RESPONSE GENERATION IN QUESTION - ANSWERING SYSTEMS

Ralph Grishman
New York University

1. INTRODUCTION

As part of our long-term research into techniques for information retrieval from natural language data bases, we have developed over the past few years a natural language interface for data base retrieval [1,2]. In developing this system, we have sought general, conceptually simple, linguistically-based solutions to problems of semantic representation and interpretation. One component of the system, which we have recently redesigned and are now implementing in its revised form, involves the generation of responses. This paper will briefly describe our approach, and how this approach simplifies some of the problems of response generation.

Our system processes a query in four stages: syntactic analysis, semantic analysis, simplification, and retrieval (see Figure 1). The syntactic analysis, which is performed by the Linguistic String Parser, constructs a parse tree and then applies a series of transformations which decompose the sentence into a operator-operand-adjunct tree. The semantic analysis first translates this tree into a formula of the predicate calculus with set-formers and quantification over sets. This is followed by anaphora resolution (replacement of pronouns with their antecedents) and predicate expansion (replacement of predicates not appearing in the data base by their definitions in terms of predicates in the data base). The simplification stage performs certain optimizations on nested quantifiers, after which the retrieval component evaluates the formula with respect to the data base and generates a response.

Our original system, like many current question-answering systems, had simple mechanisms for generating lists and tables in response to questions. As we broadened our system's coverage, however, to include predicate expansion and to handle a broad range of conjoined structures, the number of *ad hoc* rules for generating answers grew considerably. We decided therefore to introduce a much more general mechanism, for translating predicate calculus expressions back into English.

2. PROBLEMS OF RESPONSE GENERATION

To understand how this can simplify response generation, we must consider a few of the problems of generating responses. The basic mechanism of answer generation is very simple. Yes-no questions are translated into predicate formulas; if the formula evaluates to *true*, print "yes", else "no". Wh-questions translate into set-formers; the extension of the set is the answer to the question.

One complication is embedded set-formers. An embedded set-former arises when the question contains a quantifier or conjunction with wider scope than the question word. For example, the question

Which students passed the French exam and which failed it?

will be translated into two set-formers connected by *and*:

{s ∈ set-of-students | passed (s, French exam)}

and

{s ∈ set-of-students | failed (s, French exam)}

It would be confusing to print the two sets by themselves. Instead, for each set to be printed, we take the predicate satisfied by the set, add a universal quantifier over the extension of the set, and convert the resulting formula into an English sentence. For our example, this would mean

print-English-equivalent-of'(∀x ∈ S_1)

passed (x, French exam)'

where S_1 = {s ∈ set-of-students | passed(s,French exam)}

and

print-English-equivalent-of'(∀x ∈ S_2)

failed (x, French exam)'

where S_2 = {s ∈ set-of-students | failed(s,French exam)}

which would generate a response such as

John, Paul, and Mary passed the French exam;
Sam and Judy failed it.

The same technique will handle set-formers within the scope of quantifiers, as in the sentence

Which exams did each student take?

Additional complications arise when the system wants to add some words or explanation to the direct answer to a question. When asked a yes-no question, a helpful question-answering system will try to provide more information than just "yes" or "no". In our system, if the outermost quantifier is existential — (∃x ∈ S) C(x) — we print {x ∈ S | C(x)}; if it is universal — (∀x ∈ S) C(x) — we print {x ∈ S | ¬C(x)}. For example, in response to

Did all the students take the English exam?

our system will reply

No, John, Mary, and Sam did not.

When the outermost quantifier is the product of predicate expansion, however, it is not sufficient to print the corresponding set, since the predicate which this set satisfies is not explicit in the question. For example, in the data base of radiology reports we are currently using, a report is *negative* if it does not show any positive or suspicious medical findings. Thus the question

Was the X-ray negative?

would be translated into

negative (X-ray)

and expanded into

(∀f ∈ medical-findings) ¬show(X-ray,f)

so the system would compute the set

{f ∈ medical-findings | show(X-ray,f)}

Just printing the extension of this set,

No, metastases.

