A Kernel for a General Natural Language Interface

Miguel Filgueiras

Nucleo de Inteligencia Artificial

Departamento de Informatica
Universidade Nova de Lisboa
Quinta da Torre
2825 Monte da Caparica
PORTUGAL

Abstract

A description is given of the main ideas used in the design of SPIRAL, a kernel for a natural language interface aimed at generality in linguistic ability and domain portability.

Though Prolog is used to implement the interface, syntactic analysis is not performed via metamorphosis, definite-clause, or extraposition grammar formalisms, but rather by means of a 3-level bottom-up extensible parser making use of rewrite rules. The application of each of these rules is controled by a module capable of embodying non-syntactic knowledge.

Syntactic and semantic analyses are separately done, but semantic tests are embedded in the parser resulting in a substantial decrease of ambiguity. The application dependent parts of the semantic analyser constitute a separate module. To make it easy to adapt the interface to new applications, a set of predicates is provided to help in the definition of that module.

Introduction

> "... reality may avoid the obligation to be interesting, but (...) hypothesis may not."
>
> Death and the Compass, J. L. Borges

Results from research on natural language understanding systems made during the last 15 years, either implicitly (by failing to meet certain requirements), or explicitly, point out the need for world knowledge, inference, context analysis and the like when trying to analyse a natural language sentence (this need is more acute when phenomena such as anaphora (reference problem) is dealt with [G. Hirst 81]).

One of the main problems with the logic grammar formalisms proposed so far (metamorphosis grammars [A. Colmerauer 75,78], definite-clause grammars [F. Pereira, D. Warren 80], and extraposition grammars [F. Pereira 81]), as well as with their concrete applications (from [R. Pasero 73] to [F. Pereira 83]), is that no provision is made to check each syntactic analysis step for consistency with respect to meaning. In this sense, syntactic analysis is carried out blindly. Introduction of tests in the grammar rules, typification [V. Dahl 77] and slot-filler based approaches [M. McCord 80,81], [F. Pereira 83], are incipient steps toward the use of non-syntactic knowledge to guide parsing. But in present day systems, whenever such knowledge is used it must be interspersed within the grammar rules and there is no neat separation at this level between the syntactic and the non-syntactic modules - even if semantic analysis is performed after syntactic analysis.

In what follows I present the main ideas underlying **SPIRAL**, an open kernel for a general natural language interface that gives an answer to the above criticism and simultaneously tries to keep a high degree of portability between applications.

In **SPIRAL** a 3-level parser is used to perform syntactic analysis. The second and third levels are executed in an interleaved fashion so that the third level checks second level results on the fly. The non-syntactic knowledge the third level has can easily be extended to include criteria based on knowledge from discourse context, world knowledge,

inference, and so forth. This way it is possible to have a desirable interaction between two highly modular devices, one working on the syntactic features and being controlled by the other which uses more complex forms of knowledge.

Syntactic and semantic analyses are separately done, but semantic tests are embedded in the parser resulting in a substantial decrease of ambiguity. The application dependent parts of the semantic analyser constitute a separate module. To make it easy to adapt the interface to new applications, a set of predicates is provided to help in the definition of that module.

Syntactic Analysis

"I state ; you, if you wish, refute."

The Aristos J. Fowles

A first point of divergence from the metamorphosis, definite-clause and extraposition grammar formalisms (referred to as '**logic grammars**' in what follows) is the parsing strategy. The **SPIRAL** parser makes use of a bottom-up technique better suited to accept external guidance and to analyse elliptic sentences and all forms of extraposition (I have no intention of entering the old and tired top-down versus bottom-up controversy - please cf. the quotation above - ; though many people tend to admit that the former is more efficient than the latter, this is **false** at least for (unbiased) context-free grammars [M. Kay 80]). The stress put on the two linguistic phenomena above (ellipsis and extraposition) follows from the purpose of not restricting 'ab ovo' the interface capabilities, and also from the relatively high frequency of such forms in Portuguese, the language actually analysed by **SPIRAL**.

While rules of a logic grammar constitute an indivisible program working on normally 3 types of data (surface representations, non-terminals and syntactic structures), **SPIRAL** is stratified into levels according to the functions performed and the kinds of data dealt with.

Rewrite rules in **SPIRAL** are in some extent similar to the rules in logic grammars. Obviously they occur in inverted forms, in accordance with the bottom-up parsing strategy - while in a logic grammar we have, for instance,

a -> b1, b2 ..., bn

in SPIRAL the same rule will appear as

b1, b2 ..., bn -> a

There is no distinction between terminals and non-terminals. A sentence is represented by the list of the lexical representations for its words, and the lexical representations can have Prolog variables to hold information for future use. Lists of lexical representations are what actually appears on both sides of the rewrite rules. Hence there are no restrictions on a rule's right-hand side, in contradistinction with logic grammars' left-hand sides.

