Computational Linguistics

Gerald Gazdar
Christopher Mellish

April 1986

Cognitive Science Research Paper

Serial No. CSRP 058

Cognitive Studies Programme,
The University of Sussex,
Brighton BN1 9QN

To appear in: John Lyons, Richard Coates, Margaret Deuchar, and Gerald Ga
eds. *New Horizons in Linguistics II*. Harmondsworth: Penguin, 1987.

Computational Linguistics

Gerald Gazdar
Christopher Mellish
University of Sussex

1. The origins of computational linguistics

1.1 Words and numbers

In the beginning, computers were all about numbers. The resources available on today's programmable calculators are not dissimilar to the resources available on the early computers. If you try to imagine getting a programmable calculator to do machine translation from Russian to English, then you may get a sense of the magnitude of the tasks that confronted the pioneers of computational linguistics (CL) in the 1950's and early 1960's. Even today, computers represent linguistic objects in nonlinguistic ways. Consider the word "GO": many computers will represent this word as a sequence of two numbers, namely 71 and 79 (the ASCII codes for the letters "G" and "O", respectively). If you give a computer a list of names and ask it to sort them, then George will precede Olga in the list that results, not because G precedes O in the alphabet, but because 71 is a smaller number than 79.

Three decades of computer science has given us programming languages that make it as easy to talk about linguistic objects like words and sentences as it is to talk about integers and addition. But as recently as 25 years ago, things were very different, and the earliest work in CL should be seen in the context of the resources that were available then.

1.1.1 Early applied computational linguistics. As those who count sheep know well, counting is a very boring task. Even the very earliest computers counted fast and accurately, and they did not get bored. They could, for example, count how many times "the" occurs in Hamlet. Some of the earliest work that came to be known as CL did exactly this kind of counting. A typical application was the attempted attribution of authorship to texts whose authorship was in doubt. In this kind of research (see, e.g. Mosteller & Wallace 1963), computers were used to compile statistics (for example, the frequency of occurrence of the word "upon") in texts whose provenance was not in doubt. These figures would then be compared with a corresponding set compiled from the disputed or unknown text, and a case made that the text had, or had not, been written by the same author.

Other work then considered to be CL involved the use of computers to derive indexes and concordances from computer-readable texts (Bott 1970). Nowadays such work no longer counts as computational linguistics, or even as an academic activity, since humble word processing programs often come equipped with sophisticated indexing utilities.

1.1.2 Early machine translation. One of the first linguistic applications of computers to be envisaged and funded was machine translation (MT). The military and intelligence communities in the US and abroad, in particular, had great hopes in MT and invested accordingly. But, despite the level of funding, the first generation of work in MT was intellectually impoverished. There was little appreciation of the fact that meaning was essentially involved, nor of the extent of ambiguity in ordinary text. The linguistic theories assumed, to the extent that any were assumed, were rudimentary. And even if they had not been, the computational resources necessary to support more sophisticated theories were simply not available. The first generation MT work amounted to little more than machine language programs for word-by-word substitution. With the wisdom of hindsight, it is unsurprising that

the results were awful. By the mid 1960's this had become very apparent, and US government agency funding for MT research dried up completely in the aftermath of a damning report on MT prepared by the National Academy of Sciences in 1966. Thus, in 1970, the precursor of the present paper reported that "so far, then, no large-scale commercial application of computers to translation exists" (Bott 1970). Fifteen years later, things have changed radically in this respect, as we shall see in the final section.

1.2 From numbers to structures

Many of the developments in CL since 1970 have arisen from a changing view of the nature of computers. For, although they are good at arithmetic, it is better to think of computers as very general symbol manipulation machines. The symbols that computers manipulate can represent numbers, or they can represent more complex objects like words, sentences, trees or networks. The machine code instructions that a computer executes perform very simple operations like shunting information from one part of the machine's memory to another or adding together two numbers. The problem with early programming languages, such as FORTRAN, was that they forced the programmer to think in terms of numbers and to specify algorithms at a level close to the actual machine code. The "high level" languages that have developed since then (for example, APL and Pascal) allow the programmer to specify instructions in terms of richer, and more problem-oriented concepts. The existence of compilers for translating from this more abstract level to the primitive instructions that the machine actually executes relieves programmers of the burden of rephrasing every idea in these terms and leaves them free to concentrate on the problems they are really interested in.

A crucial landmark in the development of CL as we know it today was the appearance of Winograd's (1972) SHRDLU program. Winograd's program was written in LISP, the language of choice for most Artificial Intelligence (AI) researchers during the 197O's. One of the major contributions of Winograd was to provide an "existence proof" - to show that natural understanding, albeit in restricted domains, was indeed possible for the computer. SHRDLU demonstrated in a primitive way a number of abilities, like being able to interpret questions, statements and commands, being able to draw inferences, explain its actions and learn new words, which had not been seen together before in a computer program. SHRDLU was a considerable achievement for one person, and one that would have been impossible without the availability of high level programming languages.

1.3 Towards declarative formalisms

Computer programming is the activity of giving a computer a precise and detailed set of instructions for how to perform some task. Certainly a lot of knowledge that humans have seems to be represented in this procedural way. For instance, the chances are that you think of tying your shoelaces in terms of the sequence of actions that you would have to go through to do it. In fact, you may find it hard to describe this activity adequately without actually going through the motions. Other human knowledge, on the other hand, seems to be less dependent on how it is to be used.

For instance, the knowledge that Paris is the capital of France could be used in a number of different ways in different contexts. If we look at a computer program that performs some task involving natural language, we might well ask "what knowledge does this program have of grammar? of word meanings? of the application domain it operates within?". The trouble is that this knowledge may well be implicit in the instructions that specify how to perform the specific task. The procedural representation suggested by computer implementation can thus get in the

way of a theoretical characterisation of the task and of what knowledge is required to perform it. A way out of this problem is to represent the rules and principles themselves as symbolic structures to be manipulated by a program.

