INFORMATION SCIENCES

Volume 21
Number 2
1980

An International Journal

Abstracted in CHEMICAL ABSTRACTS; COMPUTER AND INFORMATION SYSTEMS; COMPUTING REVIEWS; ENGINEERING INDEX; LANGUAGE AND AUTOMATION; LANGUAGE AND LANGUAGE BEHAVIOR ABSTRACTS; MATHEMATICAL REVIEWS; REFERATIVNYÏ ZHURNAL MATHEMATIKA; SCIENCE ABSTRACTS; ZENTRALBLATT FÜR MATHEMATIK.

INFORMATION SCIENCES 21, 159–177 (1980)

Quasinatural Language in Consultation Systems

GIUSEPPINA GINI

and

MARIA GINI

Istituto di Elettrotecnica ed Elettronica Politecnico, Milan, Italy

Communicated by A. M. Andrew

ABSTRACT

Following the development of question-answering systems, consultation systems have emerged as a useful integration between data-base management, knowledge representation, and natural-language interaction with the computer. The idea developed in this paper is to insert in a consultation system an input language which is formal but natural-oriented. That is a feasible input, which allows a natural communication with the computer without involving too many theoretical difficulties and practical disadvantages in terms of cost and execution time. The paper illustrates an implementation based on the classical facilities of the goal-oriented languages. Using MICROPLANNER as implementation language, we directly obtain the deep structure of the input quasinatural language expressed in MICROPLANNER, and this makes the deduction activity more reliable and direct. The problems related to knowledge representation and basic functions of the system are discussed; examples of its use are illustrated.

1. INTRODUCTION

Many programming tasks in artificial intelligence require the manipulation of data bases in order to find information and to infer new knowledge. These programs need to perform complex retrieval operations in the data base together with problem-solving activities [15].

The project we have developed was aimed at designing an interactive system which could be consulted to help the user in solving many classes of problems. Our system applies its problem-solving ability to a given domain of knowledge in order to suggest possible solutions. Consultation systems are multiaspect systems, based on the capability of storing and retrieving information and of inferring solutions to new questions or problems.

0020-0255/80/05159-19$01.75

The recent trends in artificial intelligence (AI) are oriented to its application to real-world problems, characterized by a large amount of task-specific knowledge. The direction of designing knowledge-based consultation systems represents a change from the previous direction of general problem solvers. Earlier work demonstrated that only toy examples could be solved by the general rules of the problem solver. Consultation systems emphasize the acquisition of large amounts of information in a given domain and develop domain-specific techniques to solve problems. There are many examples of systems based on that philosophy, such as MACSYMA [10], MYCIN [20], and POINTY [7].

From the data-base point of view, a consultation system is a data base in which some procedures can produce answers which are not explicitly present in the data base but are logical consequences of the facts stored there [12, 14, 19]. Knowledge stored in the data base is about facts as well as about relations between them. Moreover, a consultation system should have some learning capabilities [1] to make it possible to adapt the system to new situations through its previous experience.

Another important aspect of consultation systems is ease of access to the system; the input query language tends to be the natural language [6, 20]. Research in consultation systems is parallel but not related to research in natural-language understanding. The present state of the art in natural-language understanding and its computational difficulty suggest the implementation of easier input languages in consultation systems.

The idea we discuss in this paper is to define an input which is formal but natural-oriented, a quasinatural language. We demonstrate that this is a feasible input, which allows fairly natural communication with the computer, and avoids many theoretical and practical difficulties. A system implementing this idea, QUASI, is then illustrated.

We divide the presentation to treat three aspects:

(1) *Input translation.* This process transforms the input language sentences into the internal language in order to access the knowledge stored in the system.

(2) *Knowledge acquisition.* This process allows introducing new knowledge into the data base of the system after controlling for possible inconsistency between data and relations already in the data base.

(3) *Answer extraction.* This process gives answers to questions. A search is made in the data base to check whether a question is a consequence of the data-base-stored knowledge.

A crucial point is the choice of the programming language for the system implementation; this is made easier if the deep structure of the input language, the data-base sentences, and the program are expressed in the same language.

A classical way to obtain knowledge-based systems is based on first-order predicate logic. The answer is viewed as a theorem to be demonstrated, and the data base is written in form of logical sentences [3, 8, 17, 18].

A more practicable internal language, according to our requirements, is a PLANNER-like language [4, 9, 13, 16, 21]. In this approach a system is suitable for representing and manipulating information expressed as procedures. The question is viewed as a goal to be achieved, and the answer is obtained when the problem-solving activity is successful. During the execution the achievement of the goal is attempted by direct action; if that fails, by the introduction of some lower-level goals.

In Sec. 2 the project of our consultation system is discussed. The input language is proposed; the characteristics of a formal and quasinatural language are outlined, and implementation problems are discussed.

In Sec. 3 the memory organization is illustrated and some features of present goal-oriented languages are investigated. In particular, the steps of adding new information and of building answers are examinated.

