Parsing Natural Language

edited by
Margaret King

Parsing Natural Language

Edited by

MARGARET KING

Dalle Molle Institute for Semantic
and Cognitive Studies,
University of Geneva, Switzerland

1983

ACADEMIC PRESS

A Subsidiary of Harcourt Brace Jovanovich, Publishers

London New York

Paris San Diego San Francisco

São Paulo Sydney Tokyo Toronto

ACADEMIC PRESS INC. (LONDON) LTD.
24/28 Oval Road
London NW1

United States Edition published by
ACADEMIC PRESS INC.
111 Fifth Avenue
New York, New York 10003

British Library Cataloguing in Publication Data
Parsing natural language.
1. Language and languages
I. King, M.
425 P123

ISBN 0-12-408280-7

Printed in Great Britain

CONTRIBUTORS

Eugene Charniak *Department of Computer Science, Brown University, Box 1910, Providence, Rhode Island 02912, U.S.A.*

Anne De Roeck *Dalle Molle Institute for Semantic and Cognitive Studies, 54 route des Acacias, 1227 Geneva, Switzerland*

Roderick Johnson *Centre for Computational Linguistics, UMIST, P.O. Box 88, Sackville Street, Manchester M60 1QD, England*

Margaret King *Dalle Molle Institute for Semantic and Cognitive Studies, 54 route des Acacias, 1227 Geneva, Switzerland*

Steve Pulman *School of English and American Studies, University of East Anglia, University Plain, Norwich NR4 7TJ, England*

Graeme Ritchie *Department of Computer Science, Herriot-Watt University, 79 Grassmarket, Edinburgh EH1 2HJ, Scotland*

Michael Rosner *Dalle Molle Institute for Semantic and Cognitive Studies, 54 route des Acacias, 1227 Geneva, Switzerland*

Geoffrey Sampson *University of Lancaster, Department of Linguistics and Modern English Language, School of English, Lancaster LA1 4YT, England*

Steven Small *Department of Computer Science, University of Rochester, Rochester, NY 14627, U.S.A.*

Giovanni B. Varile *Commission for the European Communities, DG XIII, Jean Monnet Building, Plateau du Kirchberg, Luxembourg*

Yorick Wilks *University of Essex, Department of Language and Linguistics, Wivenhoe Park, Colchester CO4 3SQ, England*

ACKNOWLEDGEMENTS

The initial work on this book was done in preparation for a Tutorial on Parsing Natural Language, organised by the Dalle Molle Institute for Semantic and Cognitive Studies of the University of Geneva, the second in a series of tutorials held in Lugano under the general title of the Lugano Tutorials. The authors and the organisers would like to thank the Commune of Lugano for their continuing kindness in providing us with a very beautiful setting for an intensive week's work, and for all their help with the organisation of the tutorial. We should also like to thank Signor Angelo Dalle Molle, from whose original creation of a forum for interdisciplinary and cross-national discussion so much has sprung, and Martine Vermeire without whose constant support and help we should have been left with nothing more than random pieces of paper. Finally, the editor would like to add special thanks to Franco Boschetti, without whom nothing would ever have been possible, and to Anneke De Roeck, who took over far more than was reasonable of the editorial work.

PREFACE

This book is a product of the rather odd relationship between artificial intelligence and theoretical linguistics. The area of artificial intelligence represented here is that of work designed to produce computer programs capable of analysing natural language, usually for some practical purpose such as the construction of question answering or of machine translation systems. Theoretical linguistics takes the study of language as an end in itself, and is mainly interested in giving a coherent account of how language works. Clearly, each field ought to be able to make use of the other. The theoretical linguist may regard a computer program as a valid way of demonstrating that the theory is in fact consistent: the artificial intelligence worker may make use of the linguists' theoretical proposals as alternatives to his own in the construction of his programs. This book is the result of a week long tutorial intended to facilitate and encourage the dialogue between the two disciplines. One central theme, that of parsing, was chosen as the chief topic of the tutorial, on the grounds that parsing is currently proving of great interest to both parties. In order to facilitate discussion, introductory material was included to give those from one of the disciplines with little or no knowledge of the other some basic background. Recent work on parsing was then reported and examined, with the aim of describing the current state of the art and of stimulating new research.