The idea of having programs working with explicitly represented, inspectable. rules has been very successful in applications of Ai, where these rule based systems have been developed for tasks like medical diagnosis and the interpretation of geological measurements. Programming languages (such as OPS5) are now emerging which allow the programmer to simply specify the rules and (to some extent) to leave the actual processing decisions up to the machine. One especially exciting development is the rise of "logic programming" languages, of which Prolog, due to Alain Colmerauer - himself a computational linguist, is the most well-known (Clocksin and Mellish 1981). The idea of these languages (still to be completely realised) is for programmers to simply describe their problems in logic, expressing what is to be done rather than how.

In CL. an example of this might be a programmer specifying a grammar in much the same way as a theoretical linguist. With this representation, the computer would then be able both to generate example sentences allowed by the grammar and to determine whether given sentences were indeed grammatical. As yet, logic programming languages can only produce this kind of performance for very simple grammars, but a great deal of effort is being spent on improving them. Attempts are even being made to design new kinds of computers that will support these languages better than conventional ones (Taki et al. 1985).

2. The imposition of structure

2.1 Introduction

As every chapter in this book makes plain, linguistic objects are structured objects. But they do not wear their structures on their sleeves, as it were. A grasp of the meaning of a sentence depends crucially on an ability - likely to be unconscious for a native speaker of the language in question - to recover its structure. A computational device which infers structure from grammatical strings of words is known as a "parser", and much of the history of CL over the last twenty years has been occupied with the design of parsers.

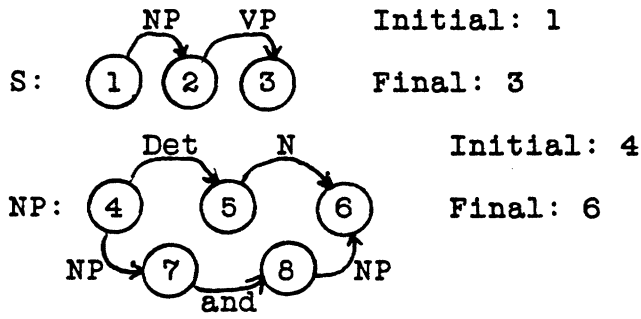
A modern parser can be thought of as a device which takes (i) a grammar, and (ii) a string of words, and gives you either a grammatical structure imposed on that string of words, if the string is grammatical with respect to to (i), or nothing, if it is not. Conceptually, the parser and the grammar are quite distinct kinds of things: a grammar is simply an abstract definition of a set of well-formed structured objects, whereas a parser is an algorithm (or set of instructions) for arriving at such objects.

Whereas in early parsers the grammars employed were inextricably interwoven with the computer programs implementing the parsing algorithm, the trend towards declarative (as opposed to procedural) formalisms has led many modern computational linguists to clearly separate these two components. Nevertheless, the manner in which grammars are represented does play a significant role in an overall parsing system.

22 Procedural formalisms: RTNs and ATNs

Recursive transition networks (RTN#s) are a way of specifying grammars weakly equivalent to the phrase structure grammars discussed in the chapter on generative grammar. Here is part of a very simple RIN grammar:

(1)



As can be seen, an RTN consists of a collection of networks, each of which is labelled with the name of some syntactic category. The networks themselves consist of a collection of "states" represented by small circles connected by directional "arcs" represented by curved arrows. The states are given arbitrary names (integers in our example above), whilst the arcs are labelled with the names of syntactic categories. Each network has, in addition, an indication of its initial and final states. The RTN can be interpreted as a collection of maps that permit you to find your way through the grammatical expressions of the language.

Suppose we wish to determine whether a given string, for instance "the man ate a frog", is a grammatical sentence of English. Then we must start in the initial state (state 1) of the S-network above and attempt to work our way through the string and the RTN simultaneously in such a way that when we get to the end of the string we are in the final state (state 3) of the S-network. Since there is an arc joining state 1 to state 2 labelled NP, we can get from state 1 to state 2 of the S-network if we can find a noun phrase at the beginning of our string of words. To find out if there is an NP there we need to invoke the NP-network and see if we can work our way through that. The first word is "the": we look this up in the dictionary and discover that it is of category Det(erminer). This means that we can move from the initial state (state 4) of the NP-network to an intermediate state (state 5). Since the next word ("man") is a noun, we can move to the final state (state 6) of the NP-network and hence "pop" back up into the middle of the S-network, namely at state 2. Now we have to see if we can get from there to state 3, going via the VP-network.

The RTN formalism used to appeal to computational linguists precisely because of the naturally procedural interpretation. RTN's themselves, however, have been overshadowed in computational linguistics by an elaborated version of the formalism known as the augmented transition network (ATN) which originates in the work of Thorne, Bratley, and Dewar (1968), but which achieved major prominence in the field thanks to Woods (1970). An ATN is simply an RTN that has been equipped with a memory and the ability to "augment" arcs with actions and conditions that make reference to that memory. ATN-based parsers were probably the most common kind of parser employed by computational linguists in the 1970's, but they have fallen out of favour in recent years. This is largely because the "augmentations" destroy the declarative nature of the formalism and because a parser using an ATN is limited in the control strategies it can employ.