In Sec. 4 the organization of QUASI is analyzed and an example of its use is given.

In the last section new research directions are outlined and concluding remarks are made.

2. INPUT LANGUAGE AND TRANSLATION

The choice of the input language for a consultation system should be connected with the general problem of representing knowledge in a computer in order to allow some reasoning activity. In [11] two criteria for adequacy of a representation are outlined. A representation is *epistemologically* adequate if it can be used to express facts that one actually has about the aspects of the world. A representation is *heuristically* adequate if the reasoning process necessary to solve a problem is expressible in the language.

One epistemologically adequate representation is, of course, the natural language, but other representations could be more suitable for our task. For instance, formal languages based on predicate logic could be epistemologically adequate representations. In addition, PLANNER-like languages integrate epistemological and heuristic information and supply a single notation for expressing these together.

In order to discuss the choice of the input language for our system, we divide the problem of representing knowledge into three parts [8]:

(1) *Determination of the relevant semantic content of data.* For example, we may decide that the semantics of the sentence "John is friend of Brian" are expressed by the binary relation "friend of" applied to the objects John and Brian.

(2) *Choice of a language in which to express this semantic content.* For example, we may use the notation of formal logic.

(3) *Choice of an internal representation for the language.* For example, a binary relation may be expressed by a list of three elements: the first is the name of the relation, and the other two are arguments.

Let us discuss the different choices.

A. A FORMAL LANGUAGE AS INPUT LANGUAGE FOR A CONSULTATION SYSTEM

The use of predicate logic as input language for a question-answering system has been proposed in [8]. Such a system employs first-order predicate logic as input language and a theorem prover based on the resolution principle as deductive mechanism.

Some of the problems that one encounters in expressing natural-language sentences in predicate logic are outlined in [17], [18].

The first problem is in reducing the natural-language constructions, which are higher-order level, to the first-order. For example, if we want to represent the sentence "S is good" by the predicate "Good(S)", then the sentence "S is better than T" might be expressed by the function "More(Good)(S, T)". "More" is a second-order function which maps a monary first-order predicate into a binary first-order predicate.

Higher-order logic sentences are useful for human understanding, but are computationally very difficult. Research in automatic theorem provers has made little progress towards theorem provers for higher-order logic. Normally first-order logic is employed, and higher-order logics are simulated.

The higher-order constructs in natural language can be easily expressed in first-order predicate calculus. The method is to reexpress what used to be a predicate as individual and to use a new predicate for the application. The first sentence of the previous example is then expressed by "Is(S, Good)", and the second sentence by "Is(S, more(Good, T))", where "more" is a function

$$\text{more} : \{\text{properties}\} \times \{\text{objects}\} \rightarrow \{\text{properties}\}.$$

In this way the representation of a sentence is obtained by the definition of a suitable set of functions, relations, and axioms. An automatic translation from natural language into logic can thus be obtained.

The well-known frame problem is present in this formal approach [8, 11, 17]. Though several authors have discussed it, a complete solution has never been given. The solution of the frame problem presented by Hewitt in PLANNER [9] is especially interesting because in that system the representation of the world is updated in a computationally simple way.

B. A NATURAL LANGUAGE AS INPUT FOR A CONSULTATION SYSTEM

The first complete example of an artificial-intelligence system based on natural language is Winograd's system [22]. This system contains the syntactic, semantic, and deductive capabilities required for representing information about a world of blocks. The key idea, derived from the procedural-embedding thesis [9], is that descriptions in English can be translated into descriptions in the form of procedures. Winograd's system was implemented in LISP and MICROPLANNER [21] for a DEC PDP-10 computer and operated with a vocabulary of 200 words.

Let us examine the organization of the system and the storage allocation, which are illustrated in Figs. 1 and 2. The system occupies 80K words of memory. The information used for the deduction activity requires about 16K words, while the information necessary for understanding natural language requires 36K words.

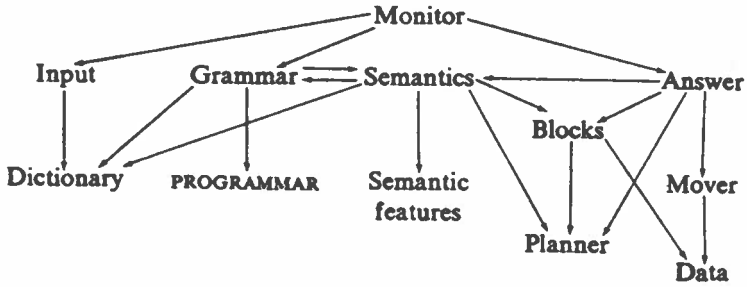


Fig. 1.

	Parser	Semantics	Deduction	Other
Interpreters 26K	PROGRAMMAR		PLANNER	LISP
English 22.5K	Grammar	Semantics		
Domain 16.5K	Dictionary	Dictionary	Blocks	
Data 2.5K			Data	Display
Free memory 12K				

Fig. 2.

