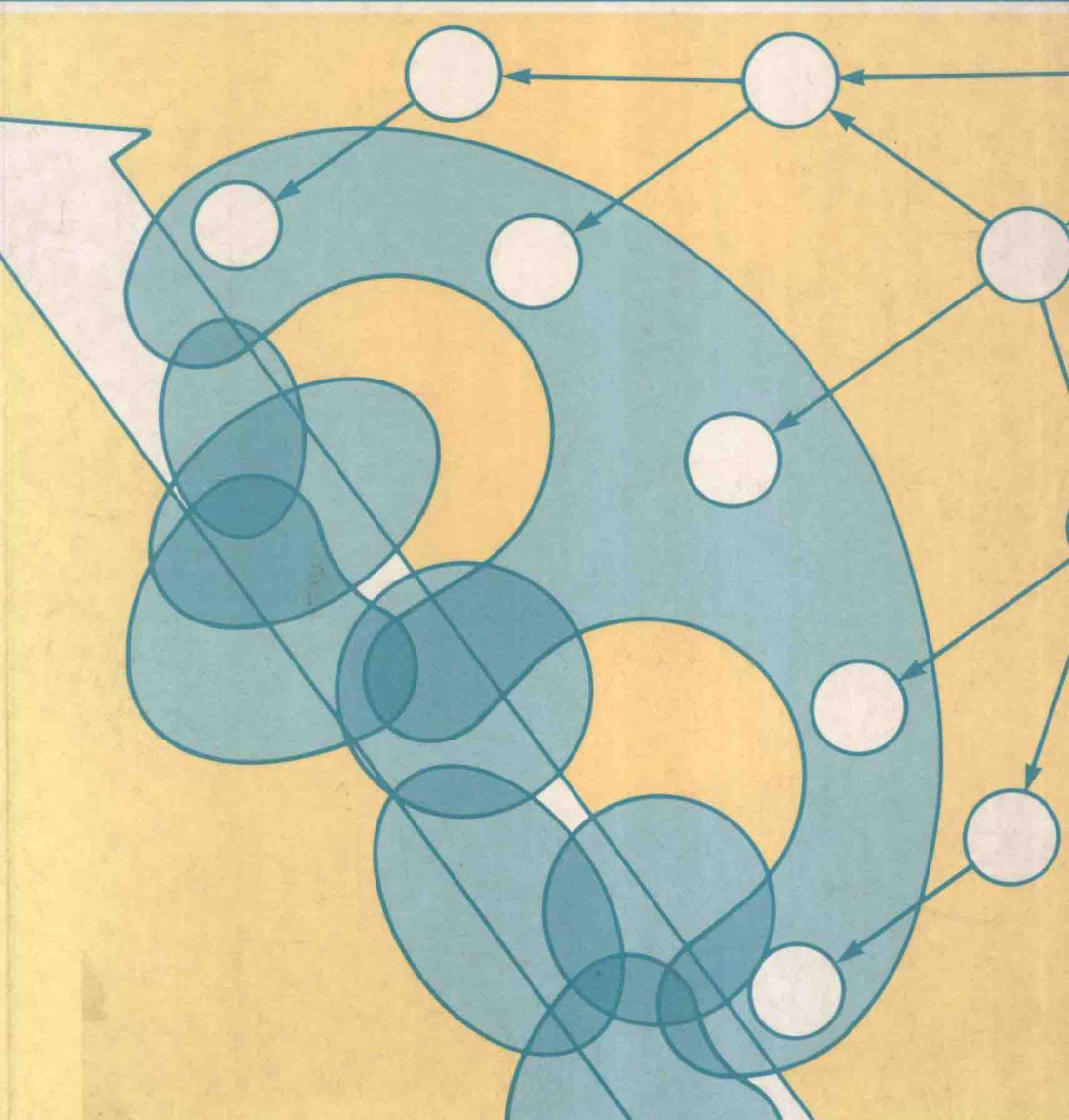
Natural Language Processing

Artificial Intelligence Texts

H.M.NOBLE

BLACKWELL SCIENTIFIC PUBLICATIONS



Artificial Intelligence Texts

NATURAL LANGUAGE PROCESSING

H. M. Noble

BLACKWELL SCIENTIFIC PUBLICATIONS
OXFORD LONDON EDINBURGH
BOSTON PALO ALTO MELBOURNE

Blackwell Scientific Publications
Editorial offices:
Osney Mead, Oxford OX2 0EL
(*Orders*: Tel. 0865 240201)
8 John Street, London WC1N 2ES
23 Ainslie Place, Edinburgh EH3 6AJ
Three Cambridge Center, Suite 208,
Cambridge, MA 02142, USA
667 Lytton Avenue, Palo Alto
California 94301, USA
107 Barry Street, Carlton
Victoria 3053, Australia

First published 1988

Set by DP Photosetting, Aylesbury, Bucks
Printed and bound in Great Britain by
Mackays of Chatham, Kent

DISTRIBUTORS
USA and Canada
Blackwell Scientific Publications Inc
PO Box 50009, Palo Alto
California 94303
(*Orders*: Tel. (415) 965-4081)

Australia
Blackwell Scientific Publications
(Australia) Pty Ltd
107 Barry Street,
Carlton, Victoria 3053,
(*Orders*: Tel. (03) 347 0300)

British Library
Cataloguing in Publication Data
Noble, Hugh
Natural language processing.—(Artificial intelligence texts).
1. Linguistics—Data processing 2. Man-machine systems
I. Title II. Series
006.3'5 P98

ISBN 0–632–01738–4
ISBN 0–632–01502–0 Pbk

Library of Congress
Cataloging-in-Publication Data
Noble, Hugh.
Natural language processing / Hugh Noble.
—(Artificial intelligence texts)
Includes index.
ISBN 0–632–01738–4 : £25.00
ISBN 0–632–01502–0 (pbk.) : £12.00
1. Linguistics—Data processing.
I. Title. II. Series.
P98.N57 1988 87–27887
410'.28'5—dc19 CIP

Introduction

The construction of a computer system which understands natural language is an important enterprise for two reasons:

(a) such a system would enable people to converse easily with a computer system and extract useful work from it without the restrictions imposed by the use of an artificial programming language, and

(b) the techniques involved would provide valuable insights into the mechanism of the human mind.

Both of these claims are contentious. A solution to the problem of natural language processing would not only make it easier for every user to converse with a computer system, but would make it easier for the unscrupulous person (or organisation or government) to invade the privacy of others by automating the task of mass surveillance, which at present is prohibitively labour intensive. It is appropriate, therefore, that we pursue our goal in the public domain so that the social implications can be debated openly and in an informed way. The proper way to prevent abuses of technology is through enlightened political policies and not through an inhibition of technological development.

