE : WHICH BLONDE ?
BORIS : GLADYS .
MCHINE : YES
BORIS : GOODBYE .
MCHINE : GOODBYE .
```

* * * * *

Example 2

```
:.CONVERSE;
GLADYS : GOOD MORNING !
MCHINE : GOOD MORNING
GLADYS : BORIS , WHO IS STUPID , LIKES EGBERT .
MCHINE : REALLY ?
GLADYS : GOODBYE !
MCHINE : GOODBYE
```

* * * * *

```
EGBERT : HI !
MCHINE : HI
EGBERT : IS IT TRUE THAT GLADYS BELIEVES THAT BORIS IS STUPID ?
MCHINE : YES
EGBERT : DOES ANYONE LIKE ME ?
MCHINE : YES
EGBERT : WHO LIKES ME ?
MCHINE : BORIS
EGBERT : CHEERIO .
MCHINE : CHEERIO
```

* * * * *

```
DELIA : GOOD MORNING !
MCHINE : GOOD MORNING
DELIA : HAVE YOU SPOKEN TO EGBERT ?
```

MCHINE : YES

DELIA : DO YOU BELIEVE THAT GLADYS BELIEVES THAT EGBERT
IS LIKED BY BORIS ?

MCHINE : YES

Example 3

```
:.CONVERSE;

GLADYS : HI !

MCHINE : HI

GLADYS : BORIS IS STUPID .

MCHINE : REALLY ?

GLADYS : IS BORIS CLEVER ?

MCHINE : NO

GLADYS : DO I BELIEVE BORIS TO BE CLEVER ?

MCHINE : DUNNO

GLADYS : DO I BELIEVE BORIS TO BE STUPID ?

MCHINE : YES
```

Example 4

```
: .CONVERSE;

GLADYS : HI !

MCHINE : HI

GLADYS : HIRAM , WHO IS CLEVER , LIKES BORIS .

MCHINE : REALLY ?

GLADYS : IS HIRAM CLEVER ?

MCHINE : YES

GLADYS : DO I BELIEVE THAT HIRAM IS CLEVER ?

MCHINE : YES

GLADYS : DOES IT SURPRISE YOU THAT HIRAM IS CLEVER ?