Besides the lexical one, two other representation forms are used : one for what I call meaning-cells (**m-cell**, for short), and another for phrase structures built from them.

A m-cell tries to represent any contiguous words group that is meaningful on its own when isolated from the rest of the sentence. m-cells may contain other m-cells and be conjoined to give a m-cell. Some of the m-cell types **SPIRAL** currently works with are :

- noun phrase
- verb
- complement (an adjective group, a prepositional phrase, or an adverb)
- subphrase (relative clause)
- wh-question

For instance, in the sentence

'The system uses techniques to encode a more general model that are very efficient'

there are the m-cells

- the system
- uses
- techniques
- to encode a more general model
- that are very efficient

the last two of them containing

- encode
- a more general model
- are
- very efficient

Phrase structures are represented by a 3-place functor whose three arguments in a given instant describe of a phrase

- its main m-cells (either a verb, or noun-phrases - verbs are envisaged as phrase 'functors'),

- complements that await attachement to nouns or verbs (this simplifies the treatment of extraposition),

- subphrases found so far.

We can now examine how the **SPIRAL** parser works. On a first level of processing words are conflated whenever possible ; information from deleted words instantiate variables that occur on the lexical representations of the remaining words. This is a deterministic pass and results from applying rewrite rules like the following (in Edinburgh syntax, with '->' as infix operator),

```
[ determiner(Quant,Agr), noun(N,Quant,Agr) | R ]

        ->   [ noun(N,Quant,Agr) | R ]  .
```

This rule states that a determiner followed by a noun is deleted if both have the same agreement. Moreover, the quantification expressed by the determiner is saved in the lexical noun representation. This particular rule is a Prolog unit clause but some other rules have a clause body to test their applicability. After the first level, a second level analyses word groups to obtain m-cells. This is done by applying recursive rewrite rules with the following format :

```
List_of_lex_reprs_1  -->

            List_of_lex_reprs_2  -  M_cell   :- ...
```

where '-->' and '-' are infix operators. Such a rule means that **M_cell** is the result of analysing the first list of lexical representations, the second one being what is remnant. Like for the first level rules, clause bodies may impose conditions on rule application.

Each m-cell extracted by the second level is embedded into the current phrase structure by a third level of processing to produce a new phrase structure. Each clause head in the third level has the format

```
M_cell  +  Phrase_str_1   --->   Phrase_str_2
```

where '+' and '--->' are infix operators - its meaning is obvious. A monitor is used to control the second and third

levels forcing their interleaved execution. This monitor is defined by the following two clauses

```
mon( L0, P0, Ln, Pn )  :-   L0  -->  L1 - M  ,
                            M  +  P0  --->  P1 ,
                            mon( L1, P1, Ln, Pn ).

mon( L, P, L, P ).
```

So, whenever the third level fails by finding out that a m-cell is extraneous to the current phrase structure, backtracking to the second level takes place. In this situation, either an alternative analysis exists, or the monitor stops producing the phrase structure built so far and the rest of the sentence that remains to be analysed. A second level clause body may include a call to the monitor forcing a recursive analysis to be performed.

The first and second levels are purely syntactic, though the latter uses semantic tests to ensure correct attachment of complements to nouns. Both work by applying rewrite rules from two distinct sets comprising, respectively, about 10 and 25 rules. The third level must decide on whether a m-cell can or cannot be added to the current phrase structure. This important function, that imposes a check on each syntactic analysis step, is based, for the time being, on criteria concerning the phrase structure and some knowledge about complements and verb arguments (nouns are typed and for verbs a slot-filler approach is used, as in [V. Dahl 77], [M. McCord 80,81]). Those criteria can easily be extended to more sophisticated ones based on knowledge from discourse context, world knowledge, inference, and so forth.

This way it is possible to have a desirable interaction between two highly modular devices, one working on the syntactic features and dealing with lexical representations, the other using more complex forms of knowledge to build phrase structures.

In summary, the characteristics of these 3 levels in SPIRAL are as follows :

1st level - has some 10 rewrite rules transforming a list of lexical entries into another such list.

2nd level - has some 25 recursive rewrite rules that from a list of lexical entries produce one m-cell and remaining list ; each m-cell is passed to the 3rd level (as soon as produced) and if not accepted, alternative rules (if any) are applied ; otherwise, the processor stops giving as result the phrase structure built so far (if any), and the remnant list.