Many other systems for natural-language understanding have been developed, but the general problem is still open. In any case, we believe the problem of understanding natural language is not crucial for a consultation system, because that does not require all the generality of natural language. We can obtain satisfactory results by using a quasinatural input language. We shall point out how a quasinatural language combines the advantages of the formal languages with the advantages of natural man-machine communication.

C. *A NATURAL AND FORMAL LANGUAGE AS INPUT*

A PLANNER-like language can be modified to obtain a quasinatural language. In PLANNER, in fact, it is possible to express directly concepts which correspond to high-level logic, because no distinction is made between predicates, variables, and functions. For that reason, a pattern in PLANNER is quite similar to a sentence expressed in natural language. The example introduced above can be expressed by the pattern "(S IS-GOOD)" and "(S IS-GOOD MORE-THAN T)".

The representation and the organization of knowledge is important for the deductive process, because every pattern is either a piece of knowledge or a tag of a memory container, which allows deduction of the information expressed in the tag. Therefore it is important to define simple patterns and to have a way of matching the patterns of the given knowledge with the patterns of the questions posed to the system. We can develop a system that is easier to use and more natural-oriented than PLANNER without introducing all the problems of natural-language understanding systems.

Some ideas in that direction have been developed, for instance, in CROMOS [2]. The user gives statements or questions in a stylized form of English. A simple parser program translates the input into an internal language, which is LISP; after the evaluation a small unparser program gives the output in the English form.

Another similar approach was developed in [5]. This system accepts a very limited subset of English as input, and a parser translates it into POPLER, a PLANNER-like language. The answering of a question takes place in two steps: the question is compiled into a piece of program which represents the meaning of the question, and then the program generates answers. When new information is added, the system automatically eliminates the old items which conflict with new statements. In Fig. 3 the sequence of actions is illustrated.

Fig. 3.

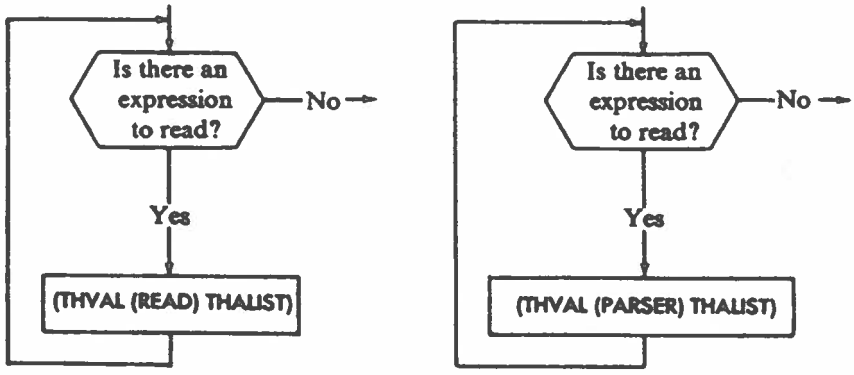


Fig. 4.

Similarly, we propose here a quasinatural language based on the present implementation of MICROPLANNER [21] and a suitable parser.

The syntactic form of any language can be improved without modifying the structure of the language. This is termed syntactic sugar. Syntactic sugar in MICROPLANNER can reduce the number of parentheses (the most troublesome aspect of LISP languages) and introduce natural-language constructs.

The insertion of a parser into the MICROPLANNER interpreter is quite natural. The basic loop of the interpreter applies the evaluation function THVAL to the read expression instantiated with the values of the variables memorized in the stack THALIST. We substitute for the call of the function READ the call of a new function PARSER, which activates the parser program. This is illustrated in Fig. 4.

3. MEMORY ORGANIZATION

In this section we examine the problems related to the organization of a consultation system, with emphasis on two steps:

(1) the introduction of new information into the system;
(2) the extraction of the answer to a problem posed as a question.

These two activities, which depend on the memory organization and the deductive process, are closely related and should be considered together. A fundamental problem in the design of a consultation system is to decide "what, how, and where" the information has to be stored in order to allow a suitable deductive process for the answer generation.

A. MEMORY ORGANIZATION AND DEDUCTION

An important choice for any problem solver, and therefore for any consultation system, concerns the level of knowledge necessary to the program. On

one hand the program might be given only the fundamental laws and premises from which it can derive all the knowledge required to answer a question. On the other hand the system might be explicitly provided with precomputed answers to commonly occurring problems.

We think that a compromise between these choices must be tailored to the specific requirements of each application domain, in order to determine its structure. The computer, the programming language employed, the amount of data, and the kind of questions should be considered.

A distinction should be made between a general-purpose and a special-purpose system. The goal in designing a special-purpose system is to achieve good performance, in terms of time and memory required to solve a problem. The solution can be obtained by implementing specialized subroutines. A general-purpose system, on the other hand, has to be constructed so as to allow the addition of new knowledge (in the same or in another domain) during the operation of the system.