MCHINE : NO
```

Example 5

```
:.CONVERSE;
GLADYS : HI !
MCHINE : HI
GLADYS : HIRAM LIKES DELIA .
MCHINE : REALLY ?
GLADYS : HAS IT BEEN SAID THAT HIRAM LIKES DELIA ?
MCHINE : YES
GLADYS : DOES MY SON LIKE DELIA ?
MCHINE : YES
GLADYS : WHO IS MY SON ?
MCHINE : BORIS
GLADYS : DOES BORIS LIKE DELIA ?
MCHINE : DUNNO
GLADYS : CLOT !
MCHINE : PARDON ?
GLADYS : WHO IS DELIA'S FATHER ?
MCHINE : BORIS
```

APPENDIX C

PERFORMANCE TABLE

(See Section VI.5.2 for details)

This summary concerns only the ability of the programs to analyse sentences, so SHRDLU's mechanisms for producing sentences have been ignored. It is not clear how punctuation will affect the various programs. MCHINE is guided by punctuation only slightly (as noted previously), and the TESSA list (Soul(1975)) includes a few examples in both punctuated and unpunctuated form. SHRDLU depends on question marks to indicate questions. Since the SHRDLU parser successfully analysed all the sentences on the list in Thorne et. al.(1968), the information in the table regarding SHRDLU is related to that for TBD, but the table takes into account the samples in Dewar et. al.(1969). Soul(1975) has also provided a list of complicated noun phrases that are within the capability of TESSA, so that program can probably do better than the performance table might indicate.

The information is presented here in the following way. First, there is a list of 110 sentences, based on the sample lists from the four programs. A bracketted integer (e.g. (2)) after a sentence indicates the numbers of interpretations of an ambiguous sentence that are being considered). Each sample sentence is followed by a list of mnemonic names, with up to six entries indicating which of the given programs can analyse the corresponding sentence. There may be six entries, since both SHRDLU and MCHINE have two each - one for isolated sentence mode, and one for conversational mode. (These are indicated as SHRDLUI, SHRDLUC, MCHINEI, MCHINEC, with I denoting isolated mode, and C denoting conversational mode). The entry ALL means that all six programs could analyse the sentence successfully.

Entries preceded by a question mark indicate that it is very hard to tell from the published examples how the program would fare on this sentence. After some entries in the performance list, there follows a bracketted integer. This indicates a footnote which qualifies the simple yes-no judgement of whether the programs would succeed on that sentence. The numbered footnotes are then listed.

Performance List

1. Say something to me !

   SHRDLUI, SHRDLUC, TBD, TESSA, MCHINEI.

2. Don't utter anything !

   ?SHRDLUI, ?TBD, ?TESSA, MCHINEI.

3. Go.

   SHRDLUI, SHRDLUC, TBD, TESSA, MCHINEI.

4. Your father is clever.

   ALL.

5. You will not have been being addressed by Mary.

   SHRDLUI, TESSA, MCHINEI. (1)

6. Wont you speak to me ?

   ?SHRDLUI, MCHINEI. (2)

7. I cant not speak to Mary.

   MCHINEI (3).

8. Are you Jim ?

   ?SHRDLUI, ?TESSA, MCHINEI.

9. Mary's mother's brother likes you.

   ALL.

10. Does Clarence like Alice ?

    ALL.

11. Who likes Albert ?

    ALL.

12. Who saw you ?

    SHRDLUI, TBD, TESSA, MCHINEI.

13. Have you spoken to the doctor ?

    SHRDLUI, TBD TESSA, MCHINEI.

14. Who did you speak to ?

?SHRDLUI, TESSA, MCHINEI, MCHINEC.

15. She visited him yesterday.

SHRDLUI, TBD, TESSA. (4)

16. This cat adores fish.

SHRDLUI, TBD. (5)

17. The clever doctor is short.

ALL.

18. Who does Mary like ?

ALL.

19. With what did you hit him ?

TESSA (6).

20. When was it broken ?

SHRDLUI, SHRDLUC, TBD, TESSA. (7)

21. When ?

TESSA, MCHINEI, MCHINEC.

22. Why did you hit him ?

SHRDLUI, SHRDLUC, TBD, TESSA (8).

23. How often can you swim ?

TESSA (9).

24. What size is it ?

TESSA (10).

25. How many blocks are there ?

SHRDLUI, SHRDLUC, TESSA (11).

26. Which day will he swim ?

SHRDLUI, SHRDLUC, TBD, TESSA, MCHINEI.

27. The brother of the mother of Mary likes Gladys.

?SHRDLUI, MCHINEI, MCHINEC. (12)

28. Mary hates my teasing you.

SHRDLUI, TBD, TESSA, MCHINEI.

29. Mary likes teasing me.

SHRDLUI, TBD, TESSA, MCHINEI, MCHINEC.(13)

30. Mary likes to tease me.

SHRDLUI, TESSA, MCHINEI, MCHINEC .(13)

31. I love him to mow the grass.

TESSA, MCHINEI.

32. Liking Mary is stupid.

SHRDLUI, TBD, TESSA, MCHINEI, MCHINEC. (13)

33. Flying aeroplanes are nice.

SHRDLUI, TBD, TESSA, MCHINEI.

34. Flying aeroplanes is nice.

SHRDLUI, TBD, TESSA, MCHINEI.

35. Flying aeroplanes can be nice (2).

TBD, MCHINEI.