The pattern of the book reflects these aims. A preliminary Section tries to establish a common starting point for those not expert in the area. The first chapter covers some basic terminology and defines concepts from the theory of formal languages which are used extensively by later chapters. The second chapter is essentially historical, covering early attempts at the computational application of the 1960s version

of transformational grammar. The importance of these attempts derives chiefly from the nature of the problems encountered there. Much of the work discussed in Sections 2 and 3 was directly or indirectly stimulated by a desire to overcome them. The remaining chapters of the introductory Section cover established techniques which are frequently used as a starting point for discussion of new ideas, and can therefore be considered essential background. In all of these chapters, the approach taken has been to give the basic outline and some critical evaluation, rather than to concentrate on detailed discussion.

The second Section concerns recent developments in the field of syntax and their application to parsing. The developments discussed fall within the field of transformational grammar and are described in those terms. Nonetheless, where the ideas have been used within a computational framework, the basic mechanisms of the parser involved are given. The current revival of interest in syntactic methods is clearly demonstrated in this Section.

The final Section, on parsing and semantics, consists mainly of a discussion on the role of semantics within parsing, and an evaluation of the different ways in which semantic information can be incorporated into a parser. Its position at the end of the book reflects the history of parsing over the last few years. An initial concentration on syntactic parsers led to the realisation that syntax alone was not enough. The reaction took the form of parsers aimed at minimising the use of explicit syntactic information, relying instead on semantics. This period in the history of parsing is by now well covered in the literature, except for recent attempts to develop lexicon based semantic parsers, as described in the final chapter.

The authors come from a variety of backgrounds within the gamut of linguistics and artificial intelligence. Although the same material is sometimes touched on by more than one author, the difference of viewpoint of the authors concerned is in itself illuminating.

A final remark should be made about content. For many workers, both in artificial intelligence and in linguistics, one of the most important aspects of their work is in its psychological implications. Thus, they use their work as the basis for the construction of psychological theories, and also regard evidence that their theories represent psychological reality as critical. This aspect of work on parsing has been for the most part deliberately avoided, on the grounds that to discuss it properly would demand a radically different approach and would justify a book by itself. The exceptions to this are where psychological considerations have had a direct effect on the design of the parser being discussed.

Margaret King January 1983

CONTENTS

SECTION I

INTRODUCTORY

Apart from the first chapter which presents some basic terminology which will be used throughout the rest of the book, this first Section is primarily concerned with setting the stage for the two later Sections and in picking out themes which will recur throughout the book.

It starts with a discussion of transformational parsing, where transformational refers to the theory of transformational grammar as it was presented between, say 1965 and 1972. It is not just historical curiosity which motivates this discussion. Many of the early computational parsers were based on, or derived from, the then current theories of transformational grammar. Work on these parsers proved seminal in many ways, not least in that they raised a whole series of problems which later work aimed at resolving. Thus, one of the reasons for the development of augmented transition networks (chapter 4), for example, was to find an efficient way of finding a correct path through a grammar, whilst preserving the power of a transformational grammar.

The proposal of charts (chapter 5) as a way of representing intermediate and final results during a parse can be seen as a proposal to solve the problem of non-determinism posed by a transformational parser. At a certain point during the parse, more than one continuation is possible, with no way of knowing which is the correct one. At this point, the parser may opt for one of the possibilities and then be prepared to come back on its tracks and try another if the choice leads to a dead end, or it may try to follow both paths simultaneously and risk finishing up carrying around a large number of competing possibilities. (If each of two possibilities has in its turn two possibilities for further development then there are four possibilities – very large numbers are very rapidly reached this way.) Chart structures offer a way of keeping all the alternatives alive whilst minimising the amount of structure being carried around.

