CONFERENCE ON ARTIFICIAL INTELLIGENCE: QUESTION-ANSWERING SYSTEMS

JUNE 23-25, 1975
CP-76-6

CP-76-6

CONFERENCE ON ARTIFICIAL INTELLIGENCE: QUESTION-ANSWERING SYSTEMS

JUNE 23-25, 1975

International Institute for Applied Systems Analysis
2361 Laxenburg, Austria

FOREWORD

The last decade has produced several profound and exciting results in computer science theory and application. Some of these results have prepared the ground for disciplines now recognized as significant branches of computer based science: the theory of formal grammars and automatic compiler construction, information retrieval and data base management, the theory of communication and computer networks, and problem solving and artificial intelligence are examples of new computer sciences.

In the area of artificial intelligence (AI), theoretical and applied research related to knowledge representation in computers, natural language analysis, deductive inference and automatic learning represent the most interesting topics and promise to become the basis for a new style of computer use. The general idea of this style consists in allowing the user to tell the computer "what to do" instead of "how to do". The computer system in this case behaves as an intelligent adviser and interpreter of predefined rules of the game in any particular problem area. Its advantages over human advisers and interpreters are based on the ability to store and handle gigantic amounts of structured data of which the end user can have only a vague idea. This approach becomes particularly attractive in different areas of applied systems analysis where computer programmed mathematical models give additional analytical power to an "intelligent" computer system.

The challenging and promising features of AI research resulted in the organization by IIASA of an international Conference on Artificial Intelligence and Question-Answering Systems in June 1975. This Conference was held in accordance with the long range research strategy of the Computer Science Project and attracted 27 computer specialists from 12 National Member Organizations. Two basic points were discussed: scientific problems and basic results in the development of question-answering systems with natural language input and inference capability, and possible IIASA efforts in establishing an intelligent question-answering system with a data base for IIASA's applied projects.

This publication contains papers devoted mostly to the first point. The particular subjects that were covered include natural language analysis, knowledge representation and deductive inference mechanisms.

An important practical consequence of the Conference was a proposal from the Conference Working Group to IIASA for the implementation of a question-answering system for data base management at IIASA.

Apart from the obvious scientific results, the meeting also helped to establish contacts between the NMO's involved in AI research. Participants agreed on future cooperation among their institutions in various AI areas.

Several people put considerable effort into the preparation of the Conference and the handling of its results. Bertram Raphael from the Stanford Research Institute initiated the discussion on the importance of AI research for IIASA. Alexander Butrimenko, leader of the IIASA Computer Science Project, and F. Klix from the Academy of Sciences of the GDR were the main initiators of the Conference and contributed greatly to its organization. Ilse Beckey devoted much of her time and energy to arrangements for the Conference; and Yuri Kriukov from the USSR helped in preparing the papers for presentation and publication. Our thanks are also due to the Computer Science secretaries for their faithful help, and to the IIASA editorial staff.

On behalf of the Working Group

V. Briabrin

TABLE OF CONTENTS

Introduction to the Conference

F. Klix

1. Opening Remarks

Dear Colleagues:

Before we go into details, I would like to make some comments on the history and aim of our conference. It was in the autumn of 1971 at the foundation meeting for IIASA when Professor Raiffa, the proposed first director, asked me which project I would like to work on as an experimental and mathematical psychologist. My choice was for selected topics of artificial intelligence. My reasons were the following: AI is an interdisciplinary field within the modern sciences that has more than one applicational value which makes it interesting for IIASA. As an interdisciplinary research field, AI connects computer science, mathematical logic and automata theory, experimental and mathematical psychology, linguistics, and other fields. AI is specific basic research work which grows quickly. The feedback of these developments encourages a greater use of computer power that is now underexploited. In order to obtain the basic support of IIASA, it was necessary to indicate the possible value of AI research within the frame and the policy of that institute. The various aspects of AI research had to be evaluated under these and other conditions.

What should be selected from the different possible approaches that are embraced under the roof of artificial intelligence?

To outline some aspects of these evaluation procedures: first there is the domain of pattern recognition and classification systems. These are well-established application fields, e.g., the identification of type- and handwritten letters, picture classification and scene analysis algorithms, medical diagnosis, etc. Until now, there has been no inner tie to a well-founded project at IIASA.

Second, there is the field of heuristic programming and search techniques, which were regarded for a long time as a key for understanding of problem solving processes. Heuristic programming is now widely applied in CAD (Computer Aided Design), or in fully automated industrial design projects. General search techniques are incorporated and refined in systems for theorem proving and automatic programming, which are supported by newly emerging high-level languages for problem solving. Some of these developments were integrated in big projects, but these projects do not appear to be of special interest to IIASA.

Third, there is the development of formal and especially of programming languages and of methods to describe their structure and semantics. The main application fields are in problem-oriented computer devices, their software organization, the optimization of man-machine systems, etc. These are also fields and tasks that have no sufficient background at IIASA.

Fourth, there are motor-action systems as part of AI research. These are components of robots, especially in interlinkage with pattern recognition, scene identification, and classification devices.

As a fifth point, we could mention fact storage and retrieval systems. Of course, they would be the most interesting part of AI, but there are well-established designs for storing and using data files. It seems to me, however, that as a research task they are nearly out of our range of interest. So, from the scientific aspect, there is reason to put aside such a proposal although the practical value of such an information storage and retrieval system goes without question. This now leads to the final proposal.

Sixth, there is a modern development in AI research which has to do with the representation, manipulation, and use of knowledge. Up to a certain degree, this aspect is representative for AI research as a whole, as Raphael (1974) and Nilsson (1974) have pointed out, and the paradigm of representation, manipulation, and use of knowledge in today's research is the question-answering system (QAS).

The main reason why I directed the attention of some IIASA staff members to QAS is that QAS has both basic and interesting research aspects, and it can extend application fields of computer capabilities. This should also be of interest to nearly all NMO countries. The main point is that although AI research, in general, cannot become a project of its own, special aspects of AI research related to QAS can support substantial projects of IIASA as well as link research activities which are going on in different NMO countries. An international coordination of research activities in this field could enhance the efficiency and lower the expenditure of realizing such a system. These are the main reasons why I have proposed the realization of a QAS at IIASA. I hope that this conference can be the beginning of such a project.

Before I continue and define this practical aspect, let me present some research aspects that will be handled during the conference and that should be the core of the discussion because they are necessarily the crucial points with regard to the extension of the application field in question as mentioned above.

2. QAS and Some Important Research Aspects to be Discussed During the Conference

The general architecture of a QAS is defined by the following conditions and components:

a) Weak standardized, approximately normal, and well-formed sentences in a living language, used as inputs as well as outputs.

b) A parsing procedure (incorporating or interacting with a formally described grammar), which decomposes the input string into a syntax tree or another arrangement of syntactic constituents (substrings).

c) A procedure for semantic interpretation of the syntactic structure. It has to find out the meaning of a sentence which will be represented in the form of a tree or a network (particularly labeled by concept words) that can be stored in the knowledge base.

d) Searching and transformation procedures which allow the transformation of surface properties of the input sentence until they fit a given entry structure.

e) Transformation procedures on the entries within a given structure until they fit a given input (often realized as theorems to be proved with given clauses).

f) Identification procedures which allow detection and deduction of implicitly given information such as frame data, causality directions, and time relations.

g) As far as possible to bring into being the self-extensibility of the system: to organize new data (fitting given arguments or relating new entries to the appropriate conceptual graph structure).

h) The mapping of an item (explicitly stored in, or derivable from the knowledge base) that represents the meaning of the answer into a language construction (at least a kernel sentence; surface transformation should be applied if possible). This last step is necessary to enable the user to communicate with the QAS completely in dialogue mode.

i) Other abilities, such as more sophisticated learning capabilities and decision procedures for the forgetting of facts or relations, have not been essential to such a system until now.

Altogether, we see that each question-answering system can be considered as being composed of three main components, namely:

a) A corpus of knowledge about a more or less extended part of reality as the disposition of the system. It is stored as a structured set of information and usually called the data base of the system. It constitutes the semantic representation of the discourse area, the so called 'universe of discourse'.

b) There is a system of mechanisms or procedures for the organization and linking of that knowledge. It is realized by programs and represents the inferential capacity of the system.

c) Mechanisms for the transformation of language units into concepts and conceptual properties of the stored knowledge. The semantic representation is taken as the basis for the description of the meaning of structural or phrase components of normal language sentences. In this sense, the semantic representation is the core of the QAS because it mediates between the language input and the language output.

Despite the given agreements on the general architecture of a question-answering system, there are very different approaches as to the detail. As these details are very important in view of the practical usefulness of the whole system, they should be discussed during the conference. I would like to go a bit more into detail on this topic and indicate some special approaches within these components.

3. Component: The Internal Representation of Knowledge

First I would like to indicate that there have been real developments within the last ten years. Together with the progress in language analysis and language processing, which began with the handling of words and groups of words and progressed to the handling of complete sentences and eventually to sentence sequences, there has also been real progress in the mode of the conceptual representation of its content or meaning. It goes from the adjoining of properties with words (represented by sequences of symbols) to tree structures. From that point (and I am simplifying the real progress) the research progressed to the representation of surface structures. Today, the central attention has turned to the extraction of deep structures from surface structures. The aim is to map the deep structure of a given normal sentence into a logically unequivocal representation in the form of a semantic net or a set of logical axioms (mostly in the predicate calculus form). The point I would like to stress is that the development obviously is removed from the representation of knowledge by language phrases, and the ability of language understanding by mapping word chunks in a semantic net representation is revealed with new and complicated problems: there are the mutual interrelations between syntactic and semantic aspects, the semantic disambiguation of phrases, and the semantic role of morphological properties complicating the attachment of linguistic units to concepts as units of the net. Forward

and backward procedures have to be implemented for clearing context dependencies, and, if I am informed correctly, these difficulties are in no case completely solved. Different approaches are proposed. In general, they demonstrate partially suitable solutions. Let me indicate some examples.

There are several ideas as to how to encode propositions within the data base. Sandewall indicated that such a representation is sufficient only if logical rules of interpretation (for instance as a set of logical axioms) are added.

When comparing different proposals for knowledge representation it is important to note capabilities and incapabilities for reflecting the various parts and properties of the reality. This now leads to competitive forms of knowledge representation. Besides the dominating semantic network approach there are other approaches: It is possible to represent the knowledge base by a set of discourse specific predicate calculus axioms or to imbed it in data and program structures of higher level languages of AI e.g. PLANNER or QA4.

If it is our aim to plan a QAS for IIASA, we should try during the discussion to clear which mode of representation is preferred. There are no contradictions in each case. Some forms of representation are nearly equivalent or alike in their efficiencies. The pros and cons, however, should be taken into account during the discussion.