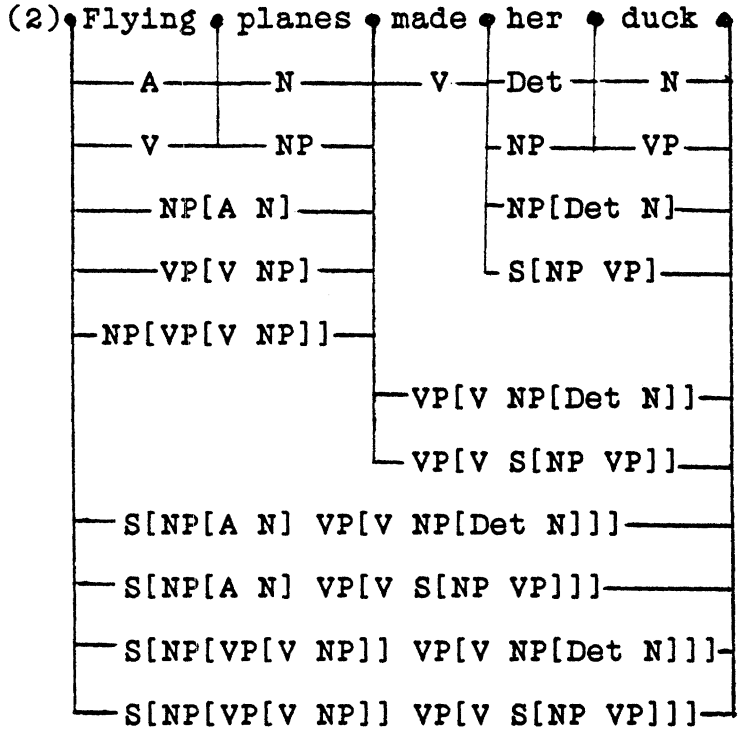
2.3 Charts as data structures

Earley (1970), and Cocke, Kasami and Younger (1967) proved mathematically that a sentence of a language described by a phrase structure grammar was parsable in a time that was, in the worst case, proportional to the cube of the length of the sentence. This relatively modest parsing time requirement relies on the fact that phrase structure parsers are able to employ a "well-formed substring table" to avoid

wasteful reduplication of effort.

Kay's (1973) practical implementation of the "well-formed substring table", known as a "chart", was very successful, and chart parsers, as they are known, have now become one of the basic tools of modern CL.

A chart is basically a data structure in which the parser records its successful attempts to parse subconstituents of the string of words. Here, for example, is a representation of the "well formed substrings" that a chart parser might build up for the quadruply ambiguous English sentence "flying planes made her duck":



Once the parser has recorded the presence of a constituent in one part of the string, it never needs to look for the same kind of constituent there again. This represents a significant improvement on the backtracking algorithms used in most ATN systems. The ability of the chart to record in addition the current goals of the parser leads to the possibility of implementing flexible and mixed control strategies (Kay 1984, Thompson & Ritchie 1984).

2.4 Declarative formalisms: DCGs, GPSGs, and LFGs

In the last few years, computational linguists have shifted from procedural grammar representations to declarative ones. In part this reflects analogous moves within theoretical linguistics itself, but it is also a consequence of the trends within AI discussed in the first section.

Prolog is an inherently declarative language and so it is not surprising that one of the first of the new breed of declarative grammar formalisms emerged from that language. Definite Clause Grammars (DCG's) were developed by Pereira and Warren (see, e.g., 1980) from ideas of Colmerauer (1978) and have been quite widely used within the Prolog community. A DCG is essentially a phrase structure grammar annotated with Prolog variables which maps straightforwardly into ordinary Prolog code. This total compatibility with Prolog is the major attraction of DCG's. Even

though they look like grammars, and are in fact grammars, they can be used as parsers directly, given the way that Prolog works. However, using them in this way can prove inefficient since Prolog does not, by itself, employ any analogue of the well-formed substring table or chart discussed in the preceding section. The DCG formalism is provably powerful enough to describe languages (both natural and artificial) of arbitrary complexity. It is not. however, especially well-adapted for providing elegant accounts of some of the complexities that show up in natural languages (e.g., the unbounded dependency type of construction discussed in the chapter on generative grammar), although this has been ameliorated in some subsequent extensions of the formalism (Pereira 1981).

Theoretical linguistics and CL have seen a quite unprecedented convergence of interest in the 1980's. One of the fruits of this mutual interest has been the essentially collaborative development of declarative grammatical formalisms of which the best known are Generalized Phrase Structure Grammar (GPSG) and Lexical Functional Grammar (LFG), which were discussed in chapter . These formalisms both owe much to Kay's (1979) computational linguistic work on Unification Grammar. LFG is the joint product of a theoretical linguist and a computational linguist. GPSG is the product of a group of theoretical linguists several of whom have worked, for a number of years, as consultants on a large commercial computational linguistics project, which, along with other work of a computational linguistic character (such as Thompson 1982, and Pereira and Shieber 1984), has had a significant impact on the theoretical framework they have developed (see Shieber 1986 for a thorough introduction to unification-based grammar formalisms).

In the light of this convergence, it is perhaps not surprising that many recent computational linguistic projects have chosen to use GPSG or LFG (or some modification of one or the other) as their grammatical formalism. Examples include Bear and Karttunen (1979). Evans (1985), Gawron et al. (1982), Kilbury (1984), Phillips and Thompson (1985), Pulman (1984), Reyle and Frey (1983), Rosenschein and Shieber (1982), Schubert and Pelletier (1982). Shieber (1984). and Yasukawa (1984).

2.5 Ambiguity and determinism

Ambiguity is arguably the single most important problem in CL. Natural languages are riddled with ambiguities at every level of description from the phonetic to the sociological, and in this respect they differ radically from formal languages such as the propositional calculus or LISP. Yet as users of natural languages we are blithely unaware of this pervasive ambiguity - it only comes to our attention in the guise of such linguistically marginal phenomena as puns, misunderstandings, and contested libel suits.

Consider the four grammatical parsings of "flying planes made her duck" shown in the chart in (2) above. One reading is distinctly improbable - it requires us to imagine a female person intermittently lowering her head to avoid a projectile whenever she is engaged in the activity of flying an aeroplane. Two other readings are semantically quite bizarre, and so only one reading remains, namely that in which she ducks in order to avoid aeroplanes that are flying overhead. We will indicate later some of the semantic mechanisms used by computational linguists to tackle the problem of ambiguity. This example is a *globally* ambiguous sentence - the entire string of words has more than one structure associated with it. Much of the ambiguity problem in CL arises, however, from *local* ambiguities, that is ambiguities that exist only in some subpart of the whole. Consider the following example:

(3) The student whose dog chased Fido hates cats.

This sentence is not globally ambiguous and it says nothing at all about whether or not Fido hates cats. But if you restrict your attention simply to the last three words then you might well hypothesise the existence of a sentential subconstituent made up of "Fido hates cats". This is a local ambiguity, and such local ambiguities can sidetrack natural language parsers into many timewasting investigations of possibilities that do not work out in the end. Marcus (1980) initiated a line of research, continued by Shieber (1983), Tomita (1984), Pereira (1985) and others, into so-called "deterministic parsers" which are parsers that are not fooled into pointless activity by local ambiguities of this kind (see also Thompson & Ritchie 1984). The human parser seems to be adept at overcoming most, but not all. cases of local ambiguity. One of the questions that work on deterministic parsers seeks to answer is why "garden path" sentences such as (4) cause such perceptual problems, whereas sentences like (5) do not.