Many people would object to the suggestion, implicit in our statement of reason (b) above, that a computer system which appears to understand natural language will of necessity do this in a way which is similar to the way humans understand natural language. There are, after all, numerous examples of 'clever' computer systems which carry out their task in a way which has very little similarity to the way humans do the same task. Automatic landing systems for aeroplanes make use of radar beams and the recognition by a robot of machine parts will often make use of sensors placed under the delivery tray to sense the weight of the objects. These use mechanisms which have little similarity to human skills.

We can, however, specify the criteria for the successful landing for an aeroplane in a way that does not involve human judgement. There is a clear goal in relation to which the success or otherwise of an automated landing system can be judged, and any system which achieves the specified goal can be described as successful no matter how it achieves it. We would argue, however, that the task of understanding natural language is somewhat different. Natural language is a product of the human mind as well as its servant. We cannot specify the successful processing of language in a way which does not involve what the

human mind does to that language, and so there can be no definition of 'meaning' or of 'understanding' which does not involve the subjective judgement of humans. When humans judge a natural language processing system as 'successful' we submit that what they are saying is 'The observable behaviour of the system is such that it is reasonable to assume that it is doing internally the same things that we do when we understand language. It has formed internally the same or nearly the same functional/logical structures which we form. It is able to make the same deductions and will note the same implications.' That seems to be the assumption which we make about a fellow human being when we say that another person 'understands' something we have said. We have no direct evidence that the minds of other people function as our own do, but alternative explanations for the behaviour of other people seem very far fetched indeed.