Following is an outline considering some aspects for the discussion with regard to these three modes of knowledge representation.

a) The semantic net representation is a graph with nodes and labeled edges or areas. The nodes represent the concepts (individuals) and the edges represent semantic relations. Generally, they are denominated. In the formal sense, such a net can be interpreted as a collection of relational triplets R(x,y), where R is the name of a binary relation, and x and y are two individual constants. Such a structure is more appropriate for representing real objects and time-invariant relationships. Under these conditions, the attachment to the lexical entries is relatively simple. Relationships between concepts can be represented in the same way as has been indicated by Schank and Rieger. There are also difficulties that are due to the limited expressivity power of the relational logic. I am not in a position to evaluate the efficient trials which have been undertaken for enlarging this power, i.e., to use higher order logic calculi to express propositions on propositions.

b) Now let us consider some critical aspects of knowledge representation with the help of logical axioms. This mode of representation can scarcely be separated from

the manipulation of the stored information. It is most suitable for using resolution-oriented theorem provers as deductive components. The data base is organized as axioms, which are given in a skolemized clause form. The basis is the first order predicate-function calculus. The first order predicate calculus is suitable to represent many general propositions but it might be difficult to express intensional constrictions. It seems possible to apply higher level logic calculi within this representation mode. It could be highly important if participants of our conference have experience in this field. Until now, it has not seemed quite clear if deductive procedures, based on the principles of general theorem provers, can be improved so far as to suffice the efficiency requirements of an usable QAS. But methods have been developed which now allow one to economize the storage volume (e.g., by sharing of common substructures of different logical terms) and the same efficiency (by applying various special strategies oriented on syntactical criteria).

c) Now let us consider some aspects of knowledge representation by immediately encoding it in higher-order programing languages. Languages such as MICROPLANNER, PLANNER, QLISP, AND QA4, can be regarded to have some important properties of higher level logical languages. In the programing systems are implemented deductive capabilities as well as mechanisms for elaborating and establishing a specific data base. Abilities of this kind can be used in QAS systems. One of the most interesting points are the procedures of pattern-dependent procedure-activation which can be used for goal-dependent deductive processes. Most of these languages are based on LISP, but they have a more complicated command structure, a greater variability of data types and altogether a more powerful descriptive character than LISP. Such a form of knowledge representation within a successful QAS was elaborated by Winograd (1971) and based on MICROPLANNER. It allows us to describe facts within the discourse area and use heuristics in the form of recommendations for joining data. Procedures of this kind are extremely powerful, but their complexity is very high. The analyses of their behavior may become difficult even for the designer himself.

4. Deductive Processes

Now let us check some aspects of deductive processes. Deductive processes in QAS are determined by sets of propositions --axioms and theorems. They have to be linked in a goal-directed manner. The main problem is to decide which proposition has to be joined. Deductive procedures are not only necessary with regard to questions which have to be answered, but also they are necessary for the understanding of sentences, i.e., for resolving

anaphoric references, for the completion of incomplete statements (the use of presuppositions), and for rejecting statements which are in contradiction with the stored knowledge. More information has to be activated for understanding sentences than is given explicitly in the input strings.

With regard to answering questions, the derivation of supplementary questions is most important. Search procedures in an extended data base are necessary in handling decisive questions.

Within the inferential processes to be conceived for getting new statements for the given ones in the data base, the deductive processes are used to an overwhelming degree. There are also different standpoints with regard to the general appropriateness of resolution-oriented theorem provers. Can they be the deductive vehicle of a QAS? At the moment, the efficiency does not seem to be sufficient. Research work seems necessary in order to learn more about semantically oriented criteria (not just syntactic ones) and heuristic principles for theorem-proving strategies. I am very curious whether a report on this will be presented in the next few days.

With regard to psychological aspects, more and more powerful inferential capabilities seem to exist. Until now, inferences due to analogies, inductive, and abductive forms have not been used. This indicates a research area on common principles in human and artificial inferential abilities.

Up to this point, I have presented some aspects of QAS with regard to different functional or procedural aspects. Properties were especially indicated where different standpoints and positions are given and where a common standpoint should be elaborated with regard to the design of a real system for IIASA.

But there is also another point where different positions will come into being (and with regard to this same requirement). This aspect concerns the design of the system as a whole. Because it is also necessary to decide this question in favor of one system, I would like to sketch the main possible alternatives, and I would like to do this with regard to the literature as it was available to me.

5. On Different Approaches in the Design of a QAS

With the construction ideas of a QAS, today's designers use--in a different degree--experiences of different scientific areas: information processing, logic, linguistics, and psychology, to mention a few. As a consequence, various types of QAS may be differentiated.

a) There is the endeavor to take already realized and checked traditional information retrieval systems as a basis, using relatively homogeneous, structured data files and a query language which is modified in the

direction of normal language utterances. Improved information access is the main purpose of these developments (Kellogs (1968; Woods (1967,1972)).

b) There is another type of system design, tried and developed by Schank et al. (1971), Simmons et al. (1972), Friedman and Woods (1972), and others. They prefer the most efficient procedures of language processing and use completely normal sentences of the English language. Some aspects especially serve to check linguistic models, i.e., with regard to the syntactical and semantic analysis of language comprehension or to the generation of paraphrases. The general purpose is to realize language understanding, but several of the developed procedures are suitable as components of question-answering systems (see Schank). The unpleasnat situation is that these language-oriented models seem to be relatively weak in their inferential power.

c) Within another group of systems, the deductive or problem-solving abilities play the most important role. Here the range of the data base as well as the linguistic variability of the allowed input sentences are small or weak, respectively. Instead, the dominant tendency is to develop, together with heuristic problem-solving programs, very general methods for representing different data structures in a strict invariant manner. Simultaneously, efficient methods are developed which are appropriate for deriving goal-oriented search programs, similar to those which are used in problem-solving strategies. The data base is handled as a problem space. Search algorithms work as goal-oriented heuristic programs. Higher programing languages like PLANNER or QA4 are suitable for realizing such procedures.

d) Another group of systems has been predominantly developed under psychological aspects (Rumelhart and Norman (1973), Quillian and Collins, Anderson and Bower (1973), Newell et al., and others). Special interest is given to the refinement of hypotheses on human long-term memory as well as on the interdependencies of short-term and long-term memory with regard to language understanding. The concept of semantic nets seems to have a powerful heuristic value especially for the understanding of language comprehension. Special classes of psychologically motivated systems are the class of so-called belief systems (Abelson, 1973). The evaluation of concepts and relations plays an important role in these simulation programs.

Although none of these models can be used as complete question-answering systems, I am convinced that these devices possess a great heuristic value for revealing efficient mechanisms of symbol manipulation and the organization of large data

bases. Properties of storage organization, remembering, cognitive learning devices, forgetting principles, as well as procedures of self-organization and extension of knowledge structures can be investigated with the help of these systems.

These are some topics which reveal properties and facilities of information processing systems under special aspects. In general, the main reason for each approach is not to exhaust the possible reachable efficiency of a QAS per se, but to demonstrate specialized procedures or techniques which can be applied or used in QAS. Obviously, the best solution seems to be a compromise among the different designs. I have mentioned these different approaches with regard to the purpose of our conference. Our discussion should also give hints as to the best compromise, i.e., which special procedures or techniques should be applied in a possible IIASA-relevant QAS.

With regard to this question, our alternative seems not to be among these four approaches. Especially in view of the given practical demands, two different approaches are possible, and it might be that they are handled as two steps in our direction. The first one is to develop a universally expandable prototype such as MIND, CONVERSE, REL, etc. It is characterized by procedures with syntactical, morphological-semantic analysis, deductive conjoining, and any semantic net representation and procedures which allow us to generate answers, paraphrases, etc. If it should be decided to pursue this approach, the decision on which discourse area--i.e., which IIASA project is most appropriate with regard to such a system--should be made simultaneously.

The other possible approach is to develop a system within a given project, i.e., with regard to its requirements for lexical entries, relationships between them, and only a small part of inferential power. The variability of input sentences can be strongly restricted. This is of great importance for the complexity of the parsing procedure, for net structure as the frame for the data representation, and for the answer generation device. Systems of this type have been developed by Woods (1972), Badre (1972), Coles (1972), Carbonel et al. (1971), and others. If this type of a system is preferred, the first decision to be made is which IIASA project should be the preferred data base or discourse area, and recommendations as to which approach we take should be made.

6. Some Suggestions for the Workshop

The purpose of this conference has to be seen from different approaches. With regard to the main goal, which is to bring AI research at IIASA into being, it seems necessary to view the contents of the reports, as well as the contents of the discussion, along the following lines:

a) To exchange information and experience gathered with QAS that are successful in practical use. One of the main points of interest should be the demonstration of

different principles with regard to their special efficiencies. Information should be given on computer capabilities necessarily needed to realize a given task.

b) To inform and to exchange ideas on the feasibility of different possible implementation languages. The practicable way seems to be the discussion of selected examples. They should be linked with the representation of semantic properties of the input language. With regard to the programing language, the so-called higher-level languages such as PLANNER, CONNIVER, QLISP, and others should be considered with a view to their special efficiencies.

c) Of theoretical as well as of practical interest is the exchange of information on the complexity problem, by which I mean the relationships between the size of the data base, its structure, the inference modes, and the storage capacity. The information should include techniques of economical storage principles and chunking rules, the incorporation of heuristic principles in the standard mechanisms of higher level language processors.

d) Together with the elaboration of proposals for IIASA, we should work out some proposals for the NMO's on the coordination of research work in AI between groups in different countries and also under the aspect of the establishment of an in-house QAS for IIASA.

The time of our conference is very limited. Until now I have mentioned only a few aspects of the reports and the discussion of some crucial points.

The main purpose I have in mind is to bring a scheme work into being between an IIASA group and research groups in NMO countries. This could be done with the following two subgoals:

a) to begin with a classical fact retrieval system for a special IIASA project as an in-house task;

b) to coordinate the work of different research groups in different NMO countries which are working on QAS with the purpose of extending the efficiency of the fact retrieval system stepwise by inserting inferential capabilities and a natural language understanding and generating part.

DILOS - Dialog System for Information Retrieval, Computation and Logical Inference

V.M. Briabrin and D.A. Pospelov

1. Introduction

There are two main objectives for creating the system described below. First, the processes of developing the system's ideology, its implementation, and its experience for its further utilization are the perfect subjects for computer science and artificial intelligence research, and as such could be proposed as topics for a computer science project at IIASA [7]. Second, the system is oriented toward becoming an instrument for applied research based on different kinds of knowledge representation in computer data base, and, therefore, it has the desirable possibility of being used as a supporting computer system for other IIASA projects [4].

Preliminary discussions have shown that at least two IIASA projects could have immediate profit from promoting and cooperating with the proposed computer system development: these are "urban problems" and "water" projects. Specific project orientation is reflected in the following parts of our system:

a) a "professional" dictionary containing a set of specific terms together with their semantic interpretation;

b) a set of grammatical rules, reflecting specific forms of language or particular phrases by means of which end-users would like to interact with the system;

c) a set of procedures for calculating specific results (usually numerical) from the given arguments; examples of such procedures are machine code subroutines or programs in high-level language for linear programing, matrix manipulation, differential calculus, etc;

d) a description of structure and contents of the data bank which has to keep all the objects (with properties) being relevant to the given problem area;

e) a set of "axioms" and rules of inference to be used for the creation and logical analysis of a semantic model for the specific problem domain.

Switching to another project means the necessity of thorough thinking about the form and contents of knowledge to be fit into the computer system. Actually, this work is a form of systems analysis and hopefully will help applied systems analysis to clear up their own views at the appropriate problem domain.

2. System Configuration and Function

All system functions are performed by a set of procedures which are grouped into four main subsets called "processors" (Figure 1):

a) dialog linguistic processor (DLP),

b) information retrieval processor (IRP),

c) computational processor (CP),

d) logical processor (LP).

Each of these processors manipulates information stored in the data base (DB) which is split into divisions. Each data base division (DBD) has a name and a set of access functions which control all operations, such as object addition, search, and removal. Access functions also provide a hierarchy of access between different DBD's.

Each DBD contains a set of data base objects (DBO's) each of which is characterized by:

a) name,

b) designation of value type,

c) standard value (optional),

d) property list (optional).

Each property in its turn is characterized by an indicator (considered as an extension to the DBO-name) and property value.

The main "users" value types are: character string, bits string, list of numbers (possibly one number), list of pointers to other DBD's or DBO's (possibly one pointer). Besides these types, additional "systems" types show that a given DBO value should be interpreted in a definite manner: for example, one type says that a value of given DBO is actually a DBD descriptor, another type says that it is a procedure body, etc.

In the process of working with the system three basic stages could be outlined:

a) system construction, performed by systems programmers;

b) system specification, performed by systems analysts;

c) system utilization, performed by end-users.