36. The boy who kissed the girl laughed uproariously.

SHRDLUI, SHRDLUC, TBD, TESSA, MCHINEI. (14)

37. The boy who the girl kissed laughed uproariously.

SHRDLUI, SHRDLUC, TBD, TESSA, MCHINEI. (14)

38. The boy the girl kissed laughed uproariously.

SHRDLUI, SHRDLUC, TBD, TESSA, MCHINEI. (14)

39. Fred gave the dog biscuits.

SHRDLUI, TBD, TESSA, MCHINEI.

40. Fred lost the dog biscuits.

SHRDLUI, TBD. (15).

41. Do you believe that I believe John to be clever ?

MCHINEI, MCHINEC. (16)

42. Mary, who is stupid, likes John.

MCHINEI, MCHINEC. (17)

43. Have I said that Mary likes John ?

TESSA, MCHINEI, MCHINEC.

44. When did John say he would come ?

SHRDLUI, TBD. (18)

45. He observed the man with the telescope (2).

SHRDLUI, TBD, TESSA. (19)

46. The rascal who John claimed committed the crime has escaped.

SHRDLUI, TBD, TESSA, MCHINEI. (20).

47. While John swam the boat dragged its anchor.

TBD, TESSA, MCHINEI.

48. While John swam, the boat dragged its anchor.

TESSA, MCHINEI.

49. The boat dragged its anchor while John swam.

SHRDLUI, SHRDLUC, TESSA, MCHINEI

50. Did you speak to Mary after John addressed you ?

SHRDLUI, SHRDLUC, TBD, TESSA, MCHINEI.

51. After you spoke to Gladys, did you believe that John was stupid ?

MCHINEI, MCHINEC.(21)

52. John reached the men swimming in the lake.

TESSA, MCHINEI.(22)

53. The thing that you hit him with is red.

?SHRDLUI, ?SHRDLUC, ?TBD, TESSA, MCHINEI.

54. The thing with which you hit him is red.

TESSA.(6)

55. The day when you swam was good.

TESSA.(23)

56. The size that it is is too big.

TESSA.

57. The blocks that there are in the box are red.

TESSA.(24)

58. The box in which you said that you put it is red.

TESSA.(25)

59. John is the man to beat Jack.

TESSA.(26)

60. John is the man for us to beat.

TESSA.(26)

61. John is the man beating Jack.

TESSA, MCHINEI.

62. John is the man beaten by Jack.

TESSA, MCHINEI.

63. For John to mow the grass is nice.

TESSA.(27)

64. He did it by pushing the machine.

TESSA.(28)

65. John said it when you asked.

TESSA, MCHINEI. (29)

66. I know what hit you.

TESSA, MCHINEI.(29)

67. What you said surprised me.

?TESSA, MCHINEI.

68. I asked you what you liked.

TESSA, MCHINEI.(30)

69. I asked which girl you spoke to.

TESSA, MCHINEI.

70. The story was told to me by John.

?SHRDLUI, TESSA, MCHINEI.

71. The story was told to me.

?SHRDLUI, TESSA, MCHINEI.

72. I was told the story by John.

?SHRDLU, TESSA, MCHINEI.

73. I was told the story.

?SHRDLUI, TESSA, MCHINEI.

74. Who was told the story ?

?SHRDLUI, TESSA, MCHINEI.

75. Which story was I told ?

?SHRDLUI, TESSA, MCHINEI.

76. To whom was the story told ?

SHRDLUI, TESSA.(6)

77. The box was put on the table by John.

?SHRDLUI, TESSA.(31)

78. Where was the box put ?

?SHRDLUI, TESSA.

79. Whose book did you say you wanted ?

TBD.(32)

80. When he has fixed dates he will ring us .(2)

SHRDLUI, TBD, TESSA, MCHINEI.(33)

81. A lawyer who cheats the clients he sees deserves censure.

SHRDLUI, TBD, TESSA, MCHINEI

82. Has the portrait they bought disappeared ?

SHRDLUI, SHRDLUC, TBD, TESSA, MCHINEI

83. He rolled up the bright red carpet.(2)

SHRDLUI, SHRDLUC, TBD.(34)

84. She handed John a pear and Mary an apple.

SHRDLUI, TBD.(35)

85. The plants he watered and tended flourished.

SHRDLUI, TBD.(36)

86. Are the elephant and the kangaroo he adopted obeying him ? (2)

SHRDLUI, TBD.(37)

87. What was the box put in ?

?SHRDLUI, TESSA, MCHINEI.(38)

88. She said "Rubbish".

TESSA.(39)

89. "Rubbish" she said.

TESSA.(39)

90. "Rubbish", she said.

TESSA.(39)

91. "Rubbish", said Jack.

TESSA.(39)

92. That she was not there moved us.

TESSA, MCHINEI, MCHINEC.(40)

93. It moved us that she was not there.

TESSA, MCHINEI, MCHINEC.(40)

94. I know which molasses your mother told you to buy.

?SHRDLUI, ?TBD, TESSA.(41)

95. Who did you give the ball ?

MCHINEI, MCHINEC.(42)

96. Has it been said that John likes Mary ?

MCHINEI, MCHINEC.(43)

97. Do you believe John to be liked by Mary ?

MCHINEI, MCHINEC.(44)

98. Have I asked you whether John likes Mary ?

MCHINEI, MCHINEC.(45)

99. Find a block which is taller than the one you are holding and put it into the box.

SHRDLUI, SHRDLUC, ?TESSA.(46)

100. It is funny to see how fast they get away from one another.

TESSA.

101. Jack laughed and said "Take your things and go away".

TESSA.

102. How many eggs would you have been going to use in the cake if you hadn't learned your mother's recipe was wrong ?

SHRDLUI, TBD, TESSA.