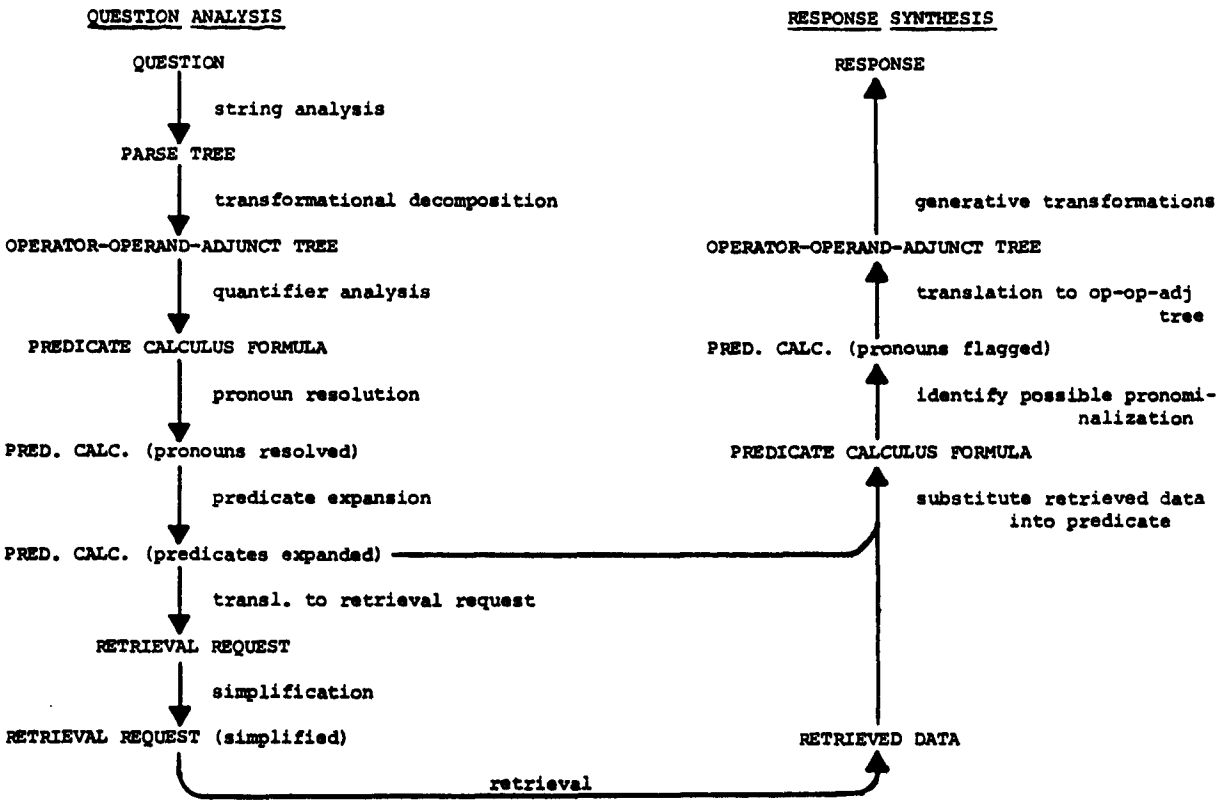

Figure 1. The structure of the NYU question-answering system.

would be confusing to the user. Rather, by using the same rule as before for printing a set, we produce a response such as

No, the X-ray showed metastases.

Similar considerations apply to yes-no questions with a conjunction of wide scope.

3. DESIGN AND IMPLEMENTATION

As we noted earlier, our question-analysis procedure is composed of several stages which transform the question through a series of representations: sentence, parse tree, operator-operand-adjunct tree (transformational decomposition), predicate calculus formula, retrieval request. This multi-stage structure has made it straightforward to design our sentence generation, or synthesis, procedure, which constructs the same representations in the reverse order from the analysis procedure.

In designing the synthesis procedure, the first decision we had to make was: which representation should the synthesis procedure accept as input? The retrieval procedure instantiates variables in the retrieval request, so it might seem most straightforward for the retrieval procedure to pass to the synthesis procedure a modified retrieval request representation. Alternatively, we could keep track of the correspondence between components of the retrieval request and components of the parse tree, operator-operand-adjunct tree, or predicate calculus representation. Then we could substitute the results of retrieval back into one of the latter representations and have the synthesis component work from there. This would simplify the synthesis procedure, since its starting point would be "closer" to the sentence representation.

A basic requirement for using one of these representations is then the ability to establish a correspondence between those components of the retrieval request which may be significant in generating a response and components of the other representation. Because predicate expansion introduces variables and relations which are not present earlier but which may have to be used in the response, we could not use a representation closer to the surface than the output of predicate expansion (a predicate calculus formula). Subsequent stages of the analysis procedure, however, (translation to retrieval request and simplification), do not introduce structures which will be needed in generating responses. We therefore choose to simplify our synthesizer by using as its input the output of predicate expansion (instantiated with the results of retrieval) rather than the retrieval request.

The synthesis procedure has three stages, which correspond to three of the stages of the analysis procedure (Figure 1). First, noun phrases which can be pronominalized are identified. Second, the predicate calculus expression is translated into an operator-operand-adjunct tree. Finally, a set of generative transformations are applied to produce a parse tree, whose frontier is the generated sentence.