Stage (a) means building up all the necessary procedures to provide for further work by systems analysts and end-users. It is clear that for the benefit of system portability and easiness of amendment and documentation, all the procedures preferably

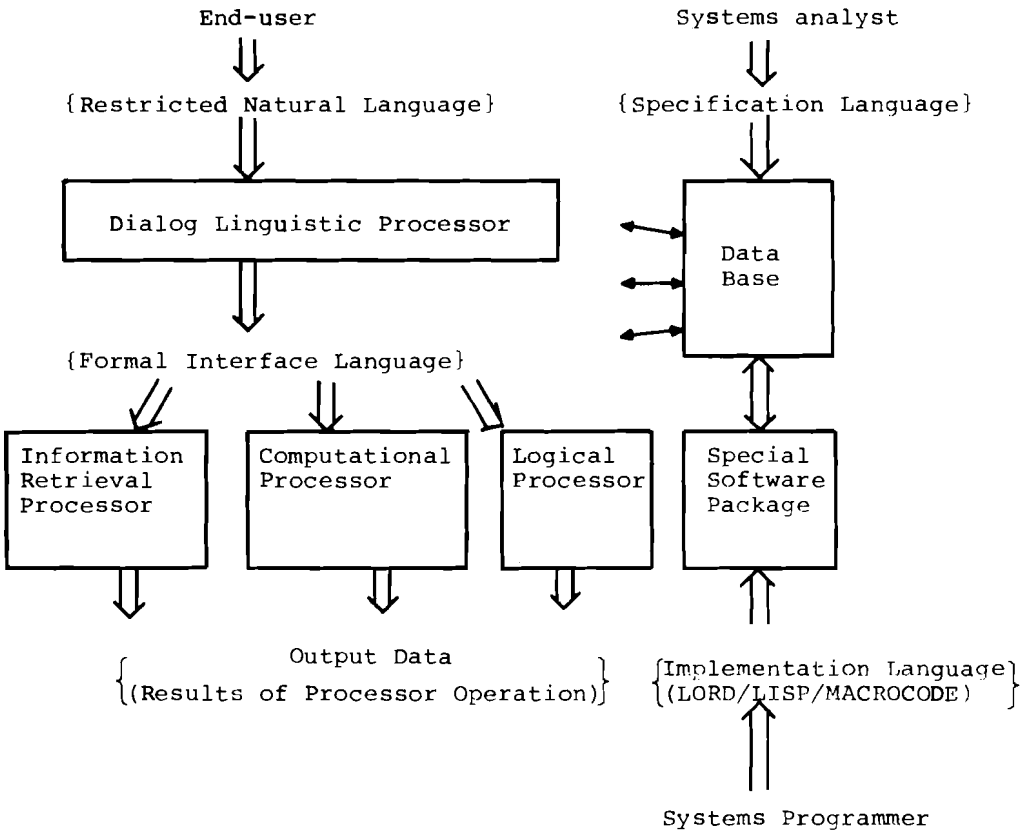


Figure 1. System configuration.

should be written in high-level implementation languages. We choose for this purpose the following combination of programing languages: LISP [3], MACROCODE [5], and LORD [2]. All these languages are available at the present time on the BESM-6 computer and can be transported to the 360-type computers.

Stage (b) creates internal system knowledge about the specific problem domain. It means filling up all the necessary parts of the DB with relevant terms, procedures, axioms, rules of inference, etc. This filling is performed with the aid of special procedures stored in the DB. Formal access language is used at this stage; it could be called "specification language."

Stage (c) implies using the system for applied research. That means running results from the given arguments, searching in the data bank for objects and their properties, answering questions about interrelations among different objects in the semantic model, making logical inference with the purpose of finding the solution and/or planning the sequence of calculation for the given problem. Access to the system at this stage is going to be done in restricted natural language, which is transformed by DLP into the language of "formal interface" (Figure 1) between DLP and other processors.

In the rest of this paper, we discuss the general ideas for implementation of system processors, contents of the appropriate DB divisions, examples of user access language, and corresponding formal interface expressions.

3. Dialog Linguistic Processor

At the stage of utilization, the access to the system is going on through DLP which converts input phrases into expressions of formal interface (ϕ-expressions).

DLP works on input phrase in three stages:

a) Morphological analysis discovers morphological characteristics of the words, searches in the dictionary for their syntactical and possible semantic interpretation, and leaves at the output a sequence of morphemes together with the lists of discovered morphological, syntactical, and semantic attributes.

b) Surface syntactical analysis builds up a syntactic tree with the nodes--morphemes or groups of morphemes and arcs--syntactical relations.

c) Deep syntactical analysis transforms a syntactic tree into ϕ-expression or a sequence of ϕ-expressions which is an output of DLP.

ϕ-expression has a functional format which looks like the following:

```
* f a1,a2,...,an  ;                                    (1)
```

where f is a function name; a1,a2,...,an -- arguments derived from the input phrase.

Each argument could be an atom (name of object or literal), or a structured list in the sense of LISP language, or a chain of the form:

$$[\alpha,\rho,\beta] \quad , \qquad (2)$$

where α and β are atoms, or lists, or syntagmatic chains; ρ-predicate name reflecting the semantic relation between α and β [6].

In some cases for the purpose of clear documentation and easiness of interpretation, arguments are prefixed by key words followed by "=" sign. The sequence of such "key arguments" looks like the following: k1 = a1, k2 = a2,....

A function name is derived from the input phrase or generated by DLP. It shows an action to be performed over the arguments. A list of possible function names is fixed, and each name serves as an indicator directing ϕ-expression to a corresponding processor.

DLP is supported by the contents of several divisions in the DB including:

a) dictionary,

b) set of procedures for morphological and syntactical analysis,

c) grammatical rules controlling all stages of input transformation performed by DLP.

One essential point about DLP is that it can interact with the user by means of auxiliary questions in order to get full "understanding" of the input phrase.

4. Information Retrieval Processor

One of the most frequently needed possibilities provided by a computer system to the end-users is an access to the large data bank containing different sorts of statistical information, reference lists, and other types of encyclopedic data. The following functions should be provided by IRP:

a) Put a new object (with properties) into an appropriate division of the data bank.

b) Find an object by its name (and possible by a descriptor) and get its standard value or the value (s) of its specified property (ies).

c) Delete an object from the DBD.

d) Change standard value or property value of the given object.

e) Perform special operation (e.g., union, intersection, exclusion) over the object standard or property values.

IRP could produce as an output:

a) an object standard or property value (s);

b) a list of references to the objects satisfying the given search criteria;

c) SUCCESS or FAIL signals indicating whether the search was successful or not.

4.1 Examples

a) The question, "What was the Moscow population in 1945?" could be transformed by DLP into ϕ-expression:

GET MOSCOW POPULATION. 1945.

Function GET here has two arguments: the name of division (MOSCOW), and the name of object (POPULATION) extended by the property indicator (1945). Corresponding IRP procedure searches in the given division for the object, extracts its property value and types it out.

b) The phrase, "Give me the numbers of all flights and trains connecting Moscow and Leningrad," could be transformed by DLP into:

GET (FLIGHTS (FROM-MOSCOW & TO-LENINGRAD)

U(TRAINS (FROM-MOSCOW & TO-LENINGRAD)) .

This expression implies that IRP searches in the FLIGHTS and TRAINS divisions for the standard values of FROM-MOSCOW and TO-LENINGRAD OBJECTS, performs two intersections and union of their results as indicated by brackets, and types out the final list of flights and trains connecting Moscow and Leningrad.

Each object in the DB has a unique pointer which can be used instead of object name where necessary (in the property lists, in the semantic model, etc.). Special procedures handle object names and/or pointers providing access to object standard value or property values.

5. Computational Processor

At the present time, most applied systems analysis research is based on a series of calculations performed by programs written in high-level algorithmic languages. Every such program could be considered as a procedure which takes some input data (arguments) and produces output data (results). One procedure's results could become another procedure's arguments or could be printed out as a final data requested by the end-user.

This philosophy constitutes a basis for CP operation. Its task is to interpret procedure calls with the necessary substitution of arguments and to handle the results of calculation.

Each applied program is stored in the DB and accompanied by special object--"applied program module descriptor" (APMD). This object contains the following properties:

a) program name (coinciding with APMD name);

b) type of calculation (the name of programming system);

c) list of arguments (possible with their types);

d) list of results (possibly with their types);

e) location of input area;

f) location of output area;

g) DB pointer to APM body (the body could be stored in symbolic or machine code representation).

Properties (a - f) are provided by experts during the definition of APM and loading it into the DB; (g) is generated by the system.

CP operation starts when DLP produces a ϕ-expression of the form:

$$\text{CALL} \quad z \quad \text{ARG} = (x1,x2,\ldots,xm) \quad \text{RES} = (y1,y2,\ldots,yn) \quad , \qquad (3)$$

where z-program name; x1,x2,...,xm--objects which are going to be substituted instead of arguments; y1,y2,...,yn--objects which are going to receive new values after performing calculation and getting the results.

CP picks up all argument values (with the necessary type conversions) and collects them in the input area. Then CP loads APM body as it is required by the programing system and makes a call for appropriate translator.

Translator handles APM body together with data from input area, and this computational process is supposed to produce required results in the output area.

The final stage of CP operation constitutes in disjoining contents of output area into separate pieces and assigning them as new values to the objects y1,y2,...,yn announced in (c). Thus APM operation could be considered as a process which converts a set of input object values x1,x2,...,xm into a set of output object values y1,y2,...,yn.

CP could discover that in order to get the value of some object x_i, it is necessary to suspend the current calculation and make a call for another procedure providing the required x_i value. This type of operation is performed by means of CP stack mechanism.

6. Logical Processor

In many cases, end-user's inquires will imply direct IRP or CP operation based on preprogramed knowledge about the problem domain. On the other hand, it is likely that some inquires will require a preliminary stage of system operation--looking for the possibility of getting the solution and generating a plan for obtaining the necessary results. This part of system operation is performed by LP.

LP is responsible for construction, amendment, and analysis of semantic model which is represented in computer memory in a form of oriented graph. Each pair of nodes in this graph connected by an arc (ρ) represents a syntagmatic chain corresponding to analytical expression (b). Semantic interpretation of such an expression depends on the meaning of ρ. Examples of semantic interpretation are:

a) "α is the name of β";

b) "α implies β";

c) "α is part of β";

d) "α has property β", etc.

Semantic models contain axioms about the problem domain as well as rules of inference giving the possibility of deducting new facts out of existing axioms and temporary results.

Resolution principles or STRIPS implementation could be good examples of LP procedures.

Besides the goal of finding the solution or plan generation for the specific task, LP cooperates with DLP in providing question-answering facilities which are based on pattern search and logical analysis of the semantic memory contents.

LP is implemented as a set of LISP [3] and LORD [2] procedures with a heavy accent on pattern search and pattern-driven procedure invocation technique which is becoming popular in the recent developments of AI programing systems [1].

7. Conclusion

The proposed system will be capable of providing the "intelligent" computer service for three main kinds of end-user inquires: information retrieval and data bank management, computation of specific results from the given arguments, semantic model creation and analysis with the purpose of problem solving or question answering. There are some theoretical and technical difficulties in developing the system. Prototype implementation and application to specific problem domain will give the necessary experience for further system development and utilization.

References

[1] Bobrow, D.G. and Raphael, B. "New Programming Languages for Artificial Intelligence Research." ACM Computing Surveys, 6, 3 (1974).

[2] Briabrin, V.M., Serebrjakov, V.A. and Jufa, V.M. "Programing System for Artificial Intelligence Problems." VII Symp. on Cybern. Tbilisi, U.S.S.R., 1974.

[3] Jufa, V.M. "LISP-BESM-6 Programming System Development." Symbol Inf. Proc. Comp. Cent., Acad. of Sci., Moscow, U.S.S.R., 1973.