103. Pick up anything green, at least three of the blocks, and either a box or a sphere which is bigger than any block on the table.

SHRDLUI, SHRDLUC.(47)

104. I own blocks which are not red, but I don't own anything which supports a pyramid.

SHRDLUI, SHRDLUC.(48)

105. Will you please stack up both of the red blocks and either a green cube or a pyramid ?

SHRDLUI, SHRDLUC.(49)

106. Put a small one onto the green cube which supports a pyramid.

SHRDLUI, SHRDLUC, TESSA.(50)

107. Put the littlest pyramid on top of it.

SHRDLUI, SHRDLUC, TESSA.(50)

108. Is there anything which is bigger than every pyramid but is not as wide as the thing that supports it ?

SHRDLUI, SHRDLUC.

109. A "steeple" is a stack which contains two green cubes and a pyramid.

SHRDLUI, SHRDLUC.(51)

110. Call the biggest block "superblock".

SHRDLUI, SHRDLUC.(52)

Footnotes to Performance List

1. TBD includes no passives (surprisingly) and the SHRDLU dialogue only one.

2. No sign of "n't" forms in other programs.

3. No sign of double negatives in other programs (not surprisingly).

4. MCHINE has no one-word time-adjuncts in its vocabulary. Cf. notes 49, 50.

5. Most programs have ignored mass nouns. MCHINE also lacks demonstratives.

6. Only TESSA seems to have catered for the "preposition+WH" phrases.

7. MCHINE cannot handle general pronouns.

8. MCHINE has not covered "why", or causal relations in any form.

9. Only TESSA and TBD seem to have generalised "how+modifier" rules.

10. If the programs use similar "wh/how" rules for 23 and 24, that would be elegant.

11. SHRDLU allows "how many" as a special case.

12. MCHINE has only one way of handling "of+NP" (attachment to previous NP).

13. MCHINE can absorb this information, but not very subtly.

14. Only TBD and SHRDLUI allow manner adverbials, but other programs allow the relative clause construction.

15. MCHINE has no facility for compound nouns; TESSA has, but offers no samples.

16. Only MCHINE includes any deeply-embedded examples of "that" clauses.

17. Only MCHINE has attempted non-restrictive relative clauses.

18. The lack of a "that" before the embedded clause presents difficulties for MCHINE and TESSA.

19. TESSA can get one reading, but may fail to attach "with..." to "the man" in the other interpretation.

20. TESSA and MCHINE both contain the relevant rules, but have not been tried on this particular combination.

21. Only MCHINE includes questions preceded by optional adjuncts.

22. This appears to be ambiguous. TESSA gets one or both readings, but MCHINE can only get the reading where "the men" are swimming.

23. Only TESSA includes adjunct wh-words (e.g. "when", "where") as relative pronouns.

24. MCHINE lacks any "there is/there are" rules.

25. Only the "in which" should prevent MCHINE from handling this one (see note 6).

26. Only TESSA allows "to+infinitive" or "for+NP+to+infinitive" as a post-NP modifier.

27. Only TESSA has "for+NP+infinitive" as a possible NP form.

28. Only TESSA has instrumental phrases (Cf. note 14).

29. SHRDLU and TBD do not include embedded wh-clauses without antecedents.

30. The readings correspond to "I asked you the question which you liked" and "I asked you which question you liked". TESSA may not get both. There is a bug in the MCHINE grammar which causes this ambiguity to appear spuriously in other wh-clauses.

31. MCHINE cannot handle prepositional phrases as adjuncts to clauses, but could cover this example by treating "on the table" as a prepositionally marked object of "put".

32. Only TBD handles "whose"; TESSA could probably cope with the rest of the structure.

33. MCHINE and TESSA have not been tested on this example, but should manage it. TESSA might miss the reading with "fixed dates" as a noun phrase.

34. MCHINE has no rules for verb-particle constructions. Only TBD definitely gets the ambiguity.

35. It is not clear whether this sentence is analysed by the general conjunction method (see Section VII.8.3).

36. The ambiguity results from the choice of conjoining verbs or conjoining verb phrases.

37. The ambiguity results from the attachment of "he adopted" to "the elephant and kangaroo" or just to "kangaroo".

38. The dangling preposition is a problem for TBD. MCHINE could handle it only subject to the proviso in note 31.

39. Only TESSA makes any attempt to handle direct quotation.

40. Although Winograd(1972, p.52) mentions constructions like this, there is no indication that his implemented grammar includes them.