The classical general-purpose systems have been based on a formal language, such as predicate calculus, and on a theorem prover. In that case the user need provide only the factual knowledge about the domain; the control knowledge is supplied by the system.

The QUASI system represents a change from the previous attempts at general problem solvers. We see computer-based consultation systems as complex and efficient systems that incorporate specialized bodies of knowledge and make that knowledge available to users who are not computer experts.

The procedures necessary for each domain have to be explicitly defined by the designer, who is responsible for organizing the data base and the deductive process. The choice of a PLANNER-like language as support language allows flexibility and modifiability, typical of a general-purpose system, together with efficiency, given by the explicit use of control information.

An important problem is to avoid inconsistencies in information stored in the data base. While in the systems based on formal logic the new information is checked for consistency before acceptance, in nonformal systems no control exists and the responsibility rests on the user.

A crucial aspect is the average amount of computation necessary to answer a question. An obvious measure of the difficulty is the average distance of the answer from the question, measured, for instance, in terms of the number of inference steps required. This measure is called the depth of the question [8].

Another factor affecting the search effort is the number of different questions that are answerable. This number depends on the amount of information given to the system, and on the relation between the memory organization and the deduction process as we shall illustrate. To increase the number of different questions answerable, without increasing the deduction effort, it is possible to increase the size of the data base, or to expand the capabilities of

the answer computation mechanism, or finally, to find a reasonable compromise between these two methods.

A very interesting aspect is that new information modifies the system. As an item of information is entered, the performance of the system is modified. The new information may have effects on the answer generation in the following ways:

(1) new information provides the answer to a new problem;
(2) new information provides the capability to get answers to a new class of questions or problems;
(3) new information provides a new procedure for answering a class of questions;
(4) new information modifies the knowledge representation;
(5) new information modifies the strategies of the program.

These changes are obtained in a quite simple way in a system based on a PLANNER-like language; we will point out these aspects later.

B. *FEATURES OF* LISP *AND* MICROPLANNER

A useful feature of LISP allows memorization of relations between different objects. This is obtained through the property-list (p-list) mechanism. In LISP each atom can have a list of properties, i.e. a list of indicator-value pairs. When we have the relation "BOX1 is in the position P1", we may place in the p-list of the atomic symbol BOX1 the value P1 under the indicator "position". The atomic symbol BOX1 provides an entry point to the information "BOX1 is in the position P1". The first argument of the relation, BOX1, is not stored explicitly with the relation, but is implied by the fact that the indicator-value pair occurs in the p-list of BOX1.

With this organization it is easy to find information about the position of BOX1, but it is difficult to find the information necessary to answer a question about the name of the box which is in the position P1. Therefore, it is necessary to have cross references. This idea is the basis of the memory organization in MICROPLANNER.

Assertions and patterns are stored as a list in the p-list of the items which appear in them; each item points to all the assertions and theorems in which it appears. Occurrences of variables are stored in the global atom called THVRB.

An assertion is "dotted" with NIL or some property of the assertion. For example, the assertion

```
(THASSERT (BOX1 AT P1))
```

yields the structure ((BOX1 AT P1)), which appears within the p-list of the items BOX1, AT, and P1, under the attribute THASSERTION.

All the assertions and all the names of theorems (i.e. deduction rules) which contain patterns of a certain length or in a certain position are on a list, which is prefixed with two integers. The first integer is the length of the pattern; the second is the number of assertions or theorem names in the list. In the previous example we have in the p-list of the atom AT

(3 1 ((BOX1 AT P1)))

Suppose we have defined a theorem PUSHBOX, whose pattern is

(X AT Y)

We find in the p-list of the atom AT under the indicator THCONSE the structure

(3 1 PUSHBOX)

All the structures in which a given item occurs in the same position are entered into another list and prefixed with an integer which is the occurrence position of that item. In the example we have

(2 (3 1 ((BOX1 AT P1))))

(2 (3 1 PUSHBOX))

All these lists are kept sorted into one list, chosen from four available lists, with an atom NIL "dotted" on the front, depending on whether the occurrence came from an assertion or from one of the three kinds of theorems.

The deductive process in MICROPLANNER follows this memory organization. The main mechanism used is the top-down activation of the consequent theorems.

Each theorem of consequent type is characterized by a pattern like (X AT Y) and by a sequence of instructions. The theorem is written in such a way that the body implies the pattern; if we want to demonstrate the truth of the pattern we must demonstrate, i.e. successfully execute, the body of the theorem.

Every step of the theorem is an instruction, whose evaluation gives "success" or "failure" as result. The possibility of accomplishing successive deductions is obtained by calling other theorems within a theorem.

