USING THE SAME SYSTEM FOR ANALYZING AND SYNTHESIZING SENTENCES

Phillipe Rincel * and **Paul Sabatier** **

* Bull S.A., CEDIAG, 68 Route de Versailles, 78430 Louveciennes, France.
** CNRS, Groupe Intelligence Artificielle, Faculté des Sciences de Luminy, 13288 Marseille, Cedex 9, France.

ABSTRACT

We specify the advantages of guided composition of sentences and illustrate them with examples from Leader, a natural language interface we have developped. Guided composition is achieved by using the same grammar for analysis and for synthesis. We detail the problems we have encountered and we provide solutions for partial synthesis. We give the principles of the analysis-synthesis algorithm.

KEYWORDS

Natural Language Interface, Guided Composition, Analysis, Synthesis, Prolog.

1. INTRODUCTION

The quality of a natural language interface must be estimated not only in terms of linguistic coverage but also in terms of procedures dealing with unexpected expressions (incorrect formulations or correct ones, not provided by the interface). Knowing that error recovery is a complex task in the "restricted" framework of programming languages (limited syntax and rigourously defined semantics), one can appreciate the difficulty of endowing natural language interfaces with such capabilities. One can resort to flexible parsing for analysing "deviant" expressions typed by users [Carbonell and Hayes 1983], but this method can mislead them about the interface's real capabilities [Mathieu and Sabatier 1986; Sabatier 1987].

Our approach is quite different. We have developped a generator of natural language (French and English) interfaces to relational databases, Leader [Benoit et al. 1988]. An interesting characteristic is that our system can lead the user towards provided formulations in a user-friendly way. The user can compose step by step questions by means of information dynamically synthesized by Leader. The same system with the same grammar is used both in analysis and synthesis. We specify in this paper the advantages of guided composition. We detail the problems we have encountered and we provide solutions for partial synthesis. We give the principles of the analysis-synthesis algorithm.

2. ADVANTAGES OF GUIDED COMPOSITION

We may distinguish two kinds of communication with natural language interfaces :

- a "free" mode : the user types sentences without knowing the limits of the interface but he hopes it will understand him. Trivial reality : user's freedom will always be the freedom the system will grant him.
- a guided mode : the system guides the user while he composes sentences (guided composition).

Unlike the "free" mode, with guided composition users quickly perceive the limits of the interface. The designer doesn't have to program all the expressions or structures having the same meaning. Unique forms and structures are sufficient. He may forget the others. A user-friendly interface with a guided composition mode must lead users towards non ambiguous formulations, as in Leader. So, it is not necessary to produce paraphrases for want of clarification from the user.

We give now an example of a session with Leader. In this application, the system interfaces a database that contains information regarding Nobel Prizes. (The original session is in French).

The user types :

Give the ...

By a mere cursor return, the user asks to be guided. And Leader synthesizes expected expressions following the word *the* :

age
average of
country
man
nationality
person
woman

Using a mouse, the user selects and clicks the word *person*. His sentence becomes :

Give the person ...

By a cursor return, he asks for assistance. And Leader synthesizes :

that
who

The user selects *who*. And so on.

Give the person who ...

did not
refused
received

Next step : the user selects *received* and decides to continue without any assistance until the conjonction *and* :

Give the person who received the Nobel prize of Physics in 1921 and ...

At this point, Leader synthesizes :

his/her age
his/her nationality
in

And the user completes his question :

Give the person who received the Nobel prize of Physics in 1921 and his/her nationality ?

3. PROBLEMS BOUND TO PARTIAL SYNTHESIS

After the last word of an incompleted sentence composed by the user, Leader's grammar runs in synthesis and produces a list of possible following words or expressions. The main problem of this kind of synthesis (we call it *partial synthesis*) is that a word (or an expression) that has been synthesized by the system (and selected by the user to compose his sentence) must not lead to a future dead end. For example, after a noun phrase the system may synthesize the relative pronoun *who* if and only if, in the application domain, there is a verb that can take this noun phrase as subject. If there is no such a verb, the relative pronoun *who* must not be synthesized.

One can avoid dead ends by developping a semantical grammar with symbols reflecting the semantics of the application domain like in Tennant's menu-based system [Tennant 1984]. This is not the case with Leader. Leader is a generator of natural language interfaces. Leader's grammar is portable to different domains. Symbols reflect linguistical properties. Associated to particular symbols, general conditions access to the semantic model of the application domain. Because of the partial synthesis problem, calls to these conditions must be placed in the grammar before concerned symbols. Their evaluation is done before the rewriting of symbols.

The following simplified rules (in a DCG style) illustrate the principle involved in the synthesis (or not) of a relative pronoun. The general condition *possible_case* takes the concept associated to the noun and verifies if it can be a case (agent, object, etc.) of a verb. Only the different values of *possible_case* are dependant of the application domain.

```
np -->  det,
        noun(Concept),
        relative(Concept).
relative(Concept) -->
        { possible_case(Concept,Case,Verb) },
        relative_pronoun(Case),
        incomplete_sentence(Case,Verb).
relative(Concept) --> [].
```

Another problem of partial synthesis is the problem of *variable symbols*. By variable symbols, we mean words or expressions that are not defined in the lexicon of the system because they are too numerous or infinite, like integers, dates, or proprer nouns for example. If these expressions belong to those following a given word, one can't synthesize them. In this case, Leader produces a message expliciting the type associated to the expected expressions. For example, after the incomplete question :

Give the persons who received the Nobel prize of Physics before ...

Leader will synthesize :

<enter a year, example : 1945>
<enter a person, example : Einstein>

Concerning variable symbols, Leader displays messages when running in synthesis, but collects and parses expressions when running in analysis.