(4) The horse raced past the barn fell.

(5) The horse ridden past the barn fell.

3. The representation of meaning

Language understanding involves relating linguistic forms to meanings; language generation involves the opposite. We can certainly represent linguistic forms in the computer, but what about meanings? All that a computer manipulates is formal symbols. How can we say that one symbolic structure corectly represents the meaning of a sentence, whereas another does not?

3.1 Procedural semantics

There have been many different ideas about how meaning can be represented in humans and machines. One possibility involved considering the meaning of an utterance to be a procedure - that is, a set of instructions to achieve what the speaker wants. Such an idea fitted in naturally with the Al concept of the mind as a computational device following rules and instructions of some kind. The competent hearer would be able to construct this procedure, and could then decide whether or not to actually run it.

According to the simplest versions of procedural semantics, the meaning of a command is a procedure to carry out the required action, the meaning of a question is a procedure to find the answer, the meaning of a statement is a procedure to add the new information conveyed to the hearer's model of the world, and so on. However, there are profound phiosophical problems with any simple form of procedural semantics (see Johnson-Laird 1977, 1978, Fodor 1978. and Woods 1981 for some discussion).

3.2 Network representations

Given the difficulty of accounting for all possible language uses in the same formalism, most logicians and Al workers have concentrated on representing the meanings of simple statements about the world. It is at least arguable that other uses of language involve statement meanings as a component of their meaning. The problem of representing the meanings of statements about the world merges into that of knowledge representation in general, and this is a central concern for Al research.

Some early formalisms for knowledge representation, based on the idea of networks, were motivated by intuitive psychological considerations. Intuitively, we think of the world in terms of concepts; when we hear something new, we may discover new concepts, or we may discover new relationships between existing concepts.

Thus we can think of representing knowledge in terms of a graph, where the nodes represent the concepts and the links between them represent the relationships that are known. Such semantic net systems are designed to facilitate certain restricted kinds of inference, such as inheritance of properties from node to node.

One network-based notation that has provoked a lot of attention is Schank's (1972) Conceptual Dependency. This was explicitly intended as a psychological model of how people represent the meaning of sentences. In 1963, Katz and Fodor had promulgated the idea of having a fixed set of primitives out of which all meanings were to be constructed. Schank continued in this tradition by proposing a set of (about) 11 primitive actions, for instance PROPEL (apply a force to) and MTRANS (mental transfer of information). The idea was that these could be combined to express any event in the world. Conceptual Dependency was designed to be independent of its use with any particular natural language, in order to allow it to act, for example, as an "interlingua" for translating between languages. Associated with each of Schank's primitive actions was a set of underlying argument positions which could be filled differently for each instance of the action. For example, every time an instance of PROPEL is represented there has to be an "actor" (the performer of the action), an "object" (the thing which the force acts on) and a "direction" (where the activity is coming from and going to).

Nowadays, many of the ideas in the early network-based systems have been subsumed in systems with more structure. Unadorned semantic networks have no way of expressing the fact that certain concepts and relationships should be grouped together into larger chunks. Thus they cannot explain how one might "focus" on a particular group of things in a discourse or how a reader might have sensible expectations about what is coming next after recognising a familiar situation. Some of the systems that have emerged in response to this need are partitioned semantic networks (Hendrix 1979), KL-ONE (Brachmann 1978) and various systems based on Minsky's (1975) "frames". One of the latter ("scripts") will be discussed in section 4.

3.3 Logicism

Semantic network systems have been criticised by a number of researchers for being inadequately defined and for failing to capture essential distinctions (see, e.g., Brachman 1985). Indeed, until recently, little effort was spent on producing formal theories of what these networks actually meant and how exactly one should go about expressing any given knowledge in them (but see, now, Touretsky 1984, for example). A rival class of knowledge repesentation languages, which has received a great deal of formal attention of this kind, is that of logics, in particular, first order logic.

First order logic has been used as a representation language in AI right from the beginnings of the subject. Thus, Woods' (1973) LUNAR program for answering questions about lunar rock samples, which was produced at roughly the same time as SHRDLU, used a logical representation for the meanings of questions. The current interest in logic programming systems is just one symptom of the growing realization of the importance of logic. Indeed, some researchers argue that all the existing network notations (insofar as they are well-defined) are equivalent to, or weaker then, first order logic, and that we should therefore dispense with them. On the other hand, others argue that logic is too neutral a representation language, not tuned to the kinds of meanings that natural language conveys.

The idea of representing natural language meanings in logic is one aspect of a vein of "logicism" that is present in much of the recent work on natural language processing. The aim of this enterprise is to produce a formal account of how

natural language conveys meaning by drawing on work in formal semantics and the philosophy of language. Almost all computer models of language understanding make use of some concepts from this work, for instance the idea that meaning can be obtained compositionally, an idea which goes back to the work of Frege in the 1890's (Frege 1962).

The compositionality principle, also known as "Frege's principle", maintains that the meaning of any phrase can be obtained by some operation on the meanings of its parts. So, given a structural description of a sentence, one can work out what the sentence means by first finding the meanings of the individual words, then combining these together to construct the meanings of small phrases, then combining these to construct the meanings of larger phrases, and so on until the meaning of the whole sentence is formed. The work of the philosophical logician Montague (1974) is taken by many as an important demonstration of the utility of the compositionality principle in the analysis of natural language. Montague showed that many previously problematic semantic phenomena in natural language were amenable to formal treatment within a strictly compositional regime. Montague's account was not computational in form, and left important computational issues, such as the role of inference and the representation of word meanings, untouched. Nevertheless, his methodology has proved an inspiration to many workers in CL (e.g. Hobbs and Rosenschein 1978, Nishida 1983).