41. MCHINE cannot cope with either the mass noun or the embedded imperative.

42. This illustrates the difference between TESSA (and SHRDLU) and MCHINE concerning objects. MCHINE allows the alternative indirect object via a lexical marking.

43. MCHINE covers this sentence automatically from the grammar for sentences like 93 and 13.

44. A well-formed but inelegant sentence ?

45. MCHINE treats "whether" the same way as "that".

46. Conjunction and comparatives are difficult, and are not handled completely generally by most programs.

47. This is possibly the most complicated sentence that any program has ever handled.

48. The semantic complexity of this sentence would defeat most programs.

49. Presumably an idiom like "will you please" is handled by a "demon".

50. Only SHRDLU includes any pronoun semantics.

51. Only SHRDLU includes definitional facilities.

52. Only SHRDLU includes naming facilities.

REFERENCES

J.L.
AUSTIN(1962) "How To Do Things with Words". Oxford University Press, Oxford.

A.J.
AYER(1936) "Language, Truth and Logic". Penguin Books, 1972.

E.
BACH(1968) "Nouns and Noun Phrases". In Bach and Harms (1968).

E. R.
BACH AND HARMS (1968) "Universals in Linguistic Theory". Holt Rinehart and Winston, New York.

Y.
BAR-HILLEL(1954) "Indexical Expressions". Mind 63, pp.359-379.

D.G. J.B.
BOBROW AND FRASER(1969) "An Augmented State Transition Network Analysis Procedure". First International Joint Conference on Artificial Intelligence, Washington, D.C. Mitre Corporation, Bedford, Massachusetts.

D.L.
BOLINGER(1965) "The Atomisation of Meaning". Language 41, No.4, pp.555-573.

" (1967) "Adjectives in English: Attribution and Predication". Lingua 18, pp.1-34.

J.
BRESNAN(1970) "On Complementizers: Toward a Syntactic Theory of Complement Types". Foundations of Language 6, pp.297-321.

R.
BROWN(1958) "Words and Things". Free Press, New York.

B.C.
BRUCE(1975) "Case Systems for Natural Language". BBN Report 3010 (AI Report 23), Bolt Beranek and Newman, Cambridge, Mass. (Also in Artificial Intelligence 6.)

R.M. R.J. J.S.
BURSTALL, POPPLESTONE AND COLLINS(1971) "Programming in POP-2". Edinburgh University Press, Edinburgh.

M.K.
BURT(1972) "From Deep to Surface Structure". Harper and Row, New York.

P.W. J. T.G.
CAREY, MEHLER, AND BEVER(1970) "When do we compute all the interpretations of an ambiguous sentence ?". In "Advances in Psycholinguistics", ed. Flores d'Arcais and Levelt. North Holland, Amsterdam.

E.
CHARNIAK(1973) "Toward a Model of Childrens Story Comprehension". Memo AI-TR-266, AI Lab, MIT, Cambridge, Mass.

" (1975) "A Brief for Case". Working Paper No.22, Institute for Semantic and Cognitive Studies, Castagnola.

N.
CHOMSKY(1955) "The Logical Structure of Linguistic Theory". Microfilm, MIT, Cambridge, Mass.

" (1956) "Three Models for the Description of Language". IRE Transactions in Information Theory, pp. 113-114.(Reprinted in Luce, Bush and Galanter(1965)).

" (1957) "Syntactic Structures". Mouton, The Hague.

" (1958) "A Transformational Approach to Syntax". Third Texas Conference on the Problems of Linguistic Analysis in English. (Reprinted in Fodor and Katz(1964)).

" (1959) "On Certain Formal Properties of Grammars". Information and Control 1, No.2, pp. 137-167. (Reprinted in Luce, Bush and Galanter(1965)).

" (1961) "On the Notion 'rule of grammar' ". In Jakobson(1961). (Reprinted in Fodor and Katz(1964)).

" (1964) "Current Issues in Linguistic Theory". Mouton, The Hague. (Reprinted in Fodor and Katz(1964)).

" (1965) "Aspects of the Theory of Syntax". MIT Press, Cambridge, Mass.

" (1966) "Topics in the Theory of Generative Grammar". Mouton, The Hague.

" (1971) "Deep Structure, Surface Structure and Semantic Interpretation". In Steinberg and Jakobovitz(1971).

" (1972) "Some Empirical Issues in the Theory of Transformational Grammar". In Peters(1972).

N. M.
CHOMSKY AND HALLE(1965) "Some Controversial Questions in Phonological Theory". Journal of Linguistics 1, No.2, pp.97-138.

" (1968) "The Sound Pattern of English". Harper and Row, New York.

M.E.
CONWAY(1963) "Design of a Separable Transition-Diagram Compiler". CACM Vol.6 No.7. July 1963, pp.396-408.

A.C.
DAVEY(1974) "The Formalisation of Discourse Production". Ph.D. Thesis, School of Artificial Intelligence, University of Edinburgh, Edinburgh.

D.J.M.
DAVIES(1973) "Popler 1.5 Reference Manual". TPU Report No.1, School of Artifical Intelligence, University of Edinburgh, Edinburgh.

H. P. J.P.