[4] Klix, F. "Problems and Preparatory Steps in Realization of a Question-Answering System at IIASA." Private communication, November 1974.

[5] Mikchelev, V.M. and Shtarkman, V.S. "MACROCODE: Language Description." IAM, Acad. of Sci., Moscow, U.S.S.R., 1972.

[6] Pospelov, D.A. Big Systems (Situation Control). Moscow, Znanie, 1975.

[7] Raphael, B. "Artificial Intelligence: Possible Activities for IIASA." Private communication, June 1974.

Some Comments on Efficient Question-Answering Systems

Hiroji Nishino

1. Introduction

The research and development of a QAS using natural language is one of the most active and hopeful fields in artificial intelligence (AI) research. The review paper on AI given by N.J. Nilson at 1974s IFIP conference gives us a well-sketched overview of the research on natural language understanding [7]. The first peak of the research in 1970s decade was SHRDLU developed by T. Winograd [10]. Inspired by his success, many researchers have been hoping that a QAS for practical uses will be realized in the near future. In connection with Figure 1, Figure 5 shows an enlarged recent history of QA research since T. Winograd.

I would like, however, to mention some practical approaches, different from AI approaches, with respect to the following standpoints. The basic features of the systems proposed for practical uses are as follows:

a) They make use of existing large data bases, created mostly by conventional methods.

b) The use of a natural language is just a convenience for nonprogrammers as end-users. The direct objective is not a sophisticated understanding of natural language.

2. Development Strategy

Although they are closely related to each other, the problems concerning a QAS are roughly divided into the following three areas shown in Figure 1:

a) sentence analysis and generation (linguistic studies),

b) inference and decision (discourse analysis and psychological studies of cognition),

c) data-base management (storage and retrieval of knowledge).

The first area, in which linguists are mainly concerned, has a long traditional history. A linguist tends to make a mammoth syntax analyzer in order to expect the completeness based on linguistic viewpoints. To my thinking, such analyzers seem to be too sophisticated for our purpose and furthermore cannot handle the full complexity of natural language. More compact and efficient analyzers are desirable.

Recent works on a natural language understanding in AI research are mostly concentrated in the second area. There is a broad spectrum ranging from the simplest inference to the most sophisticated one. The recently active studies on psychological cognition are at an extreme end. I feel, however, that these studies are still in the pure research stage.

The third area is also contained in today's computer technologies and is a hot topic at that. In most cases of QAS's in AI research, the data base is small. Conversely, the data base required in practical uses is usually large. Notice that an appropriate method for a small system is, in general, not necessarily efficient for a large one. However, the method of effectively storing and retrieving the knowledge is common in both cases.

From these above-described reasons, a loosely coupled system is recommended. The basic functions of each subsystem and the interfaces among them are briefly illustrated in Figure 2.

It may be clever to adopt two different approaches at the same time in order to promote its development. One approach is, of course, the AI approach. The other one is a data-base oriented one, and the emphasis is placed on managing a real data base. This paper is mainly concerned with the latter approach.

3. Design Criteria

In practical computer technologies, several kinds of query languages for data bases have been proposed and implemented. They contain:

a) a series of procedural operators, or

b) the representation of predicate calculus, or

c) a restricted subset of a natural language.

It is particularly noticeable that certain experimental systems belonging to the third category are quite recently increasing [2,5,6,9]. In this case, the general scheme of the QAS is shown as something like Figure 3. Figure 3 shows a lossely-coupled system with two subsystems. The upper part is a preprocessor for a natural language understanding followed by the lower part, a conventional data-base system. In order to keep the close interface between them, the query language is to be open-ended.

Although the general scheme of a QAS seems to be fairly clear, it is extremely difficult to specify the external specification of the dialog between the machine and the user at the design stage.

The disadvantage of a restricted natural language and a natural languagelike query language is the fact that the casual user forgets the restriction because of the similarity to a natural language.

From the practical viewpoint, it is, however, sufficient enough to make the user only an "illusion" interacting with the machine in a natural way. In this case, the output responses from the machine may be employed for clarifying the meaning of the user's questions, i.e., (a) the ambiguity of the meaning, and (b) the meaning of words not found in the dictionary, and so on.

For example:

a) "Which do you mean by X, X1 or X2?"

b) "Sorry, I don't know the word Y. Please tell me in other words."

Furthermore, the choice of the decision by inference may be helped by the user as follows:

"Can I assume that you mean Z?"

Such clarifying responses will be effective to simplify the upper part of Figure 3 and particularly the dictionary. The detailed mechanism of this part, which is omitted here, will be enhanced step by step by adopting the results of AI research.

How to represent knowledge--i.e., formalism of knowledge--is one of the most important topics, not only in a natural language understanding, but also in other fields of AI research. Most representations have a list structure or semantic network structure of variations of these. In some cases, they have the forms of primitive assertions or predicate calculus. Newly proposed concepts, such as frame or schemata, will give us more flexible structures for the representation of knowledge.

In computer technologies, there are similar circumstances. There were several data-base management systems (DBMS) reviewed in the CODASYL report published in 1971 [3]. Now only a few of them still remain, and the others are already obsolete. Therefore, at first, we must select a flexible data-base model suited for our purposes.

Recently, several new models of the data base, such as a relational-data model [4], a data-semantics model [1], an entity-data model [8], etc., have been proposed, and some of them have been implemented. Although these models are formalized comparable to the ones in AI research, they seem to be flexible enough to satisfy the necessary conditions for our purposes. Particularly the retrieval efficiency for a large amount of knowledge is seriously considered.

4. Implementation Method

LISP is frequently used in the research on natural language understanding. Therefore, LISP's users (rigorously speaking, PDP-10 users) are supported by many useful programs concerning natural language understanding.

Higher-level languages such as PLANNER, CONNIVER, etc., which are based on LISP and developed for inference, are too inefficient in practical applications. The execution time of PLANNER is approximately ten times slower than that of LISP interpreter, and CONNIVER is nearly ten times slower than PLANNER.

A dedicated, high-level language machine which executes the language directly is thus desirable. Recently, some proposals have been reported.

In addition, a dedicated data-base machine is also recommended. The total system which operates as a function-distributed system is illustrated in Figure 4.

Acknowledgment

I am very grateful for the helpful suggestions of H. Tanaka and S. Uemura of our institute.

References

[1] Abrial, J.R. "Data Semantics." Proc. IFIP Working Conf. on Data Base Management System. Cargése, 1974.

[2] Chamberlin, D.D. et al. "SEQUEL: A Structured English Query Language." Proc. ACM SIGFIDET Workshop. Ann Arbor, Michigan, 1974.

[3] COSASYL System Committee Data Base Task Group Report. Feature Analysis of Generalized Data Base Management Systems. Comm. ACM, 1971.

[4] Codd, E.F. "A Relational Model of Data for Large Shared Data Banks." Comm. ACM, 13 (1970).

[5] Codd, E.F. "Seven Steps to RENDEZVOUS with the Casual User." Proc. IFIP Working Conf. on Data Base Management System. Cargése, 1974.

[6] Kellogg, C.H. et al. "The CONVERSE Natural Language Data Management System: Current Status and Plans." Proc. ACM Symposium on Information Storage and Retrieval. University of Maryland, 1971.

[7] Nilson, N.J. "Artificial Intelligence." Proc. IFIP 1974. Stockholm.

[8] Senko, M. et al. "Data Structures and Accessing in Data-Base Systems-Data Representation and DIAM." IBM System Journal, 12 (1973).

[9] Thompson, F.P. et al. "REL; A Rapidly Extensible Language System." Proc. 24th ACM National Conf. New York, 1969.

[10] Winograd, T. "Procedures as Representation for Data in a Computer Program for Understanding Natural Language." Tech. Rep. AI TR-17. Massachusetts Institute of Technology, 1971.

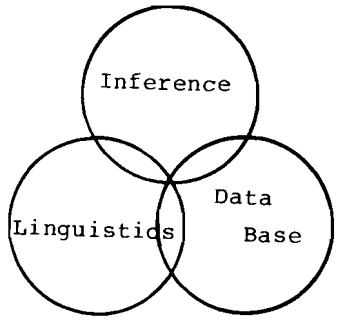


Figure 1. Three problem areas.

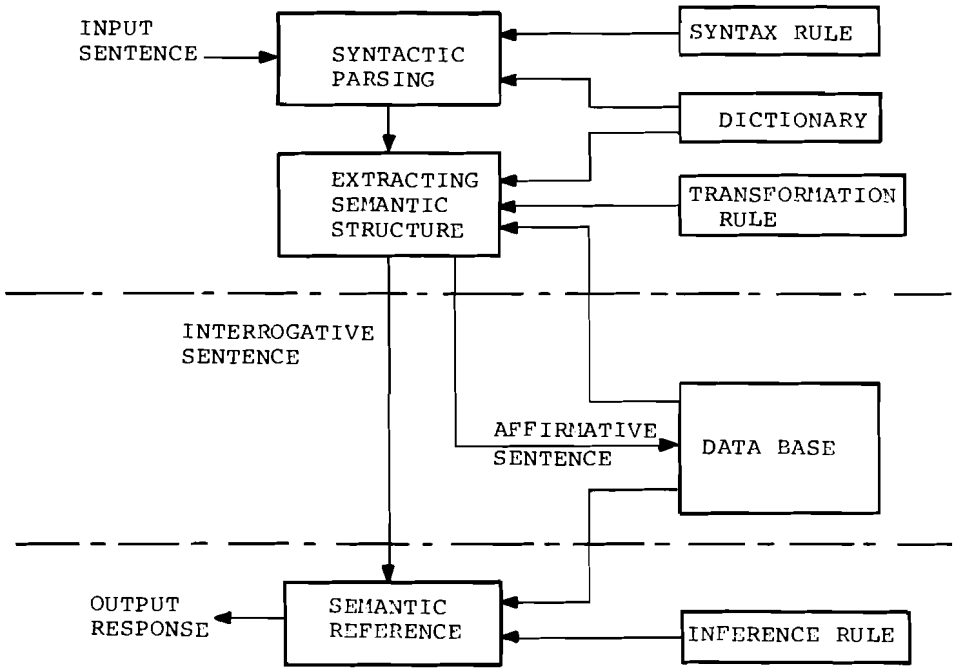


Figure 2. Functional diagram of a QAS.

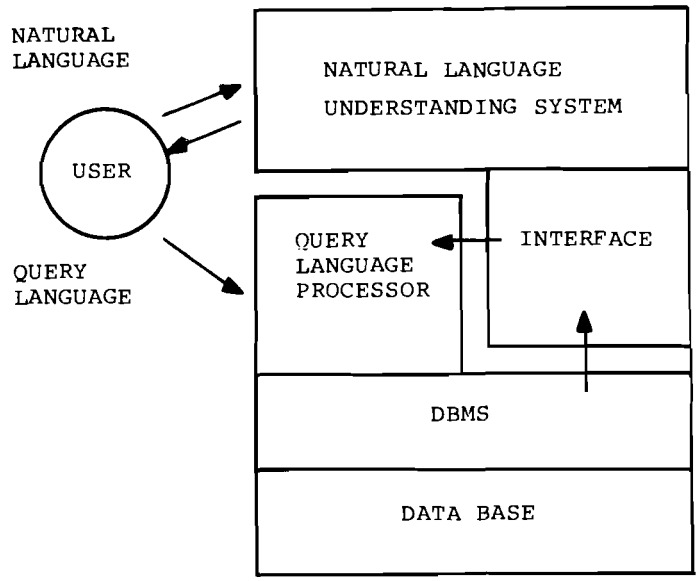


Figure 3. General scheme of a QAS.