3,4 Ambiguity and selectional restrictions

As we have seen, a processing account of language comprehension must provide some explanation of how ambiguity is resolved. This involves specifying extra mechanisms for filtering out syntactic or semantic analyses that are inappropriate. Many computer programs for manipulating natural language have made use of selectional restrictions, originally proposed by Katz and Fodor (1963).

Selectional restrictions were introduced as a way of accounting for the fact that, for instance, a sentence as a whole can be unambiguous even though individual words may have several alternative possible senses. The theory was based on the idea that with each sense of a word one could associate semantic markers specifying features of the meaning as well as conditions on the features of word senses that could combine with it. So, for instance, one possible sense of the word "spirit" (alcoholic fluid) would have the marker "physical object", whereas another sense (vital principle) would not. The adjective "yellow", when describing a colour, requires that a noun that it qualifies have the marker "physical object". Another sense of "yellow" (cowardly) requires the noun to have the marker "animate", but neither of the senses for "spirit" considered here has this marker. Thus the phrase "yellow spirit" can be shown quite formally to have only one possible sense here ("yellow-coloured alcoholic fluid"), rather than four (to keep things simple, we are ignoring the other possible senses of "yellow" and "spirit").

Within a processing framework, it is possible to extend the basic Katz and Fodor idea so that, for instance, semantic marker analysis causes a syntactic analysis to be rejected if no corresponding semantic readings can be found, or so that the semantic tests operate on the referents of phrases rather than simply the word senses (see chapter XX, for the sense/reference distinction).

The use of semantic markers is a crude device that can nevertheless be computationally effective, especially within restricted domains. It is. however, only the tip of a very large iceberg of possible ways in which knowledge can resolve ambiguity. This is well illustrated by considering an example due to Winograd (1972). If somebody says "the city councillors refused the demonstrators a permit because they feared violence", most people will understand without apparent

difficulty that "they" refers to the councillors. If instead the sentence is "the city councillors refused the demonstrators a permit because they advocated revolution", most people will decide that "they" means the demonstrators. Knowledge of some kind is enabling people to resolve this ambiguity, and yet this is considerably more subtle than anything that could be expressed by markers.

4. The Role of Knowledge

We have seen that resolving ambiguity in sentences requires an understander to have general knowledge of the world. Understanding the significance of a vague utterance expressed in context also requires knowledge. Thus a theoretical model of language understanding is not complete without a model of knowledge representation and retrieval, and we cannot construct a robust understanding computer without providing it with an encyclopaedic knowledge of the world. These are rather pessimistic conclusions, but they need not prevent us from continuing with the theoretical study of language or indeed from constructing useful computer programs which operate in limited domains. They do, however, suggest that we must try to classify in a rigorous way the kinds of knowledge that guide a language user and codify enough of it to ensure that our overall models are realistic.

4.1 World knowledge

How can knowledge of the world guide a reader to correctly interpret a partially ambiguous piece of natural language encountered in context? Where a sentence is ambiguous, world knowledge must indicate that some possible readings are to be prêferred over others. At the simplest, it may allow some readings to be rejected because they are presupposing physically impossible situations, but even this will be inadequate if the writer is using metaphor or has established a context where the normal physical laws do not apply. A more powerful idea is to concentrate on the power of a text to create expectations in the mind of a reader, and to think of the reader as preferring readings which are in accord with these expectations. A simple-minded model of expectation-based understanding is provided by Schank's "script applier mechanism" (SAM, see Schank and Abelson 1977). This works on the principle that many events in the world (especially those involving humans, such as going to a restaurant or using public transport) proceed in a stereotyped way. Therefore, when we read about them, at any point there is a limited number of things that we expect to happen next. We can represent knowledge of prototypical events as sequences of expected actions in "scripts" for the events. In order to come up with sensible expectations, a robust natural language understander will need to have a large number of such scripts. This introduces many questions - how can an understander decide which script is appropriate for a given situation? whether the current script is no longer adequate? how to handle deviations from expected behaviour? In the more interesting stories about human beings, one frequently cannot understand them in terms of stereotyped situations, but rather must reason at a lower level about the goals and plans of the participants in order to generate expectations (Wilensky 1983).

4.2 Goals, speech acts and beliefs

If we are reading a story, then, we will be more successful if we can understand the goals and plans of the various characters. We must also, however, bear in mind to some extent the objectives of the writer. Knowledge of the other participant in a communicative exchange is even more important if the medium is speech, as spoken utterances are frequently elliptical, oblique and subject to context-peculiar interpretation. Consider, for example, the following:

(6) A: Excuse me, do you know if there is a newsagent near here?
B: It's early closing today.
A: What about Brighton?
B: Johnson's in Ship Street is open until 5.

It is hard, if not impossible, to come up with any sensible interpretation of this exchange between two people if one ignores the fact that the people are communicating and cooperating. Likewise, a hearer who can ask "why is she telling me this?" and come up with a reasonable hypothesis is going to be in a better position to understand than someone who cannot. An intelligent hearer must regard utterances in the same way as any other actions performed by an intelligent being. That is, an utterance is an action that, given certain preconditions, will achieve effects planned for by the speaker.

The notion of planning has always been of great interest to AI workers, and there is now the possibility that planning work in other domains (such as the movement of robot arms) will be applicable to natural language understanding (Allen 1983) and production (Cohen 1978).

Unfortunately for CL, the plan associated with an utterance is not usually transparently marked in the syntax. Often the intention is conveyed in a form of words that superficially suggests a different intention. A classic example is an utterance like "can you pass the salt?", which looks as if it ought to be a question but which is normally intended as a request, or an utterance like "it is rather cold in here" which looks like a simple assertion of fact but may well convey intentions having to do with getting a window closed by the addressee. It is hoped that these so-called "indirect speech acts" can be explained in a unified framework that treats utterance as actions involving the beliefs and goals of the participants and assumes principles of cooperative behaviour on behalf of speaker and addressee. That is, a successful utterance will cause a change in the addressee's beliefs or goals. The cooperative addressee will attempt to understand the relevance of this change for the speaker's overall plan and will hence establish a suitable cooperative response. In order to adequately appreciate the effects of utterances, then, it is necessary to be able to reason about beliefs.