DEWAR, BRATLEY AND THORNE (1969) "A program for the Syntactic Analysis of English Sentences". CACM Vol.12, No.8, pp.476-479.

R.C.
DOUGHERTY(1969) "An Interpretive Theory of Pronominal Reference". Foundations of Language 5, pp.488-519.

C.J.
FILLMORE(1968) "The Case for Case". In Bach and Harms(1968).

" (1972) "On Generativity". In Peters(1972).

C.J. D.T.
FILLMORE AND LANGENDOEN(1971) "Studies in Linguistic Semantics". Holt Rinehart and Winston, New York.

J.A. M.F.
FODOR AND GARRETT(1967) "Some syntactic determinants of sentential complexity". Perception and Psychophysics 2, pp.289-296.

J.A. M.F. T.G.
FODOR, GARRETT AND BEVER(1968) "Some syntactic determinants of sentential complexity, II: Verb Structure." Perception and Psychophysics 3, pp.453-461.

H.W. F.G.
FOWLER AND FOWLER(1906) "The Kings English". Oxford University Press, Oxford, 1973.

G.
FREGE(1892) "On Sense and Reference". In "Translations from the Philosophical Writings of Gottlob Frege", Geach and Black. Blackwell, Oxford, 1960.

M.F.
GARRETT(1970) "Does ambiguity complicate the perception of sentences ?". In "Advances in Psycholinguistics", ed. Flores d'Arcais and Levelt. North Holland, Amsterdam.

N.
GOLDMAN(1973) "Sentence Paraphrasing from a Conceptual Base". International Conference on Computational Linguistics, Pisa, September 1973.

J. P.M.
GRINDER AND POSTAL (1971) "Missing Antecedents". Linguistic Inquiry 2, No.3, pp.269-312.

A.
GROSU(1972) "The Strategic Content of Island Constraints". Working Papers in Linguistics No. 13, Department of Linguistics, Ohio State University .

M.
HALLE(1962) "Phonology in Generative Grammar". Word 18, pp.54-72. (Reprinted in Fodor and Katz(1964)).

M.A.K.
HALLIDAY(1967a) "Notes on Transitivity and Theme Part I". Journal of Linguistics 3 No.1, pp.37-81.

" (1967b) "Notes on Transitivity and Theme Part II". Journal of Linguistics 3 No.2, pp.199-244.

" (1968) "Notes on Transitivity and Theme Part III". Journal of Linguistics 4 No.2, pp.179-215.

C.G.
HEMPEL(1966) "Philosophy of Natural Science". Prentice Hall, Englewood Cliffs, N.J.

J.
HINTIKKA(1973) "Language-Games for Quantifiers". In "Logic Language-Games and Information", Hintikka. Oxford University Press, Oxford, 1973.

F.W.
HOUSEHOLDER(1965) "On Some Recent Claims in Phonological Theory". Journal of Linguistics 1.

" (1966) "Phonological Theory: A Brief Comment". Journal of Linguistics 2.

S.D.
ISARD(1974) "What would you have done if...". Theoretical Linguistics 1, No 3.

S.D. H.C.
ISARD AND LONGUET-HIGGINS(1973) "Modal Tic-tac-toe". In "Logic, Language and Probability", ed. Bogdan and Niiniluoto. Reidel

Publishing Co, Dordrecht.

R.S.
JACKENDOFF(1968) "An Interpretive Theory of Pronouns and Reflexives". Indiana University Linguistics Club, Bloomington, Indiana.

" (1973) "Semantic Interpretation in Generative Grammar". MIT Press, Cambridge, Mass.

R.
JAKOBSON(1961) Proc. 12th Symposium in Applied Mathematics. American Mathematical Society, Providence, Rhode Island.

R.M.
KAPLAN(1971) "Augmented Transition Networks as Psychological Models of Sentence Comprehension". Second International Joint Conference on Artificial Intelligence, London. British Computer Society, London.

J.J.
KATZ(1967) "Recent Issues in Semantic Theory" Foundations of Language 3, pp.124-194.

" (1970) "Interpretive Semantics vs. Generative Semantics". Foundations of Language 6, pp.220-259.

" (1971) "Generative Semantics is Interpretive Semantics". Linguistic Inquiry 2 No.3, pp.313-332.

" (1972) "Semantic Theory". Harper and Row, New York.

J.J. J.A.
KATZ AND FODOR(1963) "The Structure of a Semantic Theory". Language 39 No.2, pp.170-210.

J.J. P.M.
KATZ AND POSTAL(1964) "An Integrated Theory of Linguistic

Description". MIT Press, Cambridge, Mass.

M.
KAY(1975) "Syntactic Processing and Functional Sentence Perspective". In Schank and Nash-Webber(1975).

E. L.
KEENAN(1975) "Formal Semantics of Natural Language". Cambridge University Press, Cambridge.

J.
KIMBALL(1973) "Six or Seven Principles of Surface Structure Parsing". Indiana University Linguistics Club, Bloomington, Indiana. (Also in Cognition 2).

T.S.
KUHN(1970) "The Structure of Scientific Revolutions". Chicago University Press, Chicago.

S.Y.
KURODA(1966) "English Relativization and Certain Related Problems". In "Modern Studies in English", ed. Reibel and Schane, Prentice Hall, Englewood Cliffs, N.J.

G.
LAKOFF(1970) "Repartee". Foundations of Language 6, pp.389-422.

" (1971) "On Generative Semantics". In Steinberg and Jakobovitz(1971).

G. J.R.