Therefore while it may be possible to build a computer system which processes natural language, and responds to it in some interesting way, what it does cannot be described as 'understanding' unless what it does is in some way logically equivalent to what humans do. We speak only of the functional or logical mechanism, however. The actual physical mechanism can be quite different. When we use a data structure to represent some element of meaning in a computer we do not wish to imply that humans will necessarily construct an actual data structure of the same kind, with pointers in the same places and so on. What we do think is that the human mind may have its logical equivalent. That is, it will associate elements of information in the same way using some storage technology of its own and use it for the same purpose.

But the goal of developing a fully functional natural language processing system is an extremely elusive one. Some progress has been made. One can purchase a natural language front-end to a database system, or conduct a helpful conversation with a computerised 'medical consultant'. Computer systems can scan press reports and produce standardised summaries, and crude systems are available which translate from one natural language into another. Full natural language processing which is comparable to human performance has not been attained, however, and is unlikely to be attained in the near future.

We should not be too disheartened or surprised by this. The only surprising thing is that anyone should be surprised by the failure. A typical child learns to speak and understand language gradually over many years, and has at his or her disposal (24 hours a day), the brain, a computer immensely more complex and powerful than any artificial computer built to date. No one knows how many millions of years the 'program' which executes this process took to be developed. The human mind also has at its disposal perceptual apparatus which provides it with 'experiences' in terms of which its representations can be couched. In a computer system we will need to construct these for ourselves.

In this book we shall try to explain the nature of the difficulties rather than provide a solution to them, but we also hope to provide limited solutions to

specific application-oriented problems. We shall begin with the simplest of systems and add complications gradually, so that we can see the limits for particular techniques and why it is necessary to abandon them for new and better techniques. The 'solutions' provided in the early parts of the book will be seen to be inadequate later in the book.

A few years ago most of the literature on natural language processing by computer was contained in technical journals, monographs and research papers which were not always accessible to people who were not members of staff in one of the academic centres for such research. Fortunately this position is changing and a wealth of new books on the topic are now appearing. They still tend, however, to be oriented towards research reports, and reprint these without much analysis or comparison. In this book we have tried to describe different approaches as a coherent progression, characterised by increasing complexity and richness of semantic description, and have not, for the most part, identified particular techniques with particular researchers or systems. To maintain the continuity we have occasionally ignored certain aspects of a particular published system. Since the book is intended for undergraduates the bibliography is confined to the most accessible sources.

The book is in three parts.

In Part 1 we begin with a very simple system, but one which has a very practical function – a user front-end to a microcomputer-based graphics facility. With this as our basic example we illustrate certain general principles – the need for a formally defined grammar, the need for a formal internal representation, the need for pattern matching, the notion of 'focus', the use of transition networks, and the need for the creation of 'side-effects' and for recursive mechanisms to deal with failure and backtracking. Finally we try to generalise this work by providing a general strategy for writing an ATN parser and a grammar of English.

In Part 2 we study various approaches which have been tried for dealing with the semantics rather than the syntax of language. We have organised these approaches in sequence of increasing semantic content. We begin with case grammar and end with Schank's Conceptual Dependencies and the use of scripts which, at least in some implementations, almost completely abandon any kind of formal grammatical analysis.

In Part 3 we discuss some of the philosophical issues which we avoided earlier in the book. To prevent this discussion becoming entirely esoteric and unrelated to practical matters, we have also provided a formalism for the representation of meaning. It is incomplete and cannot be considered a solution to the NLP problem, but it allows us to discuss some of the issues in a concrete way within a common representational framework. Our concern here is to provide the reader with an exposition of the problems to be faced and to help him or her avoid the overconfident naivety which has afflicted research workers in this field in the past.