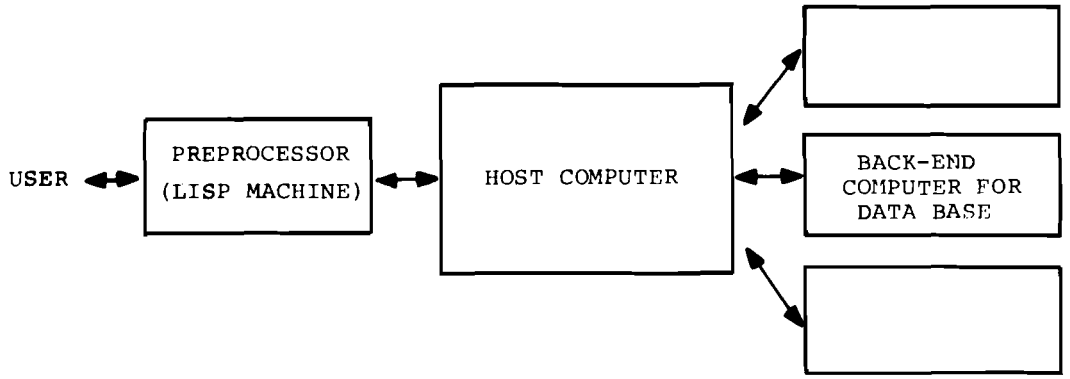


Figure 4. Function-distributed QAS.

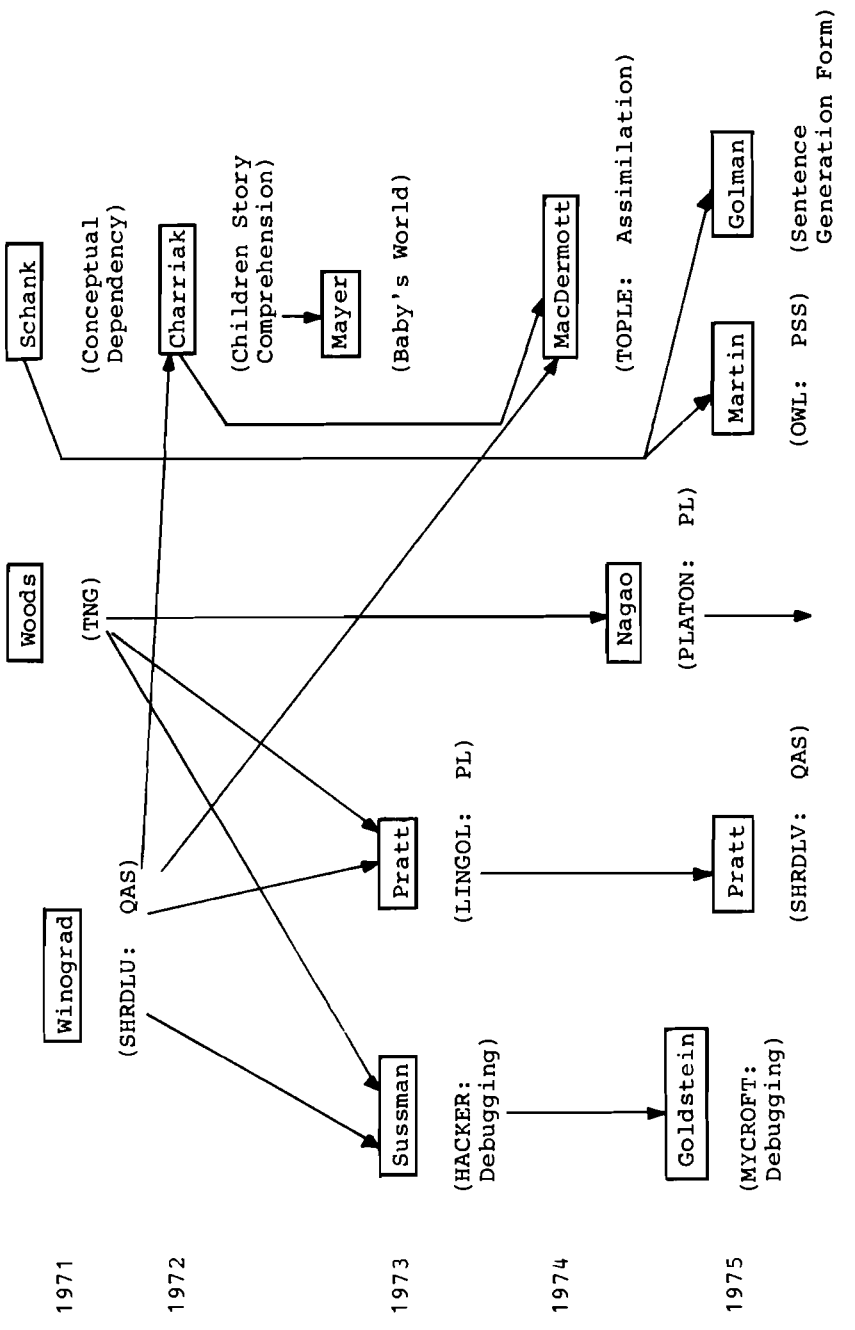


Figure 5. Recent history of QAS.

Partitioned Semantic Networks for Question-Answering Systems

Gary G. Hendrix

1. Basic Network Notions

In its simplest form, a semantic network is a set of nodes interconnected by an accompanying set of arcs. A node may be used to represent an object--where an object may be virtually anything--including physical objects, relationships, sets, events, rules, and utterances. Arcs are used to represent certain "primitive" omnichronic (i.e., time invariant) relationships. (Such relationships may also be represented as nodes.)

A feeling for how nodes and arcs are organized to represent various facts may be gained by considering the network of Figure 1. In this network, the node 'PHYSICAL.OBJECTS'[1] represents the set PHYSICAL.OBJECTS, the set of all physical objects. Likewise, node 'MACHINE.PARTS' represents the set of all machine parts. The arc labeled "s" from 'MACHINE.PARTS' to 'PHYSICAL.OBJECTS' indicates that MACHINE.PARTS is a subset of PHYSICAL.OBJECTS. Similarly, the network indicates that BOLTS is a subset of MACHINE.PARTS and that B, an element of BOLTS, is a particular bolt. Following the hierarchy of another family, X is a particular box, an element of BOXES, a subset of CONTAINERS, a subset of PHYSICAL.OBJECTS.

Node 'C' encodes a containing situation, an element of the situations set ⟨sit-contain⟩, a subset of SITUATIONS, the set of all situations. In particular, 'C' represents the containing of bolt B by box X from time T1 until time T2. The various component parts of situation C are associated with it through special deep-case relationships. For example, in the network there is an arc labeled "content" from 'C' to 'B.' This arc indicates that B is the #@content of situation C, where the notation "#@content of C" means "the value (#) of the content attribute (@) of C." Similarly, X is the #@container of C while T1 and T2 are the #@start-time and #@end-time, respectively.

As a general principle, arcs encode only element, subset, and case relationships. (Under one interpretation, element and subset relationships may be viewed as deep cases also.) Arcs are never allowed to encode relationships such as ownership, which are time bounded.

---

[1] Single quotes are used to delimit nodes.

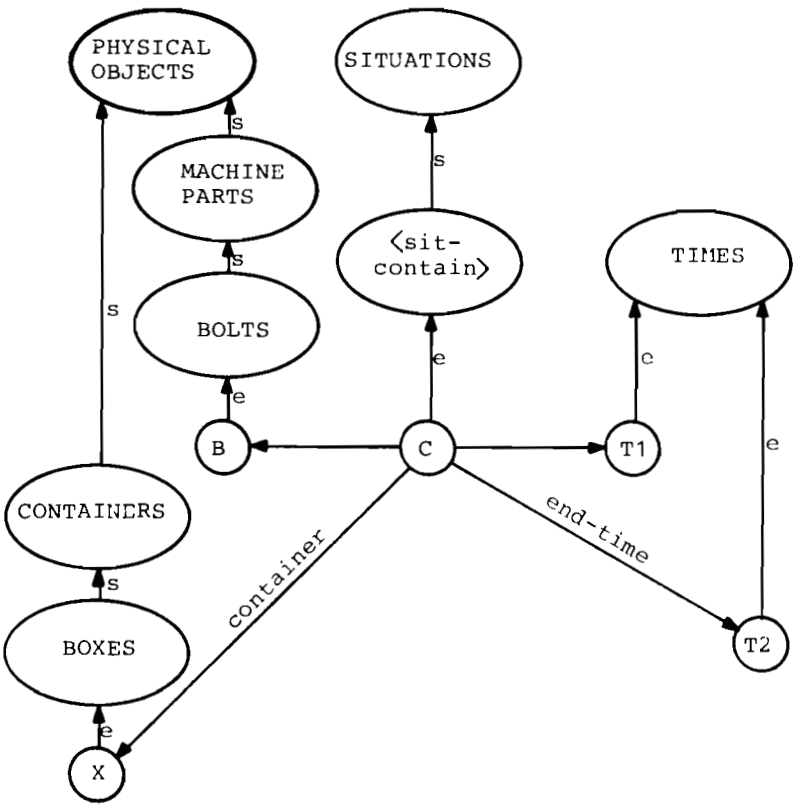


Figure 1. A typical net fragment.

2. Net Partitioning

The central idea of net partitioning is to separate the various nodes and arcs of a network into units called spaces. Every node and every arc of the overall network is assigned to exactly one space with all nodes and arcs that lie in the same space being distinguishable from those of other spaces. While nodes and arcs of different spaces may be linked, the linkage must pass through certain boundaries that separate one net space from another.

Net spaces are typically used to delimit the scopes of quantified variables and to distinguish alternative hypotheses (during parsing and planning). However, before taking up such practical applications, consider the simpler (if atypical) network partitioning exhibited in Figure 2. As shown, each space of the partitioning is enclosed within a dotted line. For example, space S1 is at the top of the figure and includes nodes 'PHYSICAL. OBJECTS,' 'BOLTS,' '⟨sit-contain⟩' and 'BOXES.' S1 also includes the two s arcs indicating that the set of BOLTS and the set of BOXES are subsets of the set 'PHYSICAL.OBJECTS.' In our diagrammatic representations of semantic nets, an arc belongs to a space if the arc's label is written within the dotted-line boundaries of the space. Thus the e arc from 'B' to 'BOLTS' lies in space S2.

The various spaces of a partition are organized into a partial ordering, such as that shown in Figure 3. In viewing the semantic network from some point S in this ordering, only those nodes and arcs are visible that lie in S or in a space above S in the ordering. Thus, for example, from space S2 of Figures 2 and 3, only those nodes and arcs lying in S2 or S1 are visible. In particular, it is possible to see that B is a BOLT and that BOLTS are PHYSICAL.OBJECTS, but it is not possible to see that X is a BOX. From space S5, information in spaces S5, S3, S2, and S1 is visible. Hence, from S5, the whole of the semantic network of Figure 2 may be seen. (For certain applications, the net may be inspected one space at a time. For example, it is possible to query the net in such a way that only nodes and arcs lying in space S2 are visible even though information in S1 is normally visible whenever S2 is inspected.)

In practice, partitioned networks are constructed by creating empty net spaces, adding them to the partition ordering, and then creating nodes and arcs within each newly created space. The use of partitioning in the encoding of quantified statements and categories is the subject of the next two sections.