LAKOFF AND ROSS(1967) "Is Deep Structure Necessary?". Indiana University Linguistics Club, Bloomington, Indiana.

J.A.
LEVIN AND MOORE(1976) "Dialogue Games: A Process Model of Natural Language Interaction". Proc. AISB Conference, July 1976. Department

of Artificial Intelligence, Edinburgh.

H.C. S.D.
LONGUET-HIGGINS AND ISARD(1970) "The Monkey's Paw". New Scientist, Vol.47, No.717, Sept.1970.

R.D. R. E.
LUCE, BUSH AND GALANTER(1965) "Readings in Mathematical Psychology". Wiley, New York.

J.
LYONS(1968) "Introduction to Theoretical Linguistics". Cambridge University Press, Cambridge.

M.
MARCUS(1974) "Wait-and-See Strategies for Parsing Natural Language". Working Paper 75, AI Lab, MIT, Cambridge, Mass.

" (1975) "Diagnosis as a Notion of Grammar". In Schank and Nash-Webber(1975).

J.D.
MCCAWLEY(1968) "The Role of Semantics in a Grammar". In Bach and Harms(1968).

" (1970) "English as a VSO Language". Language 46 No.2, pp.286-299.

" (1971a) "Pre-lexical Syntax". Report of the 22nd Annual Round Table Meeting on Linguistics and Language Studies. (Georgetown Monographs 22).

" (1971b) "Tense and Time Reference in English". In Fillmore and Langendoen(1971).

D.V.
MCDERMOTT(1973) "Assimilation of New Information by a Natural Language-Understanding System". MSc Thesis, Department of Electrical Engineering, MIT, Cambridge, Mass.

D.V. G.J.
MCDERMOTT AND SUSSMAN(1972) "CONNIVER Reference Manual". AI Memo No 259a, AI Lab, MIT, Cambridge, Mass.

G.A. S.D.
MILLER AND ISARD(1964) "Free Recall of self-embedded English Sentences". Information and Control 7 No.3, pp. 292-303.

R.
MONTAGUE(1968) "Pragmatics" . In "Contemporary Philosophy: A Survey", ed.Klibansky. La Nuova Editrice, Florence.

" (1970a) "Universal Grammar". Theoria 36, pp.373-398.

" (1970b) "English as a Formal Language I". In "Linguaggi nella societa e nella tecnica", ed. Visentini et al. Edizioni di Comunita, Milan.

" (1972) "Pragmatics and Intensional Logic". In "Semantics of Natural Language", ed. Harman and Davidson. Reidel Publishing Co., Dordrecht.

J.
MOSES(1970) "The function of function in LISP". AI-199, MAC-M-428, Project MAC, MIT, Cambridge, Mass.

B.H.
PARTEE(1970) "Negation, Conjunction and Quantifiers: syntax vs. semantics". Foundations of Language 6, pp.153 - 165.

" (1971) "On the requirement that transformations preserve meaning". In Fillmore and Langendoen(1971).

P.S.
PETERS(1972) "Goals of Linguistic Theory". Prentice Hall, Englewood Cliffs, N.J.

P.S. R.W.
PETERS AND RITCHIE(1969) "A note on the universal base hypothesis". Journal of Linguistics 5, No.1, pp. 150-152.

" (1971) "On Restricting the Base Component of Transformational Grammars". Information and Control 18, No 5, pp.483-501.

" (1973) "On the Generative Power of Transformational Grammars". Information Sciences 6, No 1.

S.
PETRICK(1973) "Semantic Interpretation in the REQUEST System". International Conference on Computational Linguistics, Pisa, September 1973.

W.
PLATH(1973) "Transformational Grammar and Transformational Parsing in the REQUEST System". International Conference on Computational Linguistics, Pisa, September 1973.

P.M.
POSTAL(1972) "The Best Theory". In Peters(1972).

R.J.
POWER(1974) "A Computer Model of Conversation". PhD Thesis, School of Artificial Intelligence, University of Edinburgh, Edinburgh.

M.R.
QUILLIAN(1969) "The Teachable Language Comprehender: a simulation

program and theory of language". CACM Vol 12, No 8, August 1969.

H.
REICHENBACH(1966) "The Elements of Symbolic Logic". Free Press, New York.

C.
RIEGER(1974) "Conceptual Memory". PhD Thesis, Computer Science Department, Stanford University, Stanford, California.

C.
RIESBECK(1973) "Expectation as a Basic Mechanism of Language Comprehension". International Conference on Computational Linguistics, Pisa, September 1973.

" (1974) "Computational Understanding: Analysis of Sentences and Context". PhD Thesis, Computer Science Department, Stanford University.

" (1975) "Computational Understanding". In Schank and Nash-Webber(1975).

G.D.
RITCHIE(1976) "Problems in Local Semantic Procesing". Proc. AISB COnference, Edinburgh, July 1976. Department of Artificial Intelligence, Edinburgh.

" (1977) "Augmented Transition Network Grammars and Semantic Processing". Paper submitted to the Fifth International Joint Conference on Artificial Intelligence.

J.
ROBINSON(1970) "Dependency Structures and Transformational Rules". Language 46 No 2, pp. 259-285.

J.R.