The correspondence between analysis and synthesis extends to the details of the analytic and generative transformational stages. Both stages use the same program, the transformational component of the Linguistic String Parser [3]. Most analytic transformations have corresponding members (performing the reverse transformations) in the generative set. These correspondences have greatly facilitated the design and coding of our generative stage.

One problem in transforming phrases into predicate calculus and then regenerating them is that syntactic paraphrases will be mapped into a single phrase (one of the paraphrases). For example, "the negative X-rays" and "the X-rays which were negative" have the same predicate calculus representation, so only one of these structures would be regenerated. This is undesirable in generating replies — a natural reply will, whenever possible, employ the same syntactic constructions used in the question. In order to generate such natural replies, each predicate and quantifier which is directly derived from a phrase in the question is tagged with the syntactic structure of that phrase. Predicates and quantifiers not directly derived from the question (e.g., those produced by predicate expansion) are untagged. Generative transformations use these tags to select the syntactic structure to be generated. For untagged constructs, a special set of transformations select appropriate syntactic structures (this is the only set of generative transformations without corresponding analytic transformations).

4. OTHER EFFORTS

As we noted at the beginning, few question-answering systems incorporate full-fledged sentence generators; fixed-format and tabular responses suffice for systems handling a limited range of quantification, conjunction, and inference. However, several investigators have developed procedures for generating sentences from internal reprsentations such as semantic nets and conceptual dependency structures [4,5,6,7].

Sentence generation from an internal representation involves at least three types of operations:

- recursive sequencing through the nested predicate structure
- sequencing through the components at one level of the structure
- transforming the structure or generating words of the target sentence.

The last function is performed by LISP procedures in the systems cited (in our system it is coded in Restriction Language, a language specially designed for writing natural-language grammars). The first two functions are either coded into the LISP procedures or are performed by an augmented transition network (ATN). Although the use of ATNs suggests a parallelism with recognition procedures, the significance of the networks is actually quite different; a path in a recognition ATN corresponds to the concatenation of strings, while a path in a generative ATN corresponds to a sequence of arcs in a semantic network. In general, it seems that little attention has been focussed on developing parallel recognition and generation procedures.

Goldman [5] has concentrated on a fourth type of operation, the selection of appropriate words (especially verbs) and syntactic relations to convey particular predicates in particular contexts. Although in general this can be a difficult problem, for our domain (and probably for the domains of all current question-answering systems) this selection is straightforward and can be done by table lookup or simple pattern matching.

5. CONCLUSION

We have discussed in this paper some of the problems of response generation for question-answering systems, and how these problems can be solved using a procedure which generates sentences from their internal representation. We have briefly described the structure of this procedure and noted how our multistage processing has made it possible to have a high degree of parallelism between analysis and synthesis. We believe, in particular, that this parallelism is more readily achieved with our separate stages for parsing and transformational decomposition than with ATN recognizers, in which these stages are combined.

The translation from predicate calculus to an operator-operand-adjunct tree and the generative transformations are operational; the pronominalization of noun phrases is being implemented. We expect that as our question-answering system is further enriched (e.g., to recognize presupposition, to allow more powerful inferencing rules) the ability to generate full-sentence responses will prove increasingly valuable.

6. ACKNOWLEDGEMENTS

I would like to thank Mr. Richard Cantone and Mr. Ngô Thanh Nhàn, who have implemented most of the extensions to our question-answering system over the past year.

This research was supported in part by the National Science Foundation under Grant No. MCS 78-03118, by the Office of Naval Research under Contract No. N00014-75-C-0571, and by the Department of Energy, under Contract No. EY-76-C-02-3077.

7. REFERENCES

[1] R. Grishman and L. Hirschman, Question Answering from Natural Language Medical Data Bases, *Artificial Intelligence 11* (1978) 25-43.

[2] R. Grishman, The Simplification of Retrieval Requests Generated by Question-Answering Systems, *Proc. Fourth Intl. Conf. on Very Large Data Bases* (1978) 400-406.

[3] J. R. Hobbs and R. Grishman, The Automatic Transformational Analysis of English Sentences: An Implementation. *Intern. J. Computer Math. A* 5 (1976) 267-283.

[4] R. Simmons and J. Slocum, Generating English Discourse from Semantic Networks. *Comm. A.C.M. 15* (1972) 891-905.

[5] N. Goldman, Sentence Paraphrasing from a Conceptual Base. *Comm. A.C.M. 18* (1975) 96-106.

[6] H. Wong, Generating English Sentences from Semantic Structures. *Technical Report No. 84*, Dept. of Computer Sci., Univ. of Toronto (1975).

[7] J. Slocum, Generating a Verbal Response. In *Understanding Spoken Language*, ed. D. Walker, North-Holland (1978) 375-380.