Research in this area is usually justified, in public, in terms of practical applications, and these are real enough. But the true motivation is that this is a fascinating problem, the solution for which seems tantalisingly near and yet just beyond our grasp.

Note

POP11 was chosen as the main programming language for illustrating various algorithms because its syntax is similar to that of Pascal and C, which is not true of either LISP or Prolog, the other two languages which have all of the features required for this type of programming.

A brief introduction to POP11 is provided in an appendix.

Contents

Part 1

CHAPTER 1

An Initial Problem

1.1 The Micro-graphics 'World'

A small personal microcomputer is equipped with a simple graphics facility and an appropriately simple graphics language. We wish to construct a user-friendly interface which will accept a user's commands in natural English and translate these commands into the equivalent statements in the graphics language. The natural language allowed will be restricted, so that the interface will not be too difficult to construct, but it will be sufficient to illustrate a number of points and lead us towards more complex examples.

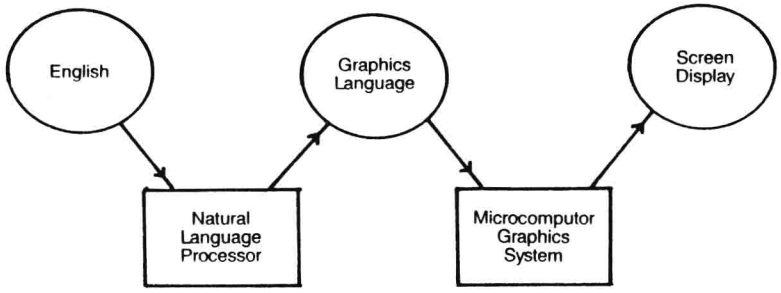


Fig. 1.1. The outline structure of the micro-graphics system.

The problem is therefore characterised as follows. The input text is a very limited subset of English, constrained to conform to a strict and limited grammar and the 'meaning' of the input text statements is represented in the form of statements in an internal language (the internal text).The internal text is the graphics language supplied with the machine. There is therefore no complication about what we mean by the 'meaning' of an input statement. If a statement cannot be translated into a statement in the graphics language then it must be considered to have no meaning at all. That is why we refer to the system as a 'world'. It has a definite boundary beyond which it cannot see and beyond which (so far as the system is concerned) nothing else exists. The problem is illustrative of a number of interesting and successful natural language processing systems which have been constructed for limited worlds such as this.

The problem for our system is to convert English into graphics language. But recognising that the text which is input cannot be unrestricted English, and that

the output need not only be graphics commands, we can generalise the requirement to that of achieving the conversion: *input text –> internal text*

1.2 The Internal Text

Points on the screen (or pixels) are defined by X,Y coordinates (1000 × 1000). The user may provide names for specific points thus:

point fred 123 456

which attaches the user-defined name *fred* to the point 123, 456. Such a statement is always preceded by the keyword *point.*

Lines may be defined by the user thus:

line fred tom

which defines a straight line drawn from the point with name *fred* to the point with name *tom. Fred* and *tom* must be predefined. Alternatively the user may refer directly to coordinates thus:

line 123 456 678 901

which defines a line drawn from the point 123, 456 to the point 678, 901. A typical program in this limited graphics language would therefore consist of a series of point definitions followed by a series of line definitions. For example:

point p 123 456
point q 678 901
point r 543 654
line p q
line q r
line r p

which would draw a triangle. This or any similar program is an example of the *internal text.*

1.3 The Input Text

Suppose now that a user typed the following:

'Let p be the point 234 456 and q be the point 678 901. Draw a line from p to q and then draw a line from q to r which is the point 543 654. Join p to r.'

Clearly there are very many ways in which a user could express the same meaning. For the purposes of this exercise we wish to limit the variety of forms which the input text could take, and we deal with this by defining:

(a) an allowed vocabulary of lexical items (or words) and

(b) a grammer, which is a set of rules indicating the allowed patterns in which the words may appear.