3. Quantified Statements

In addition to an ability to encode specific facts (such as the containing event encoded in Figure 1), a semantic system needs some facility for grouping sets of similar facts into units, allowing these facts to be represented collectively through some sharing mechanism, and to be conceptualized as an integrated

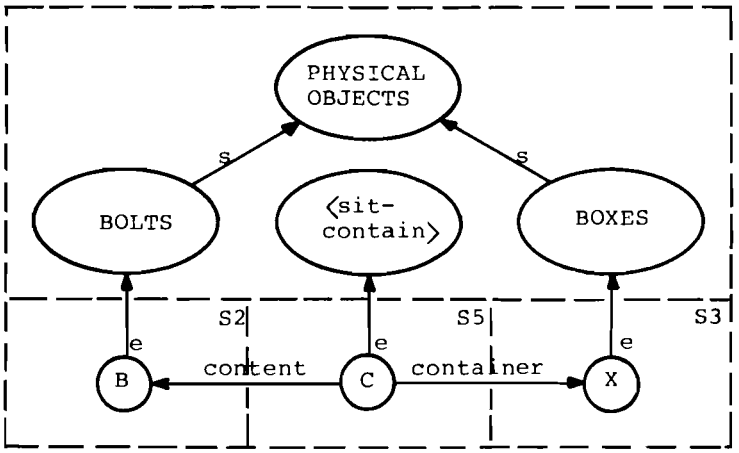


Figure 2. A sample net-space partition.

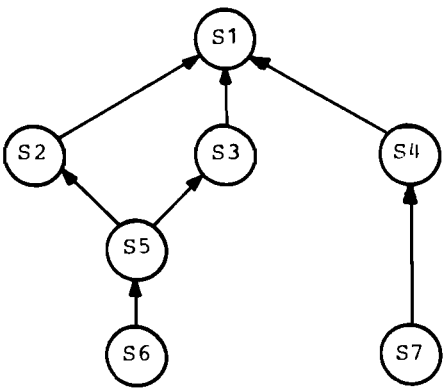


Figure 3. A net-space partial ordering.

whole. An ability to encode generalized information (in the form of quantified expressions) is of considerable importance since it is often impractical (or even impossible) to record the same information by a collection of individual specific statements, both because of the very number (possibly infinite) of statements required and because details of particular individuals may not be explicitly known. Further, since quantification is a component of language, an ability to encode quantifiers is vital to the understanding of certain classes of English sentences, e.g., "Are all subs in the Russian Fleet nuclear powered?" "Do some U.S. boats have more than five torpedo tubes?"

As an example of how quantification is handled in partitioned networks, consider the network of Figure 4 that encodes the statement, "Every bolt in the box is 3/4 inch long and has a nut screwed onto it." In this network, the node 'GS' represents the set of all general statements (the set of statements involving universal quantifiers or, under another interpretation, the set of recurring patterns of events.) The node 'g' represents the particular statement (set of events) cited above.

Characteristically, a general statement encodes a collection of separate circumstances, all of which follow the same basic pattern. This basic pattern is represented by the #@form of the general statement. The #@form of g is encoded by a net space, S4, that lies just below S1 in the partition ordering. (When one net space is pictured inside another, the inner space is below the outer in the partition ordering.) This net space may be thought of as a supernode containing its own internal structure and representing a composite variable that takes on a different value for each of the instantiations of the recurring pattern. Each node and arc within the space of the supernode may be thought of as a subvariable.

General statements are also typically associated with one or more universally quantified variables that are allowed to assume values from some specified range. Statement g, for example, has a universally quantified variable b given by its @Vv attribute. Note that variable b is necessarily a part of the #@form of g, i.e., 'b' lies in space S4. From node 'b' there is an e arc to the set the.bolts.in.the.box, indicating that the value of b (written #b) must be taken from the range set the.bolts.in.box. (The node 'the.bolts.in.the.box' has been created especially to help encode the general statement. Its meaning may be inferred subsequently when the.bolts.in.box.X is defined by the network Figure 6.)

The interpretation of a general statement is that for each assignment of the variables #@Vv to values in their corresponding ranges, there exist entities matching the structure of the #@form. For g this means that for every #b an element of the.bolts.in.the.box there exist,

#h C <has.length>

#s C <sit-screwed:simplistic>

#n C NUTS ,

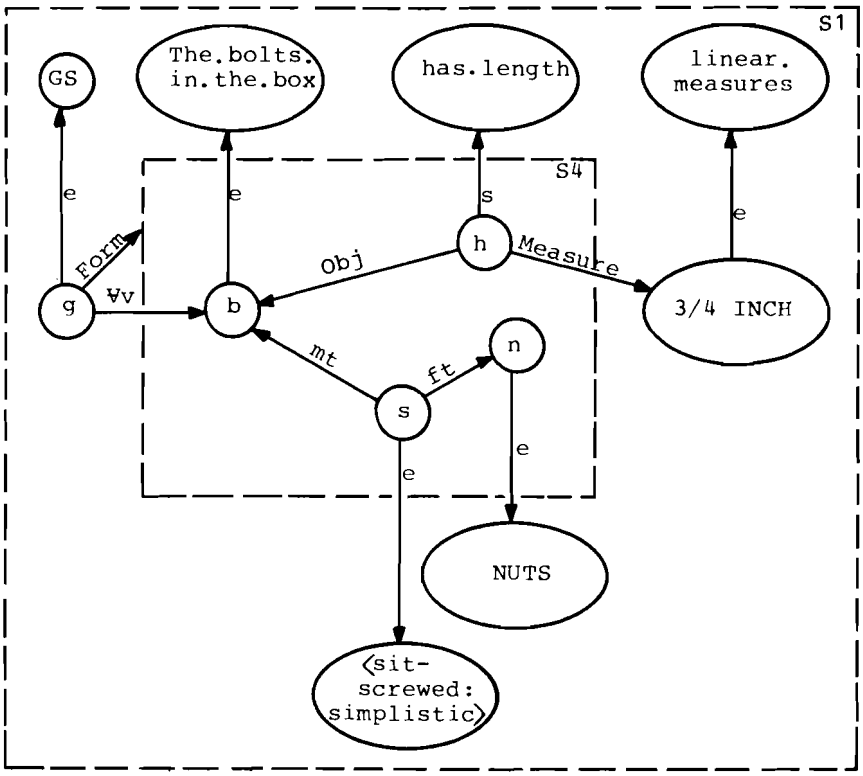


Figure 4. Every bolt in the box is 3/4-inch long and has a nut screwed onto it.

and the relations,

#b is the #@obj of #h

3/4 INCH is the #@measure of #h

#b is the #@mt of #s (i.e., #b is the male-threaded part of #s)

and #n is the #@ft of #s.

Thus, the interpretation of g is that for every #b in the.bolts. in.the.box, there exists a situation #h in which the length of OBJect #b is the MEASURE 3/4 inch. Since '3/4 INCH' lies outside space S4, there is only one measure for all the bolts in the box. Further, for each bolt #b there is a nut #n (depending on the individual #b) which is in a situation of being screwed onto #b. (A screwed:simplistic connection may exist only between two threaded objects, one with male threads [the #@mt] the other with female threads [the #@ft]. A screwed:simplistic connection may be contrasted with screwed:standard connections in which multiple unthreaded parts are held together by a bolt [or other threaded object] that passes through the unthreaded objects to engage a nut.)

Complex quantifications involving nested scopes may also be encoded by net spaces, as shown abstractly in Figure 5.

4. Rules and Categories

A convenient method for organizing information in a semantic system is to divide the various objects (including physical objects, situation, and event objects) in the semantic domain into a number of categories. Using categories, objects that are somewhat alike become grouped together, allowing similar objects to be thought about and talked about collectively. The scheme is hierarchical, in that some categories may be subcategories of more general classes. The lower a class is in the category hierarchy, the more alike its members must be. The "likeness" arises in that members of each category possess certain common, characterizing properties (such as an association, with common attributes or deep conceptual cases.)

The categorical system serves the important purposes of spotlighting similarities among objects and compressing redundant information by recording common information at the category level rather than with the individual. If an object Z is known to belong to some category K, then Z is known to possess the common properties of K's members and the common properties of the members of each of K's supercategories. This ability to encode information at the category level rather than with each individual is of practical importance because it saves computer memory and because

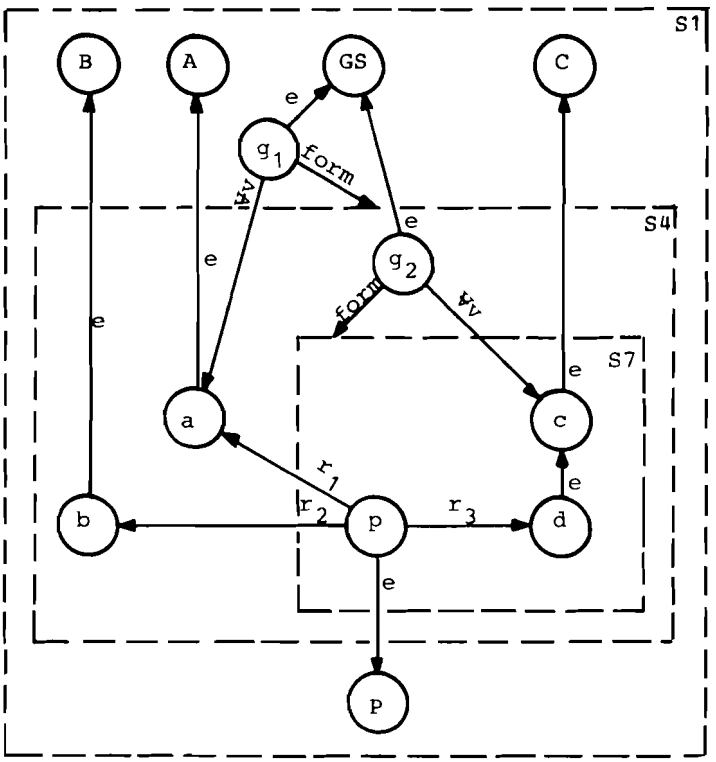


Figure 5. $(\forall a \in A)(\exists b \in B)(\forall c \in C)(\exists d \in c)[P(a,b,d)]$.

all the elements of a category (perhaps being infinite in number) may not be explicitly known.

For natural language processing, the category system has the important feature that members of the more significant categories (the categories commonly held in the minds of humans) are expressed by the same set of linguistic patterns. As an elementary example, screwdrivers, wrenches, hammers, and saws belong to a category of objects that may be expressed by noun phrases headed by the noun "tool." Various attaching events may be expressed by complete sentences using the words "attach," "mount," or "fasten" as their central verbs.

Central to the notion of a category is the notion of a rule that specifies a necessary and sufficient test for category membership. Necessary rules, which all category members must obey, and sufficient rules, which can prove that an object belongs to a given category, are also of importance.

As a simple example of a category and its defining rule, consider the category of bolts in box X. This category is represented by node 'the.bolts.in.box.X' of Figure 6 with the associated rule being encoded by net space S2. The ens arc lying in space S2 from node 'b' to 'the.bolts.in.box.X' indicates that 'b' represents what may be thought of as an archetypal element of the category. (Symbol "ens" means archetypal "element, necessary and sufficient.") Any objects with the characteristics of b belong to the category, and all members of the category have the characteristics of b. As encoded in space S2, the characteristics of b include membership in BOLTS (the set of all bolts) and involvement as the #@content in a containing situation in which box X is the #@container.

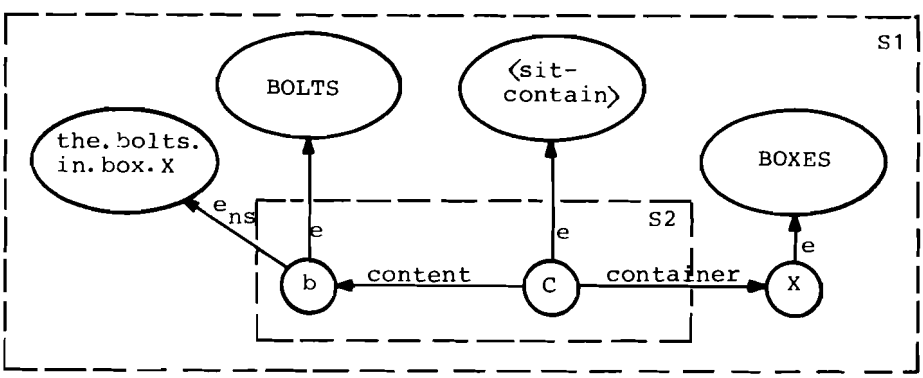


Figure 6. The necessary and sufficient rule defining "the bolts in box x".