The connection with production systems (chapter 3) is less direct in terms of historical development, but none the less strong. A transformational grammar is a special case of a production system, with all the associated problems of modularity and control. The production system model allows 'chunks' of knowledge to be expressed individually and independently, just as a transformational rule expresses items of linguistic knowledge. But when these individual rules are combined into a system of rules in order to do something with them, they necessarily interact. The problem then becomes how to ensure that they interact in the right way at the right time, which is precisely the central problem of production systems.

So far we have picked out the themes of control, of representation and of non-determinism. The first two of these themes are implicitly present throughout the rest of this book. The third will receive special attention in the next Section.

1

An Underview of Parsing

A. De Roeck

1. INTRODUCTION

A notion that should become clear at an early stage in a book like this is the one of *parsing* in the context of language applications. Clearly the answer to the question 'what is parsing?' should at least fulfill two goals. First, it should be simple enough to contribute to a general understanding of the issue. Second, it should be general enough to be applicable to all instances. At the same time over-simplification is to be avoided lest the information become irrelevant to any practical purpose.

The only way I see of respecting these criteria consists of taking recourse to what has already been established in the disciplines of mathematical linguistics and compiling theory. Talking about *grammars* seems a reasonable starting point.

2. GRAMMARS

2.1 Generative grammar

We all speak a language. Only physical limitations like tiredness, etc. can stop us from inventing sentences in that language. Each of those sentences may be different from all the ones thought of before and we can go on expressing new sentences quasi-forever. All this means

is that in a language there are infinitely numerous different sentences.

The task the linguist has set himself is to describe human languages, i.e. infinitely large sets of sentences, and to do so in a manner that enables him to make the difference between those utterances that are part of the language described and those that are not. There are essentially two ways he can go about his job. One consists of listing all the sentences in a particular language. Any possible fragment is then bound to be included in the enumeration and can be checked against it. Nevertheless this method has some severe disadvantages. First of all, having established that the number of sentences in a language is infinitely numerous it becomes easy to deduce that one would never be able to finish the list. Also, although putting a large number of sentences down on paper may in some sense be equivalent to describing them, the technique does not allow for the explicit expresion of what those sentences have in common – i.e. some generalities and characteristics of the language. For instance, listing 20,000 random English sentences does not explicitly specify that most of them have a verb. Clearly, this way of doing the job is hardly satisfactory.

There is, however, another possibility for describing all sentences that make up a language; one that takes less time and does allow for the expression of some generalities. It is based on the idea that humans, who are after all beings with a *limited* memory, can decide when hearing a sentence whether it belongs to a language they speak. This suggests that, per language, there exist a *finite* number of criteria which each sentence has to fulfill and of which humans have an *implicit* knowledge they can use when making linguistic judgements. Linguists set about trying to discover what those criteria may look like and formulate them *explicitly* in a *grammar.*

So, a *grammar* contains a limited number of *rules* that describe a language. If the rules only tell you what the sentences can look like the grammar is said to have *weak generative capacity*; if they also predict what structure will underly the sentences they describe, the grammar has *strong generative capacity.* The term 'generative' refers to the fact that the grammar *explicitly describes* or *predicts* 'all and only' the sentences in a language: a generative grammar is an *explicit definition* of a language, and the word 'generative' is in this context in no way equivalent to 'produce' or 'output'.

2.2 Formal Grammars

The point has come to translate all this into more formal terms. A language can be seen as an infinitely large set of sentences. Each sentence is characterised as a *wellformed string* over a finite vocabulary

of *symbols*. For the sake of simplicity, you can think of those symbols as words, though this view is not quite correct. *Wellformed* means that the form of the string – i.e. the way the symbols are put together – does not violate certain criteria specified in *rules of formation* which are contained in a *grammar*

At this level it no longer suffices to say that a grammar describes a language. The notion of *formal grammar* surfaces here, still serving the same purpose of describing a language, but in a form which is very rigidly defined.