3rd level - builds the phrase structure from the m-cells extracted by the 2nd level controlling it by accepting or rejecting m-cells ; the 3rd level is also responsible for the treatment of extraposition, passivization, and composite nouns (like '**the dog, the cat and the mouse**').

When the end of the sentence is reached, another **SPIRAL** module is launched to check the phrase structure built for the sentence and to carry on with ellipsis analysis if needed. At present only a few incipient anaphoric forms are analysed by **SPIRAL** and by methods not completely adequate. Personal and possessive pronouns are solved by searching a noun list (built during the lexical analysis) and selecting a noun from it ; some kinds of ellipsis are solved by the introduction and dereferencing of a pronoun, and some others are treated by comparing phrase structures. The general philosophy prescribed in [G. Hirst 81] will sooner or later be adopted in **SPIRAL**. Nevertheless, the semantic tests used in the second and third levels, together with the slot-filler approach, provide a lot of information extremely useful in solving ambiguities. This fact allows for present methods to work well in many instances. A similar situation is encountered in the case-grammar approach, qualified in [G. Hirst 81] as "... **a firm base for anaphora resolution**", though only information from cases is used.

To help fix ideas, two (**simplified**) examples of sentence analysis follow - the two sentences are from [F. Pereira 81,83]. The functor ps(_,_,_) is used for phrase structures (see above for a description of its arguments), and * and ___/ are used to mark, respectively, a failure at the third level, and the words activating a second level rewrite rule. Important information bound to variables on the lexical representations of nouns or verbs is shown informally following them and within parentheses (e.g., **mouse(the)** represents the noun '**mouse**' containing the information from the determiner '**the**') ; in verbs, subject always precedes direct object. Numbers within braces denote comments to be found after each figure.

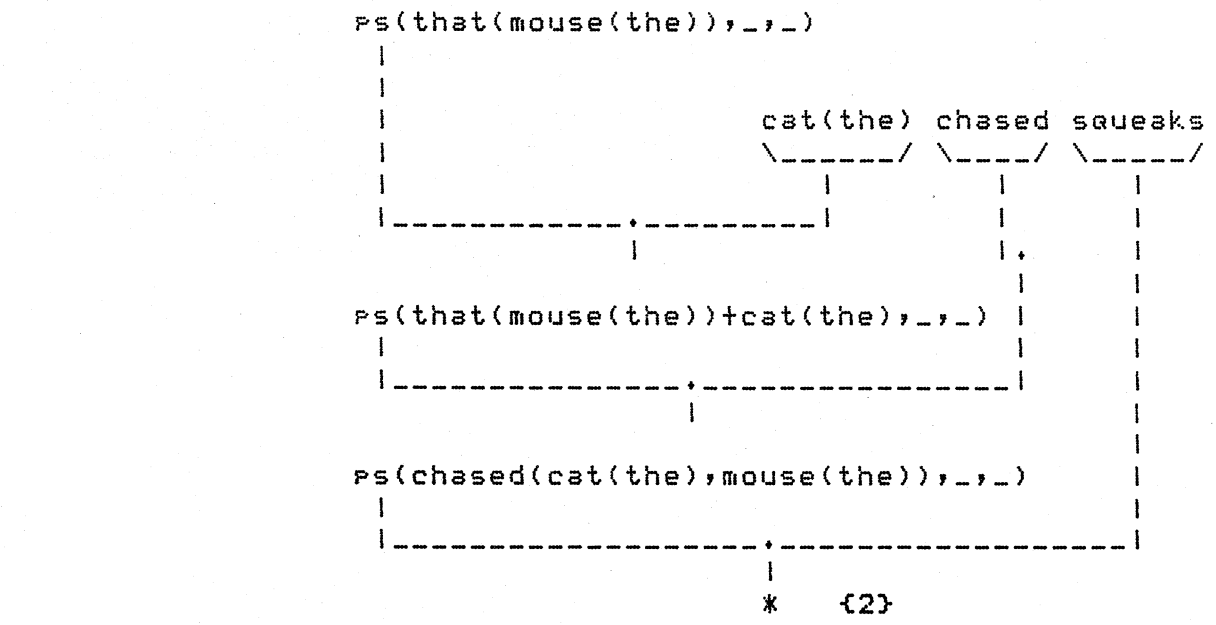
The second example shows that a sentence violating the Ross complex-NP constraint will not be accepted by **SPIRAL** - for ease of exposition the determiners are dropped.