In general, one needs to reason not only about the speaker's beliefs and the addressee's beliefs, but also about the speaker's beliefs about the addressee's beliefs, the addressee's beliefs about the speaker's beliefs about the addressee and so on. For instance, to be be successfully ironic, one must be expressing some proposition that one believes to be false. However, if that was all that was required, then simple lies would be instances of irony. But they are not. In addition, the speaker must believe that the addressee believes it is false and the speaker must believe that the addressee believes that the speaker believes it is false.

4.3 Conversation and discourse structure

Once we start to consider either natural language conversations of more than one utterance or extended discourses (such as this chapter), it becomes clear that there is more structure to be found than that in the sum of the utterances, even if one takes into account the goals of the participants. That is, people adhere to certain rules and conventions about how conversations and discourses should be organised (when turn-taking should happen in the former, for instance).

From a computational point of view, the identification of these rules and conventions can serve not only to enable computers to produce more acceptable conversational behaviour but also to control the inferences that must be made for successful

understanding. For example, if specific linguistic devices are used in a language for indicating the beginning (e.g. "by the way") and end (e.g. "anyway") of a digression or for indicating how the speaker or writer is shifting their focus from one topic to another, this is information that a computer program should pick up.

An example from the work of Grosz (1978) serves to illustrate the point. Early computational approaches to the interpretation of pronouns used simple heuristics based on recency. So such a program, on encountering a pronoun, would prefer to interpret it as referring to an item in the previous sentence rather than the one before that, and so on. One of the naturally-occurring dialogues that Grosz recorded, however, contained an example of what appears to be a pronoun referring back to something last mentioned 60 utterances (30 minutes) previously. Grosz accounts for this in terms of the discourse having an underlying structure that is tree-shaped, rather than linear. Thus although the reference is back to an object which is chronologically "distant", it is to something relatively "close" in the underlying discourse structure (see Hirst 1981 for a useful critical survey of this topic).

5. The emergence of a new technology

5.1 Current machine translation

There is a sense in which MT, that dream of the 1950's, is now a reality, and there is a sense in which it works. It is a reality because there are commercial software houses selling MT programs and customers in the marketplace buying those programs. And the programs are being used: in 1984 half a million pages were translated by machine (Slocum 1985).

Current MT programs work in the following sense: some of the companies that use them save money on translation. They do not "work" when that is taken to mean "consistently produce translations from raw text that are indistinguishable from those produced by a skilled human translator with unlimited time on their hands". They all require either pre-editing of the input into some form they can accept, or post-editing to revise or replace the passages that they could not handle, or both. However, translation by humans is a very expensive and time consuming enterprise, and, in practice, it also invariably requires post-editing. Anything which speeds the process up and reduces the number of hours that a skilled translator or post-editor has to spend with the material stands a chance of saving money.

One example of such a "working" MT program is METEO, which has translated weather forecasts from English into French on a regular basis (11,000 words a day) in Canada since 1977. 80% of what it translates is translated correctly without any intervention from human translators (Slocum 1985).

At the beginning of the 1980s, the European Economic Community saw fit to invest $25,000,000 in a multilingual MT research project called EUROTRA. Although, this is the largest single MT project in the world, the Japanese government is almost certainly spending more than this on basic research in the area - far more than the US or any single European country.

5.2 Database front ends

A computer database is a store of information about some domain. Thus a company might have a database listing all of their employees, their age, their current job, the date they were hired, the number of their office, their home address, the equipment they have at their disposal, the name of their immediate superior, and so on. Given such a store of information, one needs a way of accessing it, and, until recently, this has been done in one of two ways: either (i) the system interrogates you

(usually via a menu of options) to find out what you want to know, or (ii) you interrogate the system by asking it questions in a database query language. The first strategy is simple and does not require that users have any special training or knowledge of the structure of the database. But it is slow, and, more importantly, very inflexible. The second strategy is fast and flexible, but requires users to become familiar with the database and to learn a formal language that is usually as complex as a programming language.

Programs which translate a (very restricted subset of a) natural language into a database query language are known as natural language front ends. They promise to provide the best of both strategies via the familiarity and flexibility of a natural language. A number of such front ends are now available commercially (see the survey in Johnson 1985).

5.3 Grammar development systems

The future of machine translation depends, minimally, on the availability of formal grammars for both the source and target languages. Likewise, the spread of natural language front ends beyond the confines of the English speaking world depends on the availability of grammars for languages other than English. In these circumstances there is a need for computational tools that will permit people with some linguistic training to develop formal grammars for a range of different languages.

The first generation of such tools has already emerged: the ProGram system (Evans 1985) and "GPSGP" (Phillips & Thompson 1985) provide grammar development systems for GPSG, whilst the "Grammar Writer's Workbench" provides similar facilities for LFG. These systems allow linguists to build up a large grammar gradually, testing the consequences of each rule as they go along, experimentally parsing test sentences, checking if the grammar correctly excludes ungrammatical strings, and examining the semantics to see what ambiguities, if any, have been detected, and what meanings have been assigned.

Even with such tools, a formal grammar for even a fragment of a natural language represents a massive investment of skilled labour. This cost could be radically reduced if grammars could be constructed in a largely automatic fashion on the basis of written texts, that is to say, by grammar acquisition programs. This is an area fraught with profound theoretical problems: under one idealisation it is possible to demonstrate mathematically that no such programs could ever achieve their goal (Gold 1967). On the other hand, in at least one very circumscribed domain (numeral systems), there has been demonstrable progress (Power and Longuet-Higgins 1978). At the very least, we can soon expect to see the appearance of programs that induce the grammatical idiosyncrasies of particular words on the basis of large corpora. And if, as many theoretical linguists of different persuasions now suppose, languages differ much less in their syntax than they do in their lexicons, then there are some grounds for cautious optimism in this area.