In natural language processing, particularly during the parsing phase when surface structures are being translated into nets and when the semantic well formedness of sentences and sentence fragments is being tested, it is important to know what attributes (deep cases) are associated with certain categories of objects (especially with event, situation, and other verblike categories) and what range of values each attribute may assume. This information is of utility because attributes indicate the types of participants that are involved in particular categories of situations and because there is often a direct mapping from syntactic cases (including prepositional phrases) to these attributes. Knowing the correspondences between surface cases and attributes, and knowing the ranges of values for each attribute allows some parses to be rejected on macrosemantic grounds and provides a facility for predicting the citing of certain situation participants in the surface language. (This prediction ability is especially important for speech understanding.)

The attribute-range information for a category, collectively referred to as the category's delineation, may be associated with the category through a delineation rule. A delineation rule is a necessary rule that includes range information about every attribute of the delineated category.

As an example of a delineation rule, consider the delineation of category ⟨to-bolt⟩, the category of events in which two machine parts are attached by using bolts as fasteners. Delineation information for this category is encoded by the network of Figure 7. In this network, node '⟨to-bolt⟩' is linked to a node 'b' by an ed arc which indicates that b is the delineating "element" of ⟨to-bolt⟩. Encoded within space S4 is attribute-range information concerning each of the six attributes possessed by members of ⟨to-bolt⟩. In particular, the rule encoded by space S4 indicates that a bolting event involves an #@actor taken from the set of INTELLIGENT.ANTIMATE.OBJECTS, a #@minor-p and a #@major-p taken from the set of MACHINE.PARTS, a set of #@fasteners taken from the set of BOLT/NUTS (a bolt/nut is a bolt and a nut that work together to form a single fastener), a #@tool taken from the set of TOOLS (which includes hands and fingers), and a #@time taken from the set of TIME.INTERVALS.

Given the two sentences, "I bolted the pump to the base plate WITH 1 INCH BOLTS," "I bolted the pump to the base plate WITH THE WRENCH," the delineation of ⟨to-bolt⟩ may be used to determine that the WITH phrase in the first sentence supplies the #@fasteners case, while in the second sentence it supplies the #@tools case.

The delineation rule of Figure 7 shows all delineation information concerning ⟨to-bolt⟩ to be encoded in a single rule linked directly to the category. In practice, categorical information is almost always distributed among many points in the categorical hierarchy. To see how information is distributed at various levels, consider the hierarchy of ⟨to-attach⟩ events that is exhibited in Figure 8. The most general category in the

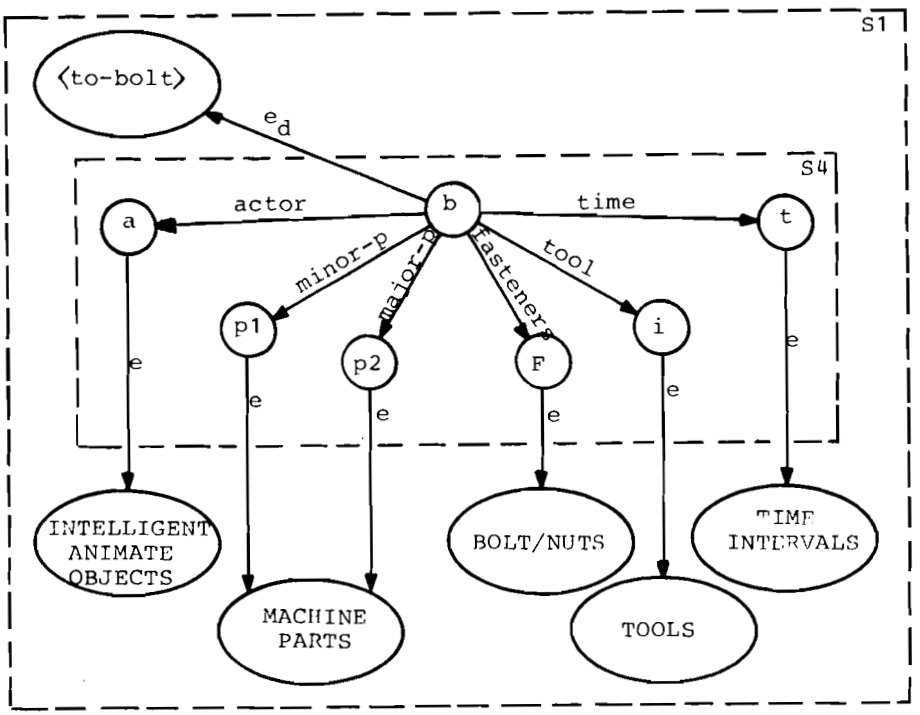


Figure 7. Delineation of ⟨to-bolt⟩.

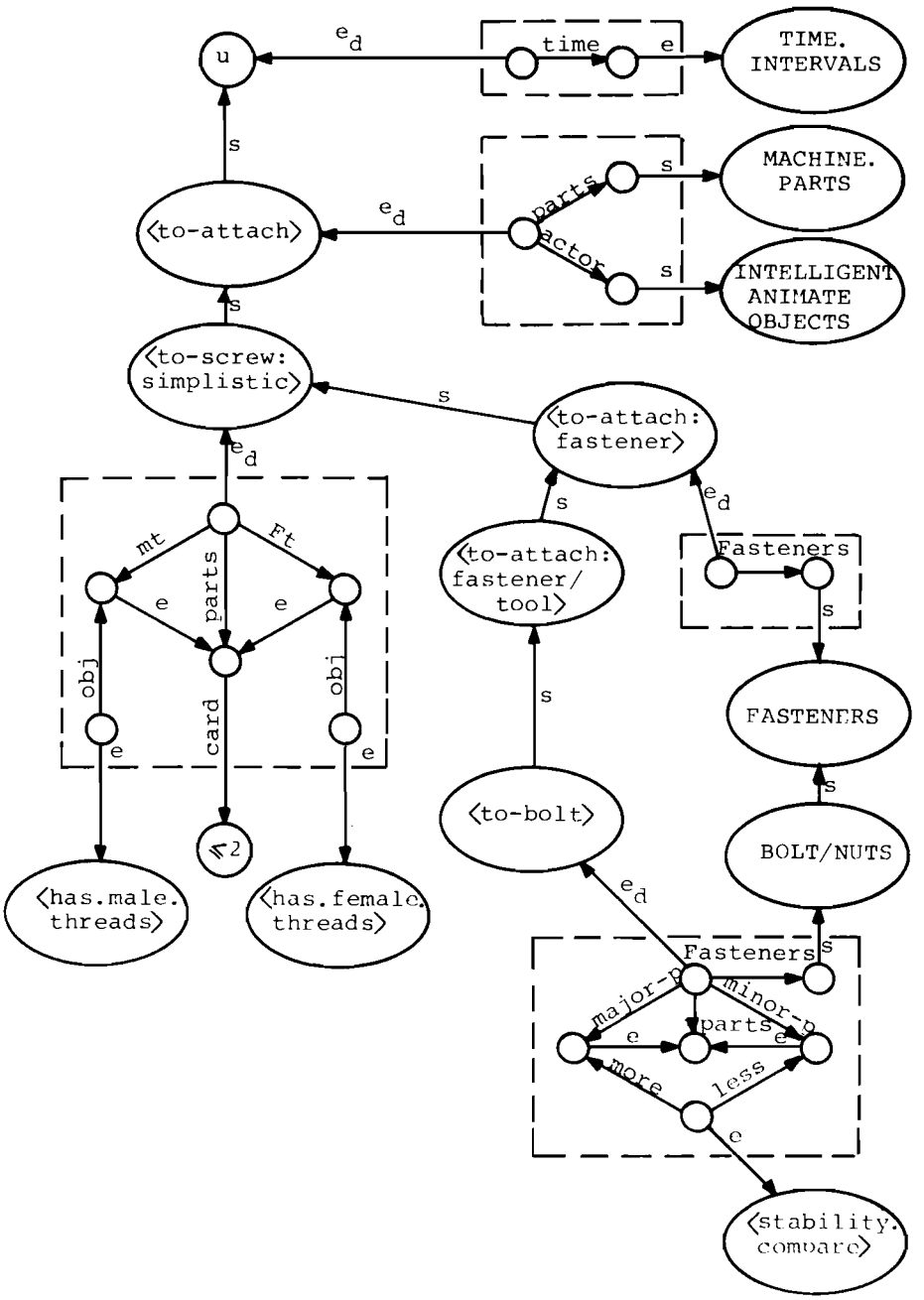


Figure 8. The ⟨to-attach⟩ family.

hierarchy is category U, the universal set. Even U has a delineation since all objects (including events and situations) exist over some (possibly one-point or infinite) time interval. A subset of U is ⟨to-attach⟩, the set of all attaching events of any nature whatever. Members of ⟨to-attach⟩ inherit the time attribute from supercategory U and add two additional attributes, #@parts and #@actor, of their own. In general, each attaching event involves a set of #@parts that an #@actor binds together in some way.

Two subcategories of ⟨to-attach⟩ are shown in the figure. The first is ⟨to-screw:simplistic⟩, which is the set of events in which two threaded objects, one (#@mt) with male threads, the other (#@ft) with female threads are engaged by twisting. Notice that the delineation rule of this category shows that the #@mt and #@ft are both elements of the #@parts. The cardinality of #@parts is at most two (but could be one as for a garden hose with one end attached to the other).

A second subcategory of ⟨to-attach⟩ is ⟨to-attach:fastener⟩, the category of fastening events in which the #@parts are attached with fasteners. (Screwing a lightbulb into a socket involves no fasteners and is a simplistic screwing event. Nailing a sign to a post involves a nail as a fastener.) The delineation of ⟨to-attach:fastener⟩ simply adds the attribute of @fasteners.

Category ⟨to-bolt⟩ is a subcategory of ⟨to-attach:tool⟩ which is a subcategory of ⟨to-attach:fastener⟩. The delineation of ⟨to-bolt⟩ shown in Figure 8 indicates how the #@major-p and the #@minor-p are related to #@parts and to each other. Further, the #@fasteners used by bolting events are restricted to be bolt/nuts as opposed to any type of fastener. Linkage to a process automaton that indicates the sequence of changes characterizing a bolting event might also be included with the category information but has been omitted here for simplicity.

5. Abstraction

Since a user may think at varying levels of detail, it is important in our second task domain for the semantic system to be able to encode information at multiple levels of abstraction and have some capability for jumping from one level to another. Figure 9 shows one way in which net partitioning may be used to encode an attaching event A at two levels of detail. By viewing the network from the vantage of space S2 (which lies below S1 in the ordering and is a sister space to S3), A is seen to be an element of ⟨to-attach⟩ since the e arc lying in S2 is visible. Since the information lying in S3 is invisible from S2, A appears to have only an #@actor and a set of #@parts and is not seen to involve #@fasteners. From S3, the same event may be viewed with more detail. First, the e arc from A to ⟨to-attach⟩ is invisible and A is thus seen as an element of ⟨to-bolt⟩, a subset of ⟨to-attach⟩. Further, at this finer level of detail, the #@fasteners involved in the attaching (bolting) event are visible (as are tools, and so on, which are omitted from the figure for simplicity.)