The reader should beware of using the word 'word' carelessly in the presence of linguists. We need to distinguish between the 'lexical item' (or string of characters) and the 'sense' of a word (or its meaning). Even the term 'meaning' is fraught with difficulties and misunderstandings because linguists like to distinguish between 'intentions' and 'implications' and 'sense'. Consider for example:

'John means well'
'You did not bring a pen so that means that I will have to lend you mine.'
'"To cry" means "to produce tears".'

We do not intend to be pedantic about this and when we use the term 'word' or the term 'meaning' it will usually be obvious from the context which interpretation is required. If not, we shall qualify the term.

It will be seen that the *internal text* consists of two kinds of statement: *definitions*, or *assignment statements*, which associate names or labels with actual coordinate points; and *action statements* which cause some graphic images to appear on the screen, usually drawing lines between the points which have been defined. We shall consider these two kinds of statement separately.

1.4 Assignment Statement Analysis

Consider first the statement which the user might type in order to define point p. We shall call this an 'assignment statement' because it assigns coordinates to a label (p).The assignment statement might take the form:

let p be x y
or *let x y be p*
or *let the point p have coordinates x y*

where p is a user-defined name or label for a point and x and y are numeric values. To cover such possibilities we classify words and phrases so that our description of the allowed statement structure can be more general. Let us use the classification *determiner* to stand for the set of words *a, an, the*. Therefore when we want to save space we simply write <*determiner*> and the reader should understand that the expression (including the angle brackets) should be replaced by any of the words in the set. In any such definition there is a danger of ambiguity between the use of words (such as 'a', 'an' and 'the') within the definition and the use of words (such as 'means', 'or', 'the set') as a part of the definition (the 'meta-words'). To avoid such confusion we shall adopt Backus-Naur Form notation (BNF), so that the definition of the classification 'determiner' becomes:

determiner := *a/an/the*

The symbol := is taken to mean 'is defined as' and the slash '/' means 'or'. Any word which appears as itself on the right-hand side of such an expression is taken to be the literal version of itself. On the left-hand side of each expression there should be only one term, the entity being defined. In our grammar defined below the expression:

assignment := let<pt_defn> / <pt_defn>

means that the expression 'assignment' is defined as consisting of either the word 'let' followed by a 'pt_defn' expression, or it is just a 'pt_defn' on its own.

A 'pt_defn' (or 'point definition') expression is defined thus:

pt_defn := <ptname><verb><coorddefn> /
<coorddefn><verb><ptname>

That is, it has two possible patterns:

ptname	*verb*	*coorddefn*
p	*(let) be*	*123, 456*
point p	*(let) be*	*the point 123, 456*
the point p	*(let) have*	*coordinates 123, 456*
p	*is*	*123, 456*
point p	*has*	*coordinates 123, 456*

coorddefn	*verb*	*ptname*
123, 456	*(let) be*	*p*
123, 456	*(let) be called*	*p*
point 123, 456	*(let) be*	*p*
the point 123, 456	*is called*	*p*

The verb is in some cases prefixed by (let). This indicates that the verb which follows (the infinitive form) is acceptable only if the whole statement is prefixed by the word 'let'. In the other cases the verb is acceptable only if the word 'let' does not appear. It is still necessary to define ptname, verb and coorddefn. This definition will fail if the user employs names such as 'a' as the name of a point because 'a' is also a word in the English language. We shall return to this point later. In the meantime we shall assume that the user avoids these ambiguities.

Without further ado we can write down the remainder of the complete grammar of the assignment expression in Backus-Naur form:

(a) *assignment:= let <point-defn> / <point-defn>*
(b) *point-defn:= <pt-name> <verb> <coord-defn> / <coord-defn>*
<verb><pt-name>
(c) *verb:= is / has / is called / (let) <infinitive>*

(d) *infinitive:= be / have / be called*
(e) coord-defn:= <coord-pair> / <classifier><coord-pair>
/ <determiner><classifier><coord-pair>
(f) *coord-pair:= number number / number , number*
(g) *classifier:= coordinates / line / point*
(h) *determiner:= the / a*
(i) *pt-name:= <user-defined-name> / <classifier><user-defined-name>*
/ <determiner><classifier><user-defined-name>
(j) *user-defined-name:= any lexical item with an undefined value.*

It is not necessary in BNF to number or label the clauses. We have done so here (a,b,c,...j) to make it easier to refer to individual clauses.