5.4 Intelligent text processing

Readers familiar with the current generation of word processing software know that these indispensable devices allow one to issue commands (usually in a very terse notation) that correspond to, for example, 'find all instances of *competant*' or 'copy lines 15 through 72 to another file'. But these programs do not allow one to issue commands like, for example, 'find all references to contemporary politicians' or 'copy all the material on Ireland to another file', or even 'correct all the misspellings'. And yet such instructions are routinely given to secretaries. However, even to correct spelling as intelligently as a human being, it is necessary to understand the

meaning of the text, and sometimes understanding will be needed even to detect the misspelling in the first place.

We can expect soon to see syntax-based spelling correctors replacing the word-based systems that are currently available. Spelling correctors based on a real understanding of the texts they work on are still, however, a long way off.

5.5 Articulate expert systems

An expert system is a computer program which offers advice. It may assist or even replace a human expert. The advice itself may require almost nothing in the way of natural language processing, thus a medical expert system for disease diagnosis might just produce the name of a disease, together with the probability that the patient has that disease. As this example may illustrate, however, humans are not, in general, satisfied merely with advice, no matter what degree of confidence they have in the expert (and with computerised experts, this may not be high). They want to know what the basis of the advice is, in other words, they want an explanation of how the expert arrived at the advice given. For any class of problems that are hard enough to make it worth constructing an expert system. there will typically be a vast number of bases for the advice given, and so an articulate expert system must be able to express itself in a correspondingly vast number of different ways. This makes the incorporation of potted scripts for each of the possible explanations quite impractical. The program needs to be able to synthesise a comprehensible explanation from scratch on the basis of the particular chain of inferences that led it to the advice it gave. To be able to this, it needs to have a very good grasp of the syntax and semantics of the fragment of a natural language that will be used in its explanations. As expert systems become more common, explanation generation is becoming an increasingly important application area for CL. Many of those working in the area of expert systems regard the explanation-giving facility as the key to public acceptance of expert systems. The explanations also have an educational value, of course, for a junior doctor who consults an expert system for diagnostic purposes can learn from it in much the same way that he or she might learn from a consultant.

6. Conclusion

As we have seen, CL has changed a lot since Bott's survey fifteen years ago (Bott 1970). Partly because of our new views of the role of the computer, and partly because of the plummeting cost of powerful machines, CL now has much more ambitious aims than it had in 1970. and, as a consequence, there are some profound theoretical problems waiting to be solved. Nevertheless, the new ideas, even in their present state, are finding practical application.

References:

Allen, James. 1983. Recognising intentions from natural language utterances. In Michael Brady and Robert C. Berwick, eds. *Computational Models of Discourse*. Cambridge, Ma.: MIT Press, 107-166.

Bear, John and Lauri Karttunen. 1979. PSG: a simple phrase structure parser. *Texas Linguistic Forum* 15, 1-46.

Bott, M. F. 1970. Computational linguistics. In John Lyons, ed. *New Horizons in Linguistics*. Harmondsworth: Penguin, 215-228.

Brachman, Ronald J. 1978. *A Structural Paradigm for Representing Knowledge*. Cambridge, Ma.: Bolt, Beranek and Newman, Inc.

Brachman, Ronald J. 1985. "I lied about the trees". *AI Magazine* 6.3, 80-93.

Clocksin, William, and Christopher Mellish. 1981. *Programming in Prolog*. Berlin: Springer-Verlag.

Cohen, Philip R. 1978. On knowing what to say: planning speech acts. *Technical Report* 118, Dept of Computer Science, University of Toronto.

Colmerauer, Alain. 1978. Metamorphosis grammars. In L. Bolc, ed. *Natural Language Communication with Computers*. Berlin: Springer Verlag, 133-189.

Earley, Jay 1970. An efficient context-free parsing algorithm. *Communications of the ACM* 14, 453-460.

Evans, Roger. 1985. ProGram: a development tool for GPSG grammars. *Linguistics* 23 213-243.

Fodor, Jerry A. 1978. Tom Swift and his procedural grandmother. *Cognition*. 6, 229-247.

Frege, Gottlieb. 1962. *Funktion, Begriff, Bedeutung: Funf logische Studien*, (Gunther Patzig ed.). Gottingen: Vandenhoeck and Ruprecht.

Gawron, Jean Mark, Jonathan King, John Lamping, Egon Loebner, Anne Paulson, Geoffrey Pullum, Ivan Sag and Thomas Wasow. 1982. The GPSG linguistics system *Proceedings of the 20th Annual Meeting of the Association for Computational Linguistics* 74-81. Also distributed as *Hewlett Packard Computer Science Technical Note* CSL-82-5.

Gold, E. Mark. 1967. Language identification in the limit. *Information and Control* 10 447-474.

Grosz, Barbara. 1978. Discourse knowledge. In Donald E. Walker, ed. *Understanding Spoken Language*. New York: North Holland, 229-346.

Hendrix, Gary. 1979. Encoding knowledge in partitioned networks. In Findler, Nicholas V., ed. *Associative Networks*. New York: Academic Press, 51-92.

Hirst, Graeme. 1981. *Anaphora in Natural Language Understanding: A Survey*. Berlin Springer.

Hobbs, Jerry, and Stanley Rosenschein. 1978. Making computational sense of Montague's intensional logic. *Artificial Intelligence* 9, 287-306.

Johnson-Laird, Philip. 1977. Procedural semantics. *Cognition* 5, 189-214.

Johnson-Laird, Philip. 1978. What's wrong with Grandma's guide to procedural semantics a reply to Jerry Fodor. *Cognition* 6, 249-261.

Johnson, Tim. 1985. *Natural Language Computing: the Commercial Implications*. London Ovum.

Katz, Jerrold J. and Jerry A. Fodor. 1963. The structure of a semantic theory. *Language* 39, 170-210. Reprinted in Jerry A. Fodor and Jerrold J. Katz, eds. *The Structure of Language*. Englewood Cliffs, New Jersey: Prentice Hall, 479-518.

Kay, Martin. 1973. The MIND system. In Randall Rustin, ed. *Natural Language Processing*. New York: Algorithmics Press.

Kay, Martin. 1979. Functional grammar. In Christina Chiarello et al.,, eds. *Proceedings of the Fifth Annual Meeting of the Berkeley Linguistics Society*, 142-158.

Kay, Martin. 1984. Algorithm schemata and data structures in syntactic processing. In *Proceedings of the Symposium on Text Processing*, Nobel Academy (to appear).