The right placement of calls to conditions in the grammar (not to lead to a dead end), and the management of variable symbols were the two major problems we encountered and solved with partial synthesis.

4. PRINCIPLES OF ANALYSIS-SYNTHESIS

The potential reversibility of certain programs written in Prolog is well known. So, in order to facilitate the implementation of a grammar running both in analysis and in synthesis, we have decided to program Leader in this language. The core of the system is a Metamorphosis Grammar [Colmerauer 1975] using immediate Prolog strategy : top-down, left-to-right, depth-first, non-deterministic.

In order to synthesize all the possible expressions following a given word, the grammar must contain no cuts (and no negation by failure). For example, the two following grammar rules :

```
pp(object) --> !, np.
pp(Case) --> prep(Case), np
```

must be replaced by the following ones :

```
pp(object) -->  np.
pp(Case) -->    { dif(Case,object) },
                prep(Case),
                np.
```

dif(X,Y) is the coroutine built-in predicate that controls at all times the validity of the inequation between *X* and *Y*. It fails as soon as *X* and *Y* become equal, and the program backtracks.

We give now the principles of our analysis-synthesis algorithm. To each word typed by the user (or selected by him in the synthesized list), one associates an integer corresponding to its position in the sentence. For example, for the question :

Give the persons who received the Nobel prize of Physics ?

we will have the following association :

Give (1) the (2) persons (3) who (4) received (5) the (6) Nobel (7) prize (8) of (9) Physics (10) ? (11)

The algorithm needs an integer, called *rightmost*, whose value is the integer associated to the rightmost word accepted by the grammar in the user's sentence. At the beginning of the analysis-synthesis, the value of *rigthmost* is 0. *rigthmost* increases according to the words accepted, but *rightmost* never decreases : backtracking in the application of grammar rules has no effect on *rightmost*. The algorithm needs another integer, called *current*, whose value is the integer associated to the current word to be analysed in the sentence. At the beginning of the analysis-synthesis, the value of *current* is 0. *current* increases according to the words accepted, but also can decrease when backtracking occurs in the application of grammar rules. For a given complete or incomplete user's sentence, rules of grammar are applied until terminal symbols. When a terminal symbol must be applied, the following operation is done. If the terminal symbol expected by the grammar rule matches with the current word of the sentence, we have the following situation :

If *current* > *rightmost*, then, we do :

rightmost := *current*

current := *current* + 1

else, we do:

current := *current* + 1

If the terminal symbol T expected by the grammar rule doesn't match with the current word of the grammar, the situation is :

If *current* < *rightmost* then we do nothing,

else, we record T as an expected word instead of the current word in the sentence.

At the end, if the analysis succeeds, the users's sentence is accepted. If it fails, we display the user's sentence until the word W whose associated integer has the value of *rightmost*, and we display all the terminal symbols T recorded as possible words following W. Then, the user selects an expected word and completes or not his sentence. And the sentence is analysed from the beginning.

As we mention it above, calls to conditions may occur in a grammar rule. Their evaluation can produce several solutions. It is in fact the nature of the words encountered that limits the number of solutions. The partial synthesis imposes to place any condition in a grammar rule before the concerned symbol in order to evaluate the condition before the rewriting of the symbol. This method is not efficient when the rewriting of the concerned symbol leads to a part of the sentence yet accepted. The evaluation of the condition could be done after.

So, for each call to a condition that may occur in a grammar rule, we place it before and after the concerned symbol. The condition will be evaluated before if :

current = *rightmost*

and after if :

current < *rightmost*

Colmerauer first, within the natural language interface to a database on planets, Orbis [Colmerauer and Kittredge 1982], used the same grammar for analyzing sentences and synthesizing expected words after an erroneous one. Our algorithm differs from Orbis'one on the following points. We introduce and manage *variable symbols*. We don't re-analyze the incomplete (or erroneous) sentence for synthesizing expected words : we do it in one pass. Efficiency in time is better by evaluating conditions before or after the concerned symbols according to the values of *current* and *rightmost*.

5. CONCLUSION

Partial synthesis is an interesting challenge when one decides to use the same system for analysing and synthesizing sentences. If Prolog seems to be a fairly technical solution, fundamental problems must be solved like writing sizable non ambiguous grammar with natural phenomena like proforms (pronouns, ellipsis, etc.), or mastering the control of partial synthesis for avoiding any future dead end. Leader illustrates a path we have decided to follow and investigate.

6. ACKNOWLEDGMENTS

Leader has been developped at Bull CEDIAG (Artificial Intelligence Development Center), 78430 Louveciennes, France. We thank Pascale Benoit and Dominique Vienne for their contributions.

7. REFERENCES

Benoit P., Rincel Ph., Sabatier P., Vienne D., *A User-Friendly Natural Language Interface to Oracle*, European Oracle Users'Group Conference, Paris, 1988.

Carbonell J., Hayes P., *Recovery Strategies For Parsing Extragrammatical Language*, American Journal of Computational Linguistics, 9, 3-4, 1983.

Colmerauer A., *Metamorphosis Grammars*, in Natural Language Communication With Computers, Bolc L. Ed., Springer Verlag, 1978; First appeared as *Les Grammaires de Métamorphose*, GIA Report, Luminy, Université Aix-Marseille 2, 1975.

Colmerauer A., Kittredge R., *ORBIS*, 9th International Conference on Computational Linguistics, COLING, 1982.

Mathieu Y., Sabatier P., *Interfacile : Linguistic Coverage and Query Reformulation*, 11th International Conference on Computational Linguistics, COLING, 1986

Sabatier P., Contribution au développement d'interfaces en langage naturel, Thèse d'Etat, Université Paris 7, 1987.

Tennant H., *Menu-Based Natural Language Understanding*, National Computer Conference, 1984.