Input sentence : the mouse that the cat chased squeaks

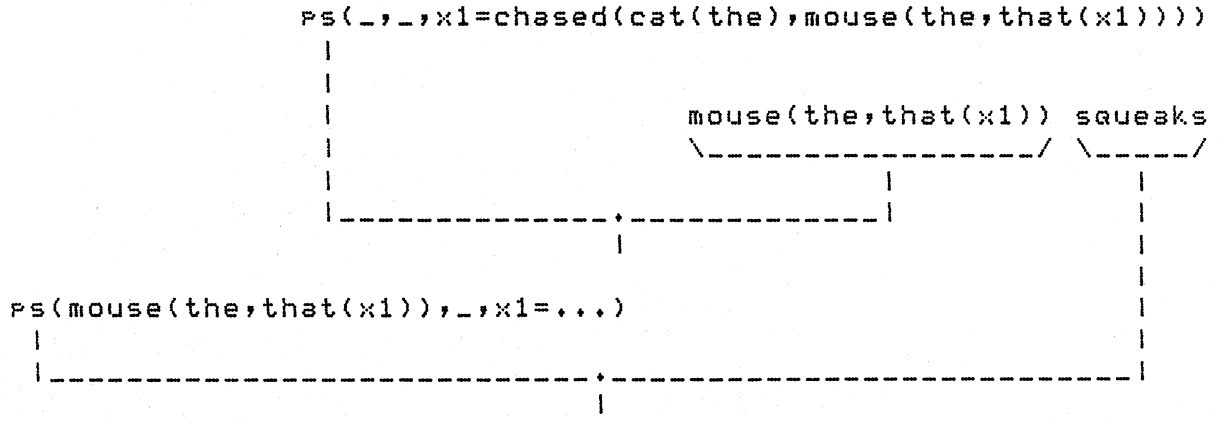
After 1st level : mouse(the) that cat(the) chased squeaks

2nd, 3rd levels : _____/ {1}

recursive analysis on :



end of recursive analysis {3}



ps(squeaks(mouse(the,that(x1))),_,
x1=chased(cat(the),mouse(the,that(x1))))

Result : squeaks(mouse(the,that(x1))) &
x1=chased(cat(the),mouse(the,that(x1)))

{1} - the relative will be analysed throush a recursive analysis.

{2} - 'squeaks' cannot be added to 'chased(the cat,the mouse)' because a phrase cannot have two main verbs, resulting in a failure at the 3rd level and the end of the recursive analysis.

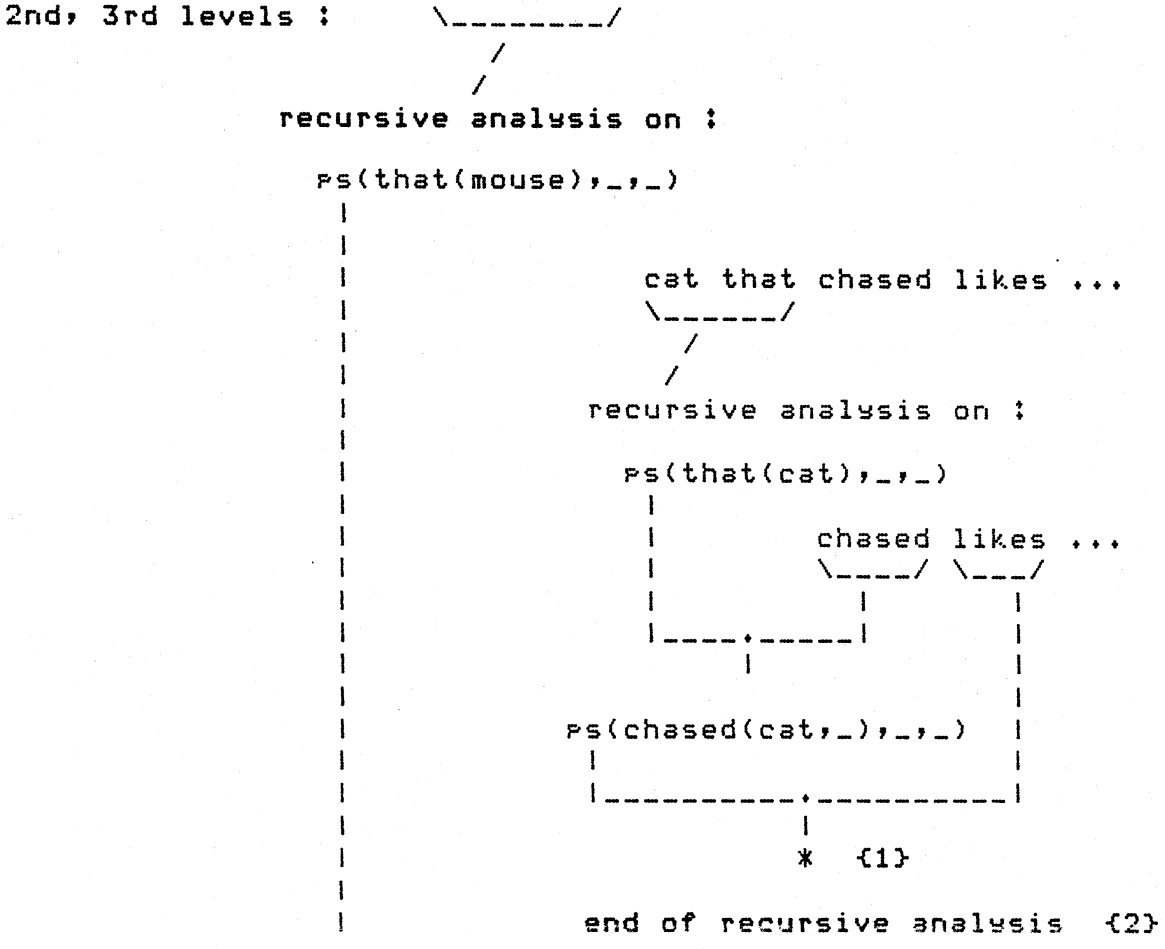
{3} - from the recursive analysis results a phrase structure that is passed as a subphrase to the 3rd level by the rule launching the recursive analysis ; this rule is also responsible for the binding of 'that(x1)' to the noun 'mouse' and for the anteposition of this noun to the remnant sentence.