1.5 The Function 'ptname'

We now have a grammar for our input text assignment statement and the next requirement is to analyse this in terms which are easily converted into a program. A good strategy is to subdivide the problem into smaller problems, and so we shall consider first the *ptname* and develop an analysis and a program for that. (We shall assume the existence of a function *isname(w)* which takes as its argument a single 'word' (in POP11 terminology, or a textual *atom* in a list) and yields *true* or *false* depending upon whether or not it is a user-defined name.

We assume that the input text is presented to this system in the form of a list of atoms (words and numbers) e.g.:

[the point fred has coordinates 123, 456]

The program will scan the input text from left to right and test for the existence of a ptname at the start of the list. If a ptname is found the program (a POP11 function) must return three results:
(1) *true* or *false*
(2) The internal text sub-list, e.g. *[point fred]* (i.e. a sub-part of the graphics language statement) (or *nil* if *false*)
(3) The remainder of the input text.

The performance of the required function is as shown below.

input-text = [the point fred has coordinates 123, 456]
result = *true,*
[point fred],
[has coordinates 123, 456]
input-text = [to be or not to be]
result = *false*
nil
[to be or not to be]

In this way the required function can test the input text. If it is successful it will extract the part of the input text it has processed, prepare a sub-part of the final internal text that is appropriate, and hand on the remainder of the input text for further processing. If it is unsuccessful in finding a ptname phrase it will report failure, deliver up a NIL list instead of the sub-part of the output text, and hand back the complete input text for an alternative form of processing.

If the function is successful and the sub-list *[point fred]* has been formed, the processing of the remainder of the input text may construct the second half of the internal text list *[123 456]*. This will be concatenated with the first part giving the correct internal text form: *[point fred 123 456]*

Now consider the finite state diagram (FSD) shown below.

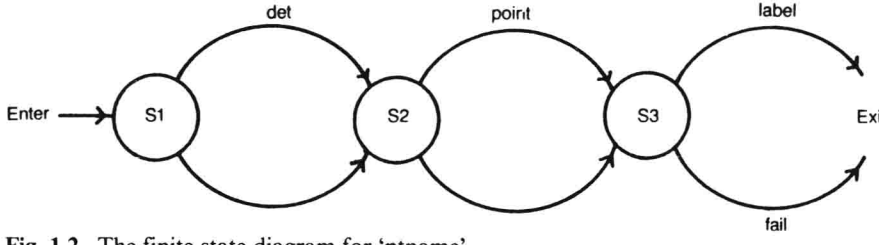


Fig. 1.2. The finite state diagram for 'ptname'.

This illustrates a system with three states (S1, S2 and S3). Each represents a 'state of expectancy' of the system. The system is, of course, our required function, which is processing the input text for a ptname phrase. Initially it is 'expecting' the determiner *the*. If it gets it it switches to state S2 and expects the classifier *point*. If this is found it switches to state S3, in which it expects a user-defined name. If this is found it exits, leaving the results indicated above.

If, on the other hand, the input text omits the determiner *the*, or both the determiner and the classifier *point*, then the system still switches to states S2 and S3 but takes different action as it does so. If the expected item is encountered, the system removes that item from the input text and goes on to test the next item in the list. If the expected item is not encountered, the input text is left unchanged and the system goes on to the next state to make a different test on the same first item as before.

Switching to a new state without making any alteration to the input text or taking any other action is shown in the FSD by the un-named arcs. These are often called 'jumps', and the introduction of jumps augments the FSD to become a new type of system.

It is very easy to turn a finite state diagram of this kind into a short program or function.

```
define isname(w);
ismemb(w,[a b c p q r tom dick harry]);
enddefine;
```