A question is posed to the system in the form of a goal; the system searches by pattern-matching among the assertions and then among the theorems the piece of knowledge which allows an answer to the question. If different assertions or theorems can match the pattern of the goal, the system makes an arbitrary choice, backing up and trying another choice in case of failure.

The solution proceeds normally in a top-down or goal-oriented way. In other words, the problem is reduced to subproblems, with the goal of reducing the original problem to a set of solved subproblems.

There is also the bottom-up procedure. In this case, new assertions are derived from the old ones with the goal of deriving a solution to the original problem. This procedure is managed by ANTECEDENT theorems or by ERASING theorems. Typically CONSEQUENT theorems are employed for making deductions, while the ANTECEDENT theorems allow expanding the assertions and the ERASING theorems allow removing an item of information from the data base. This procedure is usually more costly in time and memory.

A MICROPLANNER-based system is very flexible. With it, in a simple way, we can:

(1) introduce a new assertion to provide the answer to a new question;

(2) introduce a new theorem to provide the answer to a new class of questions;

(3) introduce a new theorem with the same pattern as an existent theorem to provide a new procedure for answering a class of questions;

(4) erase an item of information to modify the knowledge of the system;

(5) modify a theorem to change the strategy of the program.

4. THE QUASI SYSTEM AND THE DATA BASE OF UNITED STATES PRESIDENTS

We will illustrate how the QUASI system is organized and operates. The system is implemented in MICROPLANNER and runs on a UNIVAC 1108 computer.

Since the QUASI system supplies the memory management together with the query language, we will illustrate both these aspects.

The QUASI language, introduced to express queries about the data base, is a structured natural language. The objective of this language is to provide a simple means of expressing the procedures which allow the obtaining of information from the data base.

At the present time it is not practicable for a computer program to accept informal natural-language messages and to analyze them at a reasonable cost. Most of the information processing systems are characterized by highly stylized queries. The area of discourse is well structured. It seems to us reasonable to translate restricted natural-language queries into the rigid formalism of internal procedures.

The other facilities of QUASI will also be introduced. They make it possible to query, to insert, to delete, and to update relations, as well as to create new relations.

A. *THE* QUASI *LANGUAGE*

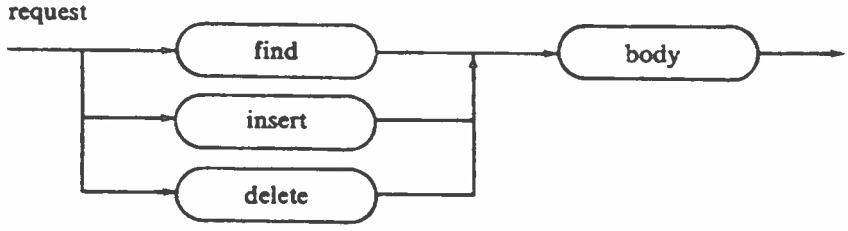
The syntax of a structural subset of natural language has been defined, and a parser program translates natural-language queries into MICROPLANNER sentences, to access the deductive mechanism.

Let us discuss some characteristics of our approach. Suppose we have a well-defined domain (for instance, the presidents of United States), from which we want to obtain information on (for instance, the year of election of a president). Every query against the data-base may be defined by the action FIND, to indicate what we want to obtain. FIND ANY may be used to find an item corresponding to the request, and FIND ALL may be used to find all the items which correspond to given conditions. Every insertion or deletion in the data base may be accomplished by the verb INSERT or DELETE. The body of any request indicates an object, in general specified by prepositional clauses. The nouns are names, of presidents for instance, or are words strictly connected with the semantics of the discourse. These special words, called elements, allow a precise semantics for the sentence.

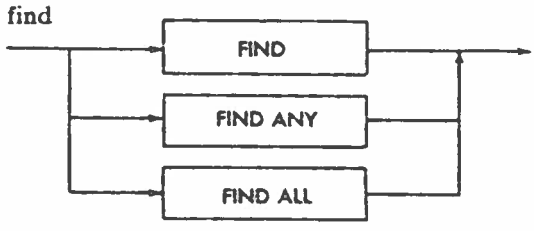
The translation to the appropriate MICROPLANNER theorem or assertion is quite easy, since the names and the patterns of theorems and assertions are known.

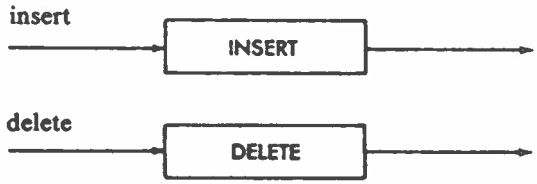
The syntax of the QUASI language is defined through the classical method of syntactic graphs. We will now illustrate a part of them, defining the most external part of the syntax. In our diagrams, nonterminal symbols are written inside an oval, terminal symbols inside a box.