A formal grammar G is a quadruple $\langle V_N, V_T, P, S \rangle$, where:

- V_T is a finite set of *terminal symbols*. If the grammar generates human language these symbols coincide more or less with the words of the language
- V_N is a finite set of *non-terminal* symbols. Sometimes they are also referred to as *variables* (Hopcroft and Ullman 1969: 10). In linguistic applications they correspond to categories. It is their presence in the rules that allows a grammar to express general wellformedness conditions.
- P is a finite set of rules called *productions*. They are of the form '$\alpha \rightarrow \beta$' (α rewrites as β) where both α and β stand for strings of elements of V_N and V_T.
- S is the *starting symbol* or *root*. S is an element of V_N and has to occur at least once on the left hand side of the rewrite arrow in the productions (in the place of α).

A grammar G generates a language $L(G)$. There exist several different types of grammar, depending on the form of the strings α and β in the rules (i.e. exactly what elements of V_T and V_N occur in α and β). The type of G determines the type of $L(G)$: grammars of a certain type generate languages of a corresponding type.

2.3 Context-free grammars

Maybe an example is called for to make all this a bit clearer. Of the different types of grammar I pick out one, namely the kind of grammar that is called *context-free* (hereafter *CFG*). The choice is motivated by practical considerations. CFGs are relatively transparent and their nature is well understood thanks to multiple applications in computer science. On top of that they are claimed to generate a subset, arguably all, of English and of some other languages, which should make them useful in the context of this volume.

In a CFG the productions have to be of the form '$A \rightarrow \alpha$' where 'A' is one symbol belonging to V_N (the set of non-terminal symbols) and 'α' is a string of terminals and/or non-terminals. Here is an

example of a CFG:

1. $V_N = \{S, NP, VP, N, ART, V\}$
   $V_T = \{cat, mouse, eats, the\}$
   $P = \{S \rightarrow NP\ VP$
   $NP \rightarrow Art\ N$
   $VP \rightarrow V\ NP$
   $VP \rightarrow V$
   $V \rightarrow eats$
   $N \rightarrow cat$
   $N \rightarrow mouse$
   $Art \rightarrow the\}$
   S

The symbols in V_N respectively stand for the categories 'Sentence', 'Noun Phrase', 'Verb Phrase', 'Noun', 'ARTicle' and 'Verb'. The symbols in V_T correspond to words occuring in everyday English. The form of the productions corresponds to the definition stipulated and 'S', the root or *axiom* is present and occurs, in this case only once, on the left hand side of a production.

The language this CFG generates consists of the following six sentences, which happen also to be wellformed strings of English:

2. The cat eats the mouse.
3. The mouse eats the cat.
4. The cat eats the cat.
5. The mouse eats the mouse.
6. The cat eats.
7. The mouse eats.

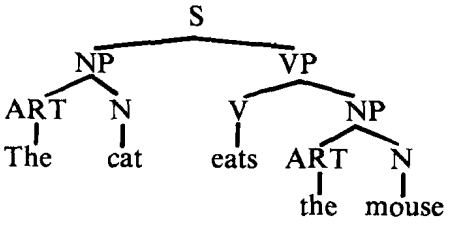
True enough, it was suggested earlier that the advantage, for linguists, of using a grammar for describing language lies in its capacity of describing *infinite* sets of sentences with finite means. Clearly the CFG in the example does not exploit that possibility. The reason for this lies in the absence of *recursion* in the rules – i.e. there is no rule or sequence of rules according to which a non-terminal symbol can be rewritten as itself. An example of such a rule would be 'S → S NP', where in one single rule 'S' can be rewritten as itself. Recursion can also be spread over several productions, e.g. as in the sequence 'VP → V NP', 'V → VP', where 'VP' rewrites as itself over two rules distance. Nevertheless, the lack of recursion in the CFG in 1 will be of no consequence for the utility of the example in this chapter.