ROSS(1967) "Constraints on Variables in Syntax". PhD Thesis, Department of Linguistics, MIT, Cambridge, Mass. (Also Indiana University Linguistics Club, Bloomington, Indiana).

D.E. D.A.
RUMELHART AND NORMAN(1973) "Active Semantic Networks as a Model of Human Memory". Proc. Third International Joint Conference on Artificial Intelligence, Stanford University, Stanford, California. Stanford Research Institute, Menlo Park, California.

B.
RUSSELL (1905) "On Denoting". Mind 14, pp.479-493.

R.
RUSTIN(1973) "Natural Language Processing". Algorithmics Press, New York.

R.C.
SCHANK(1969) "Linguistics From A Conceptual Viewpoint (Aspects of Aspects of the Theory of Syntax) ". Memo AI-88, AI Project, Stanford University.

" (1970) "'Semantics' in Conceptual Analysis". Memo AIM-122, AI Project, Stanford University.

" (1972a) "Conceptual Dependency : A Theory of Natural Language Understanding" Cognitive Psychology Vol.3 No.4,pp.552-630.

" (1972b) "Adverbs and Belief". Memo AIM-171, Stanford University.

" (1975) "Using Knowledge to Understand" In Schank and Nash-Webber(1975).

R.C. K.M.
SCHANK AND COLBY(1973) "Computer Models of Thought and Language". Freeman, San Francisco.

R.C. B.L.
SCHANK AND NASH-WEBBER(1975) Proceedings of the Workshop on Theoretical Issues in Natural Language Processing, MIT, June 1975. Cambridge, Mass.

R.C. L.
SCHANK AND TESLER(1969) "A Conceptual Parser for Natural Language". Proc. First International Conference on Artificial Intelligence, Washington D.C. Mitre Corporation, Bedford, Mass.

J.
SCHOENFIELD(1967) "Mathematical Logic". Addison-Wesley, Reading, Mass.

R.F.
SIMMONS(1965) "Answering English Questions by a Computer: A Survey". CACM Vol 8. No.1. pp. 53-70.

" (1973) "Semantic Networks: Their Computation and Use for Understanding English Sentences". In Schank and Colby(1973).

" (1975) "The Clowns Microworld". In Schank and Nash-Webber(1975).

C.S.
SMITH(1964) "Determiners and Relative Clauses in a Generative grammar of English". Language 40 No.1, pp. 37-52.

M.J.
SOUL(1975) "Parsing and Reference Determination". PhD Thesis, Computing Centre, University of Essex, Colchester, Essex.

J.L.
STANSFIELD(1974) "Programming a Dialogue Teaching Program". PhD Thesis, Department of Artifical Intelligence, University of Edinburgh.

D. L.
STEINBERG AND JAKOBOVITZ(1971) "Semantics: an Interdisciplinary reader in philosophy, linguistics and psychology". Cambridge University Press, Cambridge.

R.P. P. B.H.
STOCKWELL, SCHACHTER AND PARTEE(1973) "The Major Syntactic Structures of English". Holt Rinehart and Winston, New York.

G.J. T. E.
SUSSMAN, WINOGRAD AND CHARNIAK (1972) Micro-Planner Reference Manual. AI Memo No.203a, AI Lab, MIT, Cambridge, Mass.

S.A.
THOMPSON(1971) "The Deep Structure of Relative Clauses". In Fillmore and Langendoen(1971).

J.P. P. H.
THORNE, BRATLEY AND DEWAR(1968) "The Syntactic Analysis of English by Machine". In "Machine Intelligence 3", ed. Michie. Edinburgh University Press, Edinburgh.

U.
WEINREICH(1966) "Explorations in Semantic Theory". In "Current Trends in Linguistics, Vol III" , ed. Sebeok. Mouton, The Hague.

J.
WEIZENBAUM(1966) "ELIZA - A Computer Program for the Study of Natural Language Communication Between Man and Machine". CACM Vol.9, No.1. pp.36-45.

Y.
WILKS(1972) "Grammar, Meaning and the Machine Analysis of Language". Routledge Kegan Paul, London.

" (1973) "The Stanford Machine Translation Project". In Rustin(1973).

" (1975) "Preference Semantics". In "Formal Semantics of Natural Language" ed. E.L.Keenan. Cambridge University Press, Cambridge.

" (1976) "Processing Case" . American Journal of Computational Linguistics.

T.
WINOGRAD(1972) "Understanding Natural Language". Edinburgh University Press, Edinburgh.

W.A.
WOODS(1968) "Procedural Semantics for a Question Answering System". Proc. AFIPS Fall Joint Computer Conference, Vol 33, pt.1.

" (1970) "Transition Network Grammars for Natural Language Analysis". CACM Vol 13, No 10, Oct 1970.

" (1973) "An Experimental Parsing System for Transition Network Grammars". In Rustin(1973).

W.A.
WOODS et al.(1969) "Augmented Transition Networks for Natural Language Analysis". Report CS-1 to the NSF, Computation Laboratory, Harvard University.

W.A. B.L. R.M.
WOODS, NASH-WEBBER AND KAPLAN(1972) "The Lunar Sciences Natural Language System: Final Report". BBN Report No 2265, Bolt, Beranek and Newman, Cambridge, Mass.

V.
YNGVE(1960) "A Model and An Hypothesis for Language Structure". Proc. American Philosophical Society 104, No.5, pp.444-466. Philadelphia.

" (1961) "The Depth Hypothesis". In Jakobson(1961).