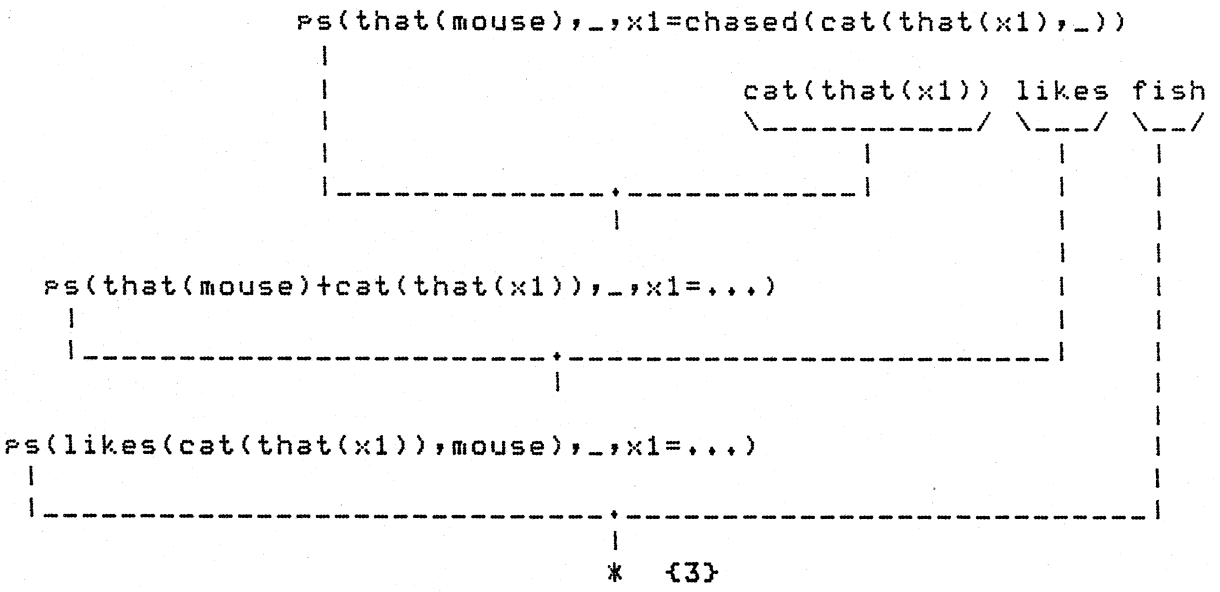
Input sentence :