Apart from generating the strings 2-7, the CFG in 1 also predicts their underlying structures. The structure it generates for sentence 2, for instance, is represented below under two different forms; first as a *bracketed string* as in 8

8. $(_S (_{NP} (_{Art}$ The $)$ $(_N$ cat $)$ $)$ $(_{VP} (_V$ eats $)$ $(_{NP} (_{Art}$ the $)$ $(_N$ mouse $)$ $)$ $)$ $)$

and as a *tree*, as in 9

9.



As a consequence it can be said that this grammar has strong generative capacity.

3. RECOGNISERS

In section 2 grammars have been defined as generating schemes, finite specifications for languages. There exists another way of specifying a language in a finite way, namely by means of *recognisers – abstract machines* which, when presented with a string of symbols will give a yes/no answer to the question '*Is this string a sentence of the language I* [*the recogniser*] *know about?* ' (For more details see also chapter 4 of this volume.)

A recogniser is often described as an *abstract* device which performs *operations* on an input string, according to a given finite set of *instructions*. Each of those instructions has to be *mechanically executable* using a *fixed* amount of time and energy. This comes down to saying that the set of instructions to a recogniser are a *procedure.*

An important point should be made here about the comparison between grammars and recognisers. A grammar and a recogniser are said to be *equivalent* if they respectively generate and recognise the same language, and for each type of grammar there exists a corresponding type of recogniser. But to say they are equivalent does not imply they have the same characteristics. A grammar offers a *static* description of a language, contained in a set of productions. Those productions though are not mechanically executable, and they are not ordered with respect to one another. They only give information about a language and give no clue about how that information gets used.

The instructions to a recogniser may *reflect* the same linguistic information as contained in a grammar but the machine will use it in a *dynamic* way, deciding which operation to perform next when confronted with some input at a given moment. This implies that the actual execution of the instructions is to some extent *ordered* as dictated by the input and by the instructions that were executed before. The set of rules to a recogniser is a procedure; the set of

productions in a grammar is not.

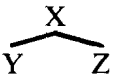
4. PARSING

Consider again the sentence in 2 and its underlying structure (in 8 and 9). Although the CFG in 1 generates both, it doesn't offer any indication on how the link between the sentence and the structure gets established. The sample grammar defines six sentences and six structures but in order to decide which structure underlies example 2 the grammar is not enough. What is needed is a *procedure* that will, this time, not just recognise the sentence but also discover how it is built. The execution of that procedure is called *parsing* and the thing that executes it is called a *parser.*

So, parsers do essentially two things. On the one hand, when presented with a string, they have to recognise it as a sentence of the language they can parse. In this respect, parsers have built-in recognisers. On the other hand they have to assign to that sentence a structure which they have to *output.* This implies that parsers must rely on linguistic information as contained in a grammar with at least strong generative capacity, whereas recognisers, because they do not output structure, can be built referring to grammars with weak generative capacity.

Parsers belong to the type of objects called *transducers* – i.e. in simple terms a recogniser augmented with output facilities. Just as for recognisers, it is important to see that grammars and parsers have a different nature, since a parser has a set of instructions which constitute a procedure (but which can, nevertheless, use the same linguistic information as expressed in a grammar). For the sake of both simplicity and perspicuity, though, I will assume for the rest of this chapter that the parsers described have access to exactly the linguistic knowledge as expressed in the CFG of 1, and that each instruction corresponds to what is expressed in a single grammar rule (see 13 below). In other words, a rule 'X → YZ' is no longer to be read as a production but as an abbreviation of a more complex instruction to a parser that will use it to output a fragment of structure:

10.



A parser usually proceeds by taking a string of symbols (the input sentence) and applying a rule to it, which mostly comes down to rewriting a bit of the string. For example, the string 'ABC' is rewritten into the string 'ADC' by applying the rule 'B → D' (rewrite 'B' as 'D') and 'ADC' into 'AdC' according to a rule 'D → d'. The strings