The graph

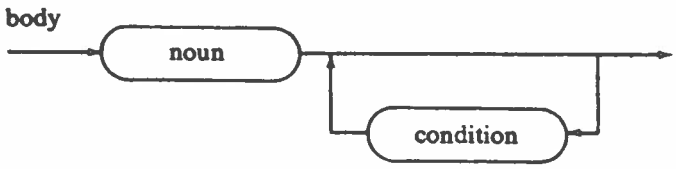


defines the structure of any user request. The action to be taken is defined as

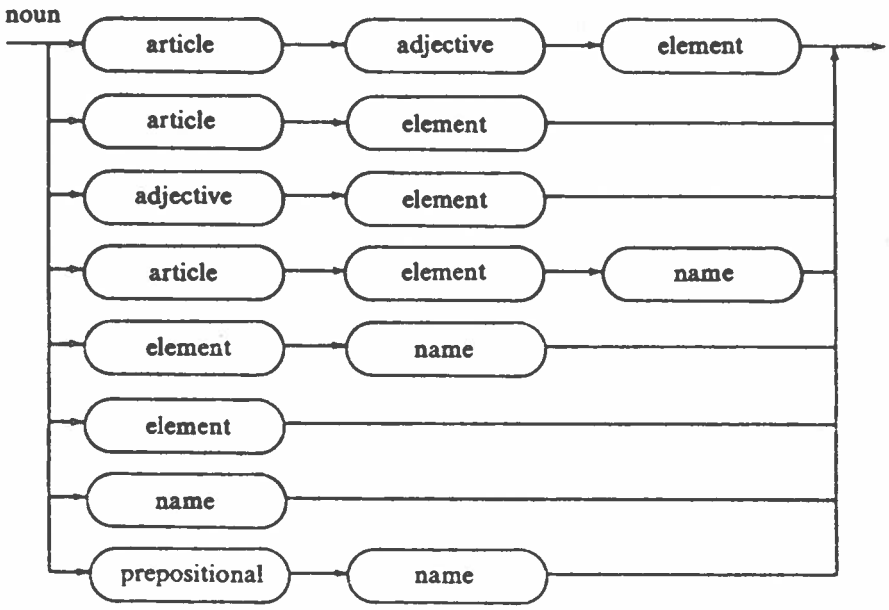


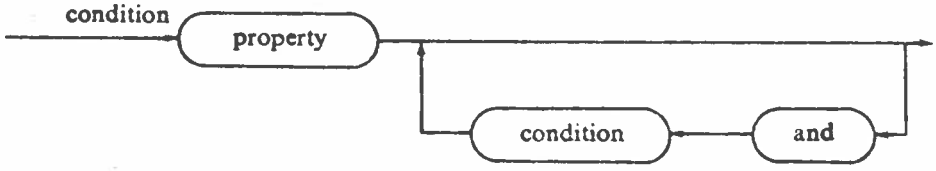


The body is a noun followed or not by conditions:



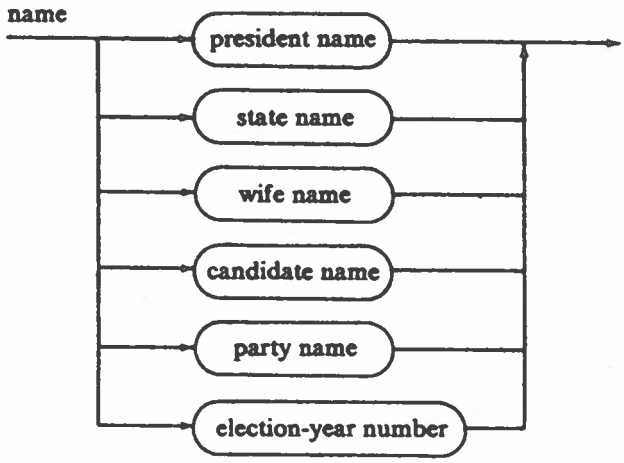
where the graphs

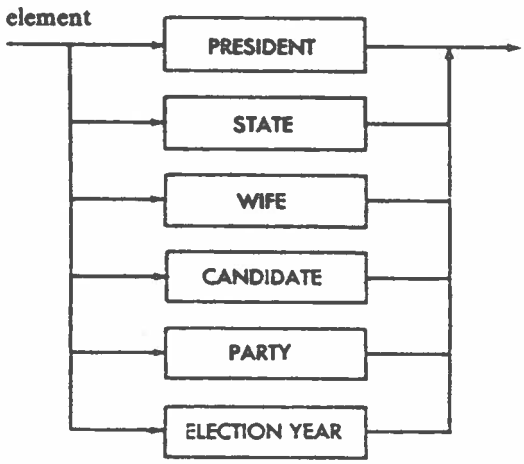


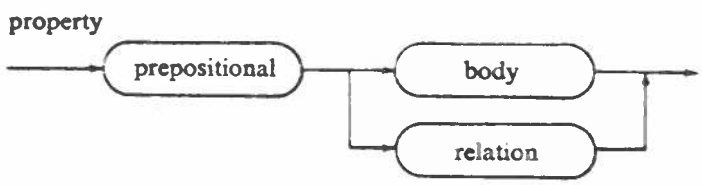


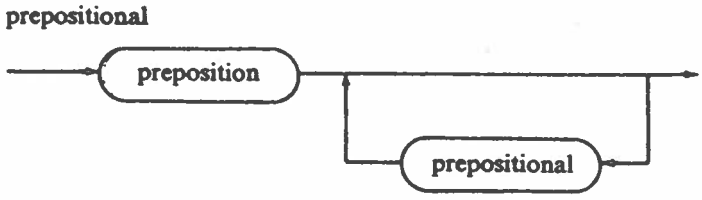
define the general-purpose part of the language.