the mouse that the cat that chased likes fish squeaks

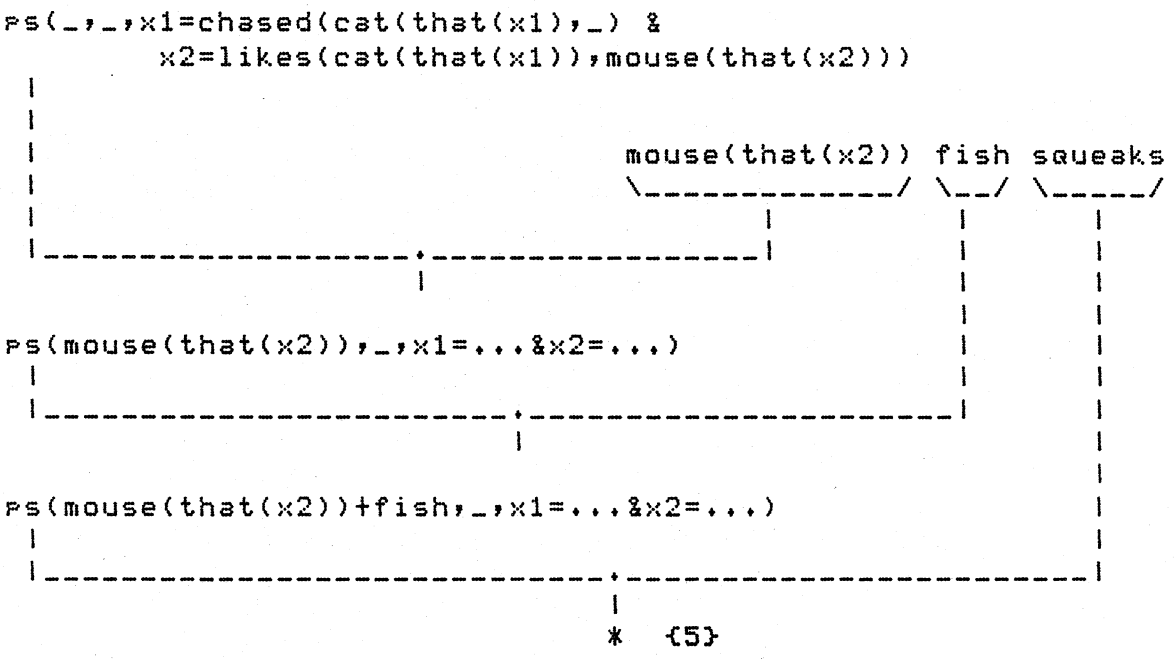
After 1st level : mouse that cat that chased likes fish squeaks

2nd, 3rd levels :





end of recursive analysis {4}



{1} - a phrase cannot have two main verbs, then 'likes' can not be added to 'chased(cat, something)'.

{2} - the subphrase just found is added to the phrase structure that already existed. Note that 'chased' is treated as transitive though with a direct object not stated - a common situation with certain verbs.

{3} - 'fish' cannot be added to 'likes(cat,mouse)' by the

reason in (1) above.

(4) - the two subphrases found so far are conjoined.

(5) - the analysis fails as 'squeaks' is intransitive.

If 'fishes' occurred instead of 'fish' and if a mouse could in any way fish squeaks (and in poetry - at least - this is obviously possible), the following analysis would be arrived at :

```
fishes(the mouse (that (
                    likes(the cat(that chased something),
                          the mouse)),
            squeaks)
```

To illustrate other capabilities of the syntactic analyser in SPIRAL some sentences that it accepts are listed below, the last of which because a direct translation from Portuguese is not correct in English - words within parentheses do not appear in the Portuguese version.

the author wrote a book in 1910.

in 1875 the author decided to write a book.

the works that the author wrote are for the piano.

the author that wrote in Venice a book.

the work that in 1920 was written by the author.

the author that was born in London and whose work was written in Paris.

the author whose work was written in the 18th century.

the piano is the instrument for which the work was written.

the authors in whose centuries works have been written.

the work A is older than the work B.

who wrote books ?

who wrote the oldest book ?

which are the works that were written in the 20th century ?

which are the works from the 19th century ?

in which century was born the author ?

the author wrote all his works in London.

the author that was born in the place where (he) wrote his works.

Lexical Analysis

In order to use dictionaries similar in content to current dictionary books (and this should be a goal for any natural language interface) some kind of suffix analysis must be performed at the lexical recognition stage. This need is still more urgent when analysing languages like Portuguese or French that make systematic use of inflections and conjugations, for substantial savings in dictionary space can then be gleaned.