Kilbury, James. 1984. GPSG-based parsing and generation. To appear in Claus-Rainer Rollinger, ed. *Probleme des (Text-) Verstehens - Ansatze der Kunstlichen Intelligenz*. Tubingen: Max Niemeyer.

Marcus, Mitch. 1980. *A Theory of Syntactic Recognition for Natural Language*. Cambridge, Ma.: MIT Press.

Minsky, Marvin. 1975. A framework for representing knowledge. In Patrick Winston, ed. *The Psychology of Computer Vision*. New York, McGraw-Hill, 211-277.

Montague, Richard. 1974. *Formal Philosophy* (edited by Richmond Thomason). New Haven: Yale University Press.

Mosteller, Frederick, and David L. Wallace. 1963. Inference in an authorship problem. *Journal of the American Statistical Association* 58, 275-309.

Nishida, Toyoaki. 1983. *Studies on the Application of Formal Semantics to English-Japanese Machine Translation*. PhD thesis, Kyoto University.

Pereira, Fernando C.N. 1981. Extraposition grammars. *American Journal of Computational Linguistics* 7, 243-256.

Pereira, Fernando C.N. 1985. A new characterization of attachment preferences. In David R. Dowty, Lauri Karttunen, and Arnold M. Zwicky, eds. *Natural Language Parsing*. Cambridge: Cambridge University Press, 307-319.

Pereira, Fernando C.N., and Stuart M. Shieber. 1984. The semantics of grammar formalisms seen as computer languages. In *Proceedings of Coling84*, Menlo Park: Association for Computational Linguistics, 123-129.

Pereira, Fernando C.N., and David H. D. Warren. 1980. Definite clause grammars for language analysis - a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence* 13, 231-278.

Phillips, John D., and Henry S. Thompson. 1985. GPSGP — A parser for generalized phrase structure grammars. *Linguistics* 23, 245-261.

Power, Richard, and Christopher Longuet-Higgins. 1978. Learning to count: a computational model of language acquisition. *Proceedings of the Royal Society, Series B* 200, 391-417.

Pulman, Stephen. 1984. Limited domain systems for language teaching. *Proceedings of the 22nd Annual Meeting of the Association for Computational Linguistics*, 84-87.

Reyle, Uwe, and Werner Frey. 1983. A Prolog implementation of Lexical Functional Grammar. *Proceedings of the 8th International Joint Conference on Artificial Intelligence* 693-695.

Rosenschein, Stanley, and Stuart M. Shieber. 1982. Translating English into logical form. *Proceedings of the 20th Annual Meeting of the Association for Computational Linguistics* 1-8.

Schank, Roger. 1972. Conceptual dependency: a theory of natural language understanding *Cognitive Psychology* 3, 552-631.

Schank, Roger and Robert Abelson. 1977. *Scripts, Plans, Goals and Understanding* Hillsdale, New Jersey: Lawrence Erlbaum and Associates.

Schubert, Lenhart, and Jeffry Pelletier. 1982. From English to logic: Context-free computation of 'conventional' logical translation. *American Journal of Computational Linguistics* 8, 27-44.

Shieber, Stuart M. 1983. Sentence disambiguation by a shift-reduce parsing technique *Technical Note* 281, SRI International. Also in *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, 113-118. And in *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, 699-703.

Shieber, Stuart M. 1984. Direct parsing of ID/LP grammars. *Linguistics and Philosophy* 7, 135-154.

Shieber, Stuart M. 1986. *Course Notes on Unification Approaches to Grammar*. Stanford CSLI.

Slocum, Jonathan. 1985. Machine translation: its history, current status and future prospects. *Computational Linguistics* 11, 1-17.

Taki, Kazuo, Minoru Yokota, Akira Yamamoto, Hiroshi Nishikawa, Shunichi Uchida, Hiroshi Nakashima and Akitoshi Mitsuishi. 1984. Hardware design and implementation of the personal sequential inference machine (PSI). Tokyo: *ICOT Technical Report* TR-075.

Thompson, Henry S. 1982. Handling metarules in a parser for GPSG. *Edinburgh D.A.I. Research Paper* 175. Also in Michael Barlow, Daniel P. Flickinger and Ivan A. Sag eds. *Developments in Generalized Phrase Structure Grammar: Stanford Working Papers in Grammatical Theory, Volume 2.* Bloomington: Indiana University Linguistic Club. 26-37.

Thompson, Henry and Graeme Ritchie. 1984. Implementing natural language parsers. In Tim O'Shea and Marc Eisenstadt, eds. *Artificial Intelligence: Tools, Techniques, and Applications.* New York: Harper & Row, 245-300.

Thorne, James, P. Bratley, and Hamish Dewar. 1968. The syntactic analysis of English by machine. In Donald Michie, ed. *Machine Intelligence 3.* New York: Elsevier.

Tomita, Masaru. 1984. LR parsers for natural languages. In *Proceedings of Coling84.* Menlo Park: Association for Computational Linguistics, 354-357.

Touretsky, David F. 1984. *The Mathematics of Inheritance Systems.* PhD dissertation Caernegie-Mellon University.

Wilensky, Robert. 1983. *Planning and Understanding.* New York: Addison Wesley.

Winograd, Terry. 1972. *Understanding Natural Language.* New York: Academic Press.

Woods, William. 1970. Transition network grammars for natural language analysis *Communications of the ACM* 13, 591-606.

Woods, William. 1973. Progress in natural language understanding: an application to lunar geology. *AFIPS Conference Proceedings* 42, 441-450.

Woods, William. 1981. Procedural semantics as a theory of meaning. In Aravind K. Joshi, Bonnie L. Webber, and Ivan A. Sag, eds. *Elements of Discourse Understanding* Cambridge: Cambridge University Press. 300-334.

Yasukawa, Hideki. 1984. LFG system in Prolog. *Proceedings of the 22nd Annual Meeting of the Association for Computational Linguistics,* 358-361.

Younger, David H. 1967. Recognition and parsing of context-free languages in time n3 *Information and Control* 10, 189-208.