The definition of the lower-level part of the syntax depends more on the application field. In our application, for the United States presidents data base, we add the following graphs:









It is interesting to point out that the classes "president name", "state name", and so on are defined according to the internal structure of the data base. During execution they also allow checking the correctness of the request.

In our version of QUASI the parser is a top-down parser, which analyzes the input string to construct the syntactic tree. More, looking at the classes "name" and "element", the parser individuates some semantics about the sentence.

In Fig. 5 we will give as example the syntactic tree generated in parsing the sentence "FIND ANY REPUBLICAN PRESIDENT FROM OREGON".

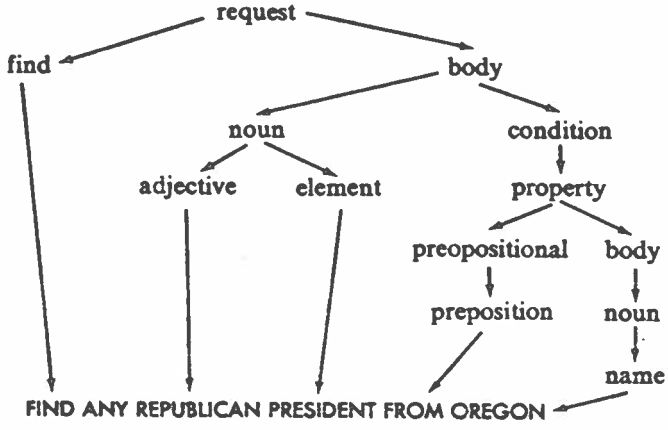


Fig. 5.

B. *THE MEMORY ORGANIZATION IN* QUASI

Knowledge about the U.S. presidents is embodied in a PLANNER data base. Knowledge is organized in classes, according to the "element" graph of our language. The LISP atoms PRESIDENTS, STATE, WIFE, CANDIDATE, PARTY, ELECTION-YEAR contain in their property-list the complete records of presidents, states, etc.

Another part of the PLANNER data base is made up of some "theorems", expressing some logical consequences of the given information. For instance, when we ask for the losing party in an election year, we may find in the record

of the president elected in that year the party, and then give the name of the other party as the answer.

In Fig. 6 we illustrate the general form of president records and give some instances.

```
(PUT
'PRESIDENT
'RECORD
'(PRES_NAME PARTY STATE WIFE))

(KENNEDY    DEM    MASS  JACKIE)
(JOHNSON    DEM    TEXAS LADY_BIRD)
(NIXON      REP    CA    PAT)
```

Fig. 6.

In a similar way we organize all the data. For instance, the knowledge about the candidates has the similar structure shown in Fig. 7.

```
(PUT
'CANDIDATE
'RECORD
'(EL_YEAR  NAME        VOTES WINNER))

(1960      KENNEDY     303   YES)
(1960      NIXON       219   NO)
(1964      JOHNSON     486   YES)
(1964      GOLDWATER 52      NO)
```

Fig. 7.

A typical internal form of a query against the data base is

(THGOAL pattern)

whose meaning is:

if the pattern is present in the data base, the answer is the pattern; otherwise the answer is NIL.

Another form of question is

(THGOAL pattern $T)

whose meaning is analogous to the previous one in the case where the pattern is asserted in the data base; otherwise the system calls a theorem in order to deduct the answer.

A FIND question is usually translated into one of the following forms. The internal form of FIND ANY questions is

```
(THPROG(x)
   (THGOAL pattern)(THRETURN x))
```

whose meaning is:

if there exists a value of x (a variable present in the pattern) which makes the pattern true, then the answer is that value; otherwise the answer is NIL. This form acts like an existential quantifier on the variable x. The answer is determined by the expression which is the argument of the function THRETURN.

The internal form of FIND ALL questions is

```
(THFIND(min max result) pattern (varlist)
   (step 1)···(step n))
```

whose meaning is:

find a list of objects, whose length is between min and max, that are the values that cause the program (step 1)···(step n) to succeed.