To this end, I built (together with Antonio Porto, and much in the vein of [P. Sabatier, J.F. Pique 82]) a lexical analyser using a set of inflection/conjugation rules along with a dictionary containing word roots, words that constitute exceptions to the given set of rules or that are not described by them, and words that have no suffixes. For each input word (represented by the list of its characters in reverse order) the analyser tries a direct dictionary entry and subsequently (either by a failure in this attempt, or by a failure at the syntactic or semantic levels) performs suffix analysis. The current set of rules for Portuguese (some 80 Prolog clauses) covers 4 verbal conjugations in the 1st and 3rd persons, singular and plural, 4 tenses and pronominal conjugation for all this, as well as almost all inflections according to gender and number - some 17 different forms of plural. Typically a clause specifying a verb root implicitly defines some 68 different forms for it !

The counterparts to the dictionary compactness attained by this method are :

- some problems of representation duplication if word surface representation is to be kept for future use

- the dilemma of either allowing strange words to be accepted as valid by the inflection/conjugation rules, or burdenning the lexical analyser with tests

- an unfelt loss of efficiency

Concerning the dilemma above, if one accepts that the user should be responsible for the use of, e.g., 'writed' instead of 'wrote', there should be no damage if the natural language interface understands it according to the general rules. This is all the more so if the natural language interface provides a paraphrase of what has been understood after analysing a sentence - a research direction that will be taken soon. Obviously, for those not sharing this point of view there remains the possibility of providing tests to filter erroneous words.

Lexical ambiguity is treated by backtracking from the syntactic analyser. Some experiments on co-routining the lexical and syntactic analysers were made with some success by Antonio Porto using his ideas on control [A. Porto 82], and will be pursued in due course.

Semantic Analysis

For sake of modularity and generality, the semantic analyser uses an intermediate semantic representation (ISR) form to build a Prolog goal expression from a syntactic structure. An ISR form consists of Prolog goals, object (in general, entity) descriptions and auxiliar pseudo-goals (used to pass information while building the ISR form). Object descriptions are used the same way as in [A. Walker, A. Porto 83] ; in SPIRAL they occur under the form of a 3-place functor

o(Type:Var, Quant, Cond)

containing the object type, the Prolog variable associated with it, its quantification, and a defining condition in ISR that may contain other object definitions.

For instance, to the sentence

'the works from the authors of each century'

corresponds the following ISR expression and Prolog goal

```
o(work:W, each,
        o(author:A, each,
                o(century:C, each, _) &
                author(A,D) & century(D,C) ) &
        work(W,A) )
```

```
set(work/author/century) : Swac <-
        all(Swa/C,
            gen_cent(C) &
                all(Sw/A,
                    author(A,D) & century(D,C) &
                        all(W, work(W,A), Sw),
                    Swa),
            Swac)
```

where 'all' is the predicate defined in [L. Moniz Pereira, A. Porto 81] and 'gen_cent' is a generator of suitable century values.

ISR expressions are built from the syntactic structure by some general predicates, plus a separate set of application dependent ones, that define the semantics for verb and its complements, verb and its arguments, and noun and its complements. Writing such predicates for a particular application is made easy by the use of pseudo-goals and some pre-defined predicates coping with them (adding a Prolog goal to a condition, substituting a pseudo-goal by a Prolog goal, choosing and inserting Prolog goals from a list, and so forth).

When translating an ISR expression to a Prolog one, scoping problems concerning **distributive quantifiers** (such as 'each') and **aggregations** (such as 'average') [F. Pereira 83] are dealt with.

Efficiency and Future Work

SPIRAL has been implemented using the RT-11 Prolog interpreter by Clocksin, Mellish, Byrd and Fisher [W. Clocksin, C. Mellish, R. Fisher 80] and adapted by A. Porto and I to run under an **RT-11 Extended Memory** environment on a **PDP-11/23** machine with floppy-disks. The program currently occupies some **15K** (16-bit) words (in terms of nicely presented Prolog text about **23** pages as follows : 5 for the lexical analyser (including a common dictionary), 9 for the syntactic analyser, 3 for the semantic one, and 6 for the current application dependent parts - the application dictionary included). The remaining **5K** left free are what is needed as workspace. Future extensions under these conditions may force the use of a two-job partition as in [L. Moniz Pereira, P. Sabatier, E. Oliveira 82] or [L.M. Pereira, A. Porto 82] - it is no novelty that a **PDP-11/23** is a somewhat restricted machine !

Response times, though no exact benchmarks have been made, are comparable to those described in [L.M. Pereira, P. Sabatier, E. Oliveira 82] or [L.M. Pereira, A. Porto 82] and vary from less than 1 second for most sentences, to 10 or more seconds for very complex ones - these times are better than those obtained by a Lisp program that attempts to understand noun compounds, running on a PDP 2060 (it takes some 5 seconds to analyse 'glass wine glass') [D.B. McDonald 82].