Suppose we want to know the state of a president. Example: "FIND THE STATE OF PRESIDENT KENNEDY". The input form is translated into the internal form after the parsing phase. The internal form is obtained using the semantic information contained in the word "president". The system knows the internal form of any president record, and so writes the MICROPLANNER statement

```
(THPROG($STATE)
   ($GOAL(KENNEDY $=X $=STATE $=Y)$T)
   (THRETURN $STATE))
```

In the records of presidents the record of President Kennedy is found. Dummy variables are substituted in the goal pattern for all information but president name and president state. The form THRETURN sends back the name obtained through the pattern-matching process.

Suppose we want to modify the data base, by introducing or deleting some information. We may introduce the modifications directly into the internal representation or through the special requests INSERT and DELETE. We can give the command

"DELETE CANDIDATE FOO"

and we obtain the deletion of the record corresponding to the candidate FOO.

We can give the command

"INSERT WIFE ANNY OF PRESIDENT GREEN"

and we obtain the insertion of ANNY in the wife field of president GREEN.

The deduction power of the system can be applied to different kinds of questions. Defining appropriate theorems, we may ask for the most voted-for president of a given party, we may find which states had at least one president, and so on. It is obvious that the dictionary of the QUASI language is adequate in the words and expressions required.

3. CONCLUSIONS

In this paper we have not discussed a complete consultation system, but some ideas for organizing such a system, and we have discussed our choices for the internal and external language. The implementation of the input language has been obtained by writing a parser program which allows a quasinatural input language. A simple syntax is adequate to formulate reasonably rich expressions. The dictionary should be tailored to the practical exigencies. The semantic analysis of the sentences is obtained through the analysis of some key words which allow entry to the data base.

REFERENCES

1. A. M. Andrew, Artificial learning systems and QAS, in *Conference on Artificial Intelligence: Question Answering Systems*, IIASA CP-76-6, Austria, 1975.
2. B. C. Bruce, A model for temporal references and its application in a question answering program, *Artificial Intelligence* 3 No. 1 (1972).
3. A. Colmerauer et al., "Un système de communication homme machine en français", Rapport du group de Recherche en Intelligence Artificielle, UER de Luminy, Université d'Aix-Marseille, France, 1973.
4. D. J. M. Davies, POPLER 1.5 reference manual, TPU Report 1, Univ. of Edinburgh, Edinburgh, Scotland, 1973.
5. D. J. M. Davies, Representing negation in a PLANNER system in *Proceedings of the AISB Summer Conference*, Univ. of Sussex, England, 1974.
6. G. Gini and M. Gini, Cognitive information retrieval by goal-oriented languages, in *Conference on Artificial Intelligence: Question Answering Systems*, IIASA CP-76-6, Austria, 1975.
7. G. Gini and M. Gini, Object description with a manipulator, *The Industrial Robot* 5, No. 1 (Mar. 1978).
8. C. C. Green, The application of theorem proving to question answering systems, Tech. Report CS 138, Memo AIM 96, Stanford Univ., Stanford, Calif., 1969.
9. C. Hewitt, Description and theoretical analysis (using schemata) of PLANNER: a language for proving theorems and manipulating models in a robot, AI Memo 251, MIT, Cambridge, Mass. 1972.

10. MACSYMA, *The* MACSYMA *Reference Manual*, The Mathlab Group, MIT, Cambridge, Mass., Sept. 1974.
11. J. McCarthy and P. J. Hayes, Some philosophical problems from standpoint of artificial intelligence, in *Machine Intelligence* 4, American Elsevier, New York, 1969.
12. D. V. McDermott, Very large PLANNER-type data bases, MIT AI Memo 339, MIT, Cambridge, Mass., Sept. 1975.
13. D. V. McDermott and G. J. Sussman, The CONNIVER reference manual, AI Lab Memo 259a, MIT, Cambridge, Mass., 1972.
14. A. S. Michaels et al., A comparison of the relational and CODASYL approaches to data-base management, *Comput. Surveys* 8, No. 1 (Mar. 1976).
15. N. J. Nilsson, Artificial intelligence, IFIP Congress 74, Stockholm, Sweden, 1974.
16. J. F. Rulifson et al., QA4: a procedural calculus for intuitive reasoning, AI Center, Tech. Note 73, SRI, Menlo Park, Calif., 1972.
17. E. J. Sandewall, Formal methods in the design of question answering systems, *Artificial Intelligence* 2 No. 2 (1971).
18. E. J. Sandewall, Representing natural language information in predicate calculus, in *Machine Intelligence* 6, Edinburgh U.P., Edinburgh, 1971.
19. E. J. Sandewall, Ideas about management of LISP data bases, MIT Memo 332, MIT, Cambridge, Mass., May 1975.
20. E. H. Shortliffe, MYCIN: *Computer-Based Medical Consultations*, American Elsevier, 1976.
21. G. J. Sussman et al., MICROPLANNER reference manual, AI Lab, AI TR 203, MIT, Cambridge, Mass., 1970.
22. T. Winograd, *Understanding Natural Language*, Academic, 1972.

Received July 1979