These results are quite satisfactory taking into account the machine used - whenever the 5th generation machines [T. Motooka (ed.) 82], [D. Warren 82] become a reality this section will stand as an example of concern with anachronistic values.

As already stated, **SPIRAL** is thought of as an open (as any spiral!) kernel for a natural language interface and this means that many research directions are open to further extend its abilities. Among them, those concerned with the following, to be explored soon :

- actions to be performed when a sentence cannot be understood or is ambiguous (dialogues with the user and paraphrasing will be sought)

- means to help configurate the interface to a new domain (wherever possible those used in [M. Filgueiras, L. Moniz Pereira 82])

Acknowledgements

Antonio Porto, with whom I had many fruitful discussions on these (and other) matters, gave me an invaluable help on the development of the semantic analyser.

Luis Moniz Pereira is thanked for all his enlightening comments.

References

A. Colmerauer

75 Les Grammaires de Metamorphose
Groupe d'Intelligence Artificielle
Universite' d'Aix-Marseille II

78 Metamorphosis Grammars
in Natural Language Communication with Computers
ed. L. Bolc
Lecture Notes in Computer Science
Springer Verlag

W. Clocksin, C. Mellish, R. Fisher

80 The RT-11 Prolog System (Version NU7 with Top Level)
Software Report 5a (2nd ed.)
Department of Artificial Intelligence
University of Edinburgh

V. Dahl

77 Un Systeme Deductif d'Interrogation de Banque de Donnees en Espagnol
These de 3eme Cycle
Groupe d'Intelligence Artificielle
Universite' d'Aix-Marseille II

M. Filgueiras, L. Moniz Pereira

82 Relational Data Bases `a la carte
Departamento de Informatica, Universidade Nova de Lisboa

Proceedings of the Logic Programming Workshop 83, Portugal

G. Hirst

81 Anaphora in Natural Language Understanding : a survey
Lecture Notes in Computer Science 119
Springer Verlag

M. Kay

80 Algorithm Schemata and Data Structures in Syntactic Processing
Palo Alto Research Center, Xerox

M. McCord

80 Using Slots and Modifiers in Logic Grammars for Natural Language
Technical Report No. 69A-80
Computer Science Department, University of Kentucky

in Artificial Intelligence 18(3), 1982

81 Focalizers, the Scoping Problem and Semantic Interpretation Rules in Logic Grammars
Technical Report No. 81-81
Computer Science Department, University of Kentucky

in Proceedings of the Workshop on Logic Programming for Intelligent Systems
Logicon Inc., 1981

D. B. McDonald

82 Understanding Noun Compounds
Ph.D. Thesis
Department of Computer Science, Carnegie-Mellon University

L. Moniz Pereira, A. Porto

81 All Solutions
Logic Programming Newsletter, 2, Autumn 81

82 A Prolog Implementation of a Large System on a Small Machine
Proceedings of the First International Logic Programming Conference, Marseille, France

L. Moniz Pereira, P. Sabatier, E. Oliveira

82 ORBI : An Expert-System for Environmental Resource Evaluation through Natural Language
Proceedings of the First International Logic Programming Conference, Marseille, France

T. Motooka (ed.)

82 Proceedings of the International Conference on the Fifth Generation Computer Systems, Tokyo, Japan, 1981
North-Holland

R. Pasero

73 Representation du Francais en Logique du Premier Ordre en Vue de Dialoguer avec un Ordinateur
These de 3eme Cycle, Groupe d'Intelligence Artificielle
Universite' d'Aix-Marseille II

F. Pereira

81 Extraposition Grammars
American Journal of Computational Linguistics, vol. 7, 4

83 Logic for Natural Language Analysis
Technical Note 275
SRI International

F. Pereira, D. Warren

80 Definite Clause Grammars for Language Analysis -
A Survey of the Formalism and a Comparison with
Augmented Transition Networks
Artificial Intelligence, 13

J. F. Pique, P. Sabatier

82 An Informative, Adaptable and Efficient Natural
Language Consultable Data Base System
Proceedings of the European Conference on Artificial
Intelligence, Orsay, France

A. Porto

82 Epilog : A Language for Extended Programming in Logic
Proceedings of the First International Logic Programming
Conference, Marseille, France

A. Walker, A. Porto

83 KB01 : A Knowledge Based Garden Store Assistant
Proceedings of the Logic Programming Workshop 83, Portugal

to appear as an IBM Research Report

D. Warren

82 A View of the Fifth Generation and Its Impact
Technical Note 265
SRI International