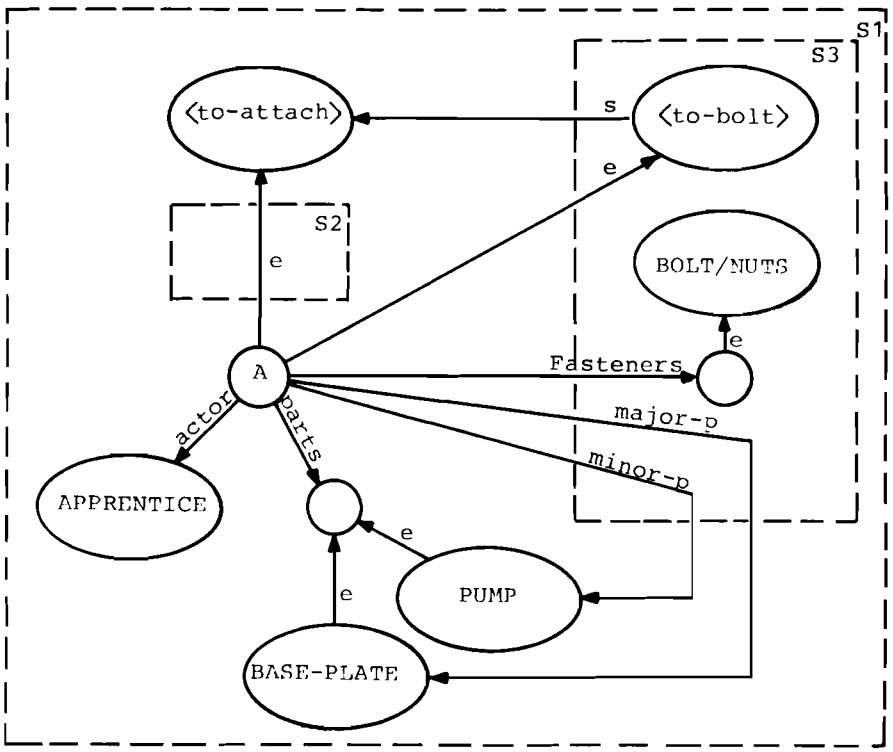


Figure 9. Viewing a bolting at two levels of detail.

6. Processes

An important aspect of the workstation domain is that of change. Since sequences of change tend to follow certain regular patterns, it is conveninet to organize the recurring sequences of change into categories, grouping similar sequences together. Each category of sequential change is tantamount to an event category, the members of which may be thought of as individual enactments of a common plot or script that encodes a generalized pattern of change. For example, every event of tightening bolts follows the plot that consists of finding a wrench, putting the wrench on the bolt, twisting the bolt clockwise, and so on. Each enactment casts different actors in the various roles, but follows the same basic pattern.

Since the members of a particular event category may be distinguished as exactly those instantiations of sequential change that follow some particular script, the script itself forms the basis for a rule defining the event category.

Acknowledgment

This report has been extracted from W.E. Walker et al., Speech Understanding Research, Annual Report, Project 3804, Artificial Intelligence Center, SRI, Menlo Park, California, 1975.

The Choice of Semantic Representation in a QAS

J.C. Simon

1. Introduction

It is now generally recognized that a question-answering system has to have some limitations in order to be realistic:

a) in the outside universe (the domain),

b) in the language of communication (input-output),

c) in the ability of reasoning (the intelligence).

At the Institut de Programmation, we decided to select the restricted domains of learning elementary arithmetic and simple stories (fables) in order to bypass, in a way, the linguistic problems of the communication language and to restrict the semantic representations to a strict model.

In fact, the choice of the "deep structure" and of the basic operators acting on it is made a priori, with some learning abilities. A capacity of planned extension seems essential to any QAS and cannot reasonably be hoped for if the semantic representation is not restricted to a formalized simple structure. Another point of importance is the nature of the implementation of the semantic information. In a sense, a comparison may be made with the implementation of a "computing function"--either an algorithm or a table. The first type is quite general and works on all the variable space (possibly); the second is more flexible, but no "extension" is permissible outside of the table entries. In representing a "semantic function," the same conflict arises between generality and exceptions; a linguist's proverb says, "a grammar always leaks."

A compromise has to be decided for the representation of the semantic space and operators acting on that space. Building it around a theorem prover means that one believes that the universe can be modeled by algorithms rather than by a set of specific informations. On the other hand, semantic nets have been found objectionable because of their limited capacity for reasoning. Our choice is to build up the semantic information into an improved semantic net, relying on the idea of categories (later on also called types or class), and also, at the same time, to build up the communication language.

2. Project of J.P. Jouannaud, G. Guiho, J.P. Treuil

The domain is formation concept in elementary arithmetic. The data base is made of "types" connected by liaisons. These types are defined by some memory information and/or properties. They are interpreted by a structure. Operators may act on the data base and on the interpretative structure. A set of heuristics controls the learning phase.

Only questions, statements, and answers are planned. The difficulties of the communication language are deliberately bypassed. In fact, we are mainly interested in the learning process--here the formation of concepts.

The operators are represented in λ-calculus formalism, convenient as the atomic types are strings of characters. At the start, the usual basic operators of λ-calculus are given.

Only a few types exist at the start, the idea being to generate new types from the base types, the question-answers, and the operators. Heuristics also allow generating new operators and estimating them (partially).

A new information such as a statement will:

a) either be put in an existing type,

b) modify the structure and create new types.

The "autonomous" work on the semantic structure allows:

a) creating new operators,

b) building new links between types,

c) creating new atomic types and possibly new heuristics of utilization.

The λ-calculus is used in (a) and (b), in particular the "combinators" such as substitution, abstraction, and generalization.

Heuristics of forgetting are planned, so as to limit the growth. The interaction with the professor will also allow pruning the structure through the human evaluation of usefulness.

3. Project of G. Sabah, G. Loyo, M. Puzin

This project relies on the facilities offered by SIMULA 67, which easily allows setting up categories by the class process. The categories are tree structures of classes, each of which has attributes and/or procedures. By linking various trees, a lattice structure may be obtained.

A phrase is again only an assertion, a question, or an answer. The deep structure into which the interpretation is made is unique. Thus a phrase has to be matched to the deep structure. The operators acting on the semantic net are the monitor, the analyzer (input-output), and the operators exploiting or modifying the memory structure.

Generalization capacity is naturally embedded in the tree structure of the class categories. The different types of classes utilized are:

a) ELEMENT - a word of the vocabulary;

b) ASSERTION - contains the information relative to a statement;

c) OBPEN - the different nominal syntagms (N.S.) occurring in a phrase;

d) PREDI - the predicate of the phrase.

Along with Fillmore, Anderson, and Bower, we decided that a phrase is a "predicate" about an "object of thought," in a context (time and locus). This is represented by the simple rules

Assertion $\longrightarrow$ (context) + N.S. + Predicate

Context $\longrightarrow$ (N.S.) + (N.S.)

N.S. $\longrightarrow$ {word} + {assertion}

( ) signifies one or $\emptyset$, { } one or more.

As an example, let us give the composition of an object of the class PREDI:

Qualification (points toward an ELEMENT)

Source
Principle agent
Theme
Goal
Secondary agent
Locus

} (points toward an OBPEN) represents the semantic relations: the case.

Most of the time, an object PREDI is a verb, though it also may be an adjective or a preposition. For a verb, five "cases" are distinguished.

state verb: the sky is blue

processes verb: the fox sees grapes

action verb: Mary dances

action processes verb: Paul cuts a string

locus verb: Paul is in the garden.

An incoming phrase will either:

a) modify the memory,

b) provoke an action (reasoning with a possible answer).

The phase, "the fox sees grapes," belongs to (a); the words are recognized but not understood. The phrase, "speak about the grapes," is understood in the sense that it will be followed by the execution of a procedure giving a speech about grapes. The phrase, "the fox is an animal," is understood in the sense that the object fox is created as the sum of the object animal.

The generalizations are potentially made by going up and down a class tree, as in NAGAO semantic trees. The restrictions are made by interaction with the professor.

The three main directing ideas of this project are:

a) the existence of a minimum a priori knowledge,

b) the choice of a semantic deep structure,

c) the possibility of a learning for the operators modifying and handling this structure.

4. Conclusion

To date (June '75), the implementation by program of the two projects is 50% made up. The first results have been obtained and are encouraging. The emphasis of our approach is on the capabilities of the semantic deep structure and on its extensibility. Of course, at the start, the linguistic communications are poor, in the sense that the QAS does not understand certain types of phrases. The communication language has to be learned and refined during the learning phases.

Analysis of Japanese Sentences by Using Semantic and Contextual Information

Makoto Nagao and Jun-Ichi Tsujii

1. Introduction

We are developing a question-answering system with natural language input. In this paper, we describe the organization of the natural language parser, which we have developed in the past two years. The parser can transform a fairly complicated sentence into a deep case structure by utilizing detailed semantic descriptions in the dictionary and contextual information extracted from the preceding sentences.

In most of the artificial intelligence approaches to the understanding of natural language, knowledge structures, which are convenient for logical operations, are used in order to represent both semantic and contextual information. In fact, logical expressions are suitable for solving some kinds of problems in natural language, especially in the deep deductive stage of understanding.

However, the intuitive reasoning, which is not well formalized as logical operation, plays more important roles in the human understanding process. We think that intuitive reasoning is completely based on the language activity in the human brain, and the association functions such as semantic similarity among words, semantic deepness of an interpretation, probability of associative occurrence of events, etc., together with the knowledge structures and short-, intermediate-, and long-term memories, are inherent in the question-answering activities. Our approach to question-answering, therefore, is not based on formal logics, but on deep structures of natural language and some heuristics.

We express the meaning of a word in view of how it is related to other words. The meaning of a verb is described in the form of "activity patterns" in the verb dictionary. Activity pattern is actually the case frame of a verb and its related additional information. Case frame represents itself as to what kind of cases the activity requires, and what kind of objects will be supplied for each case. Additional information concerns how an activity pattern is related to other activity patterns by causal relationships. By using this description, we can infer what activities and change will follow the present activity.

The human abstraction process from a sentence to cognition consists of several stages. The logical expression may be used somewhere in the final stages. In the intermediate stages, we use several kinds of memory structures such as short-term memory, intermediate-term memory, and so on. We provide such memory structures in our system. By combining these structures with semantic descriptions of words, our system can analyze fairly complicated Japanese sentences.

To start with the construction of a question-answering system, we have confined ourselves to the field of elementary chemistry where we can describe rather easily the semantic world in detail. From elementary and middle-school textbooks on chemistry, we chose sample sentences of natural language analysis. This choice was due to the small irregularity of the styles in these Japanese sentences, although they contain enough complexity of the Japanese language.

The Japanese language is a typical SOV language. The word order is rather arbitrary except that the main verb comes last. The subject noun is often omitted when it can be easily surmised. Cases such as subjective case, objective case, and dative case are syntactically indicated by postpositions, but they can be ambiguously used for several cases. So the determination of the sentence structure is strongly dependent on semantics and contextual situations. This paper is an attempt to surmount these difficulties by an artificial intelligence approach. Use is made of chemistry semantics, contextual information, and so on, and the analysis program is written by PLATON (Programing LAnguage for Tree OperatioN), which we have developed on LISP. It has the facilities of pattern matching and flexible back-tracking. A grammar written by PLATON not only maintains the clarity of syntax, but also provides adequate semantic and contextual representations.

2. Lexical Descriptions of Words

2.1 Noun Description

Most nouns have definite meaning by themselves. We call them entity nouns. An entity noun is considered to represent a set of objects, and therefore is taken as a name of the set. The objects belonging to the set may share the same properties. By introducing another property, the set may be divided into a number of subsets, each of which is expressed by another noun. We describe such set-inclusion relationships and set properties in the noun dictionary.

We represent a property of a noun by an attribute-value pair expressed simply by an abbreviated form (A V). For instance, the dictionary contents of the nouns "material" and "liquid" are:

material: ((SP) (ATTR(STATE)(MASS)(COLOR)(SHAPE)...))

liquid: ((SP material)(ATTR (STATE LIQUID)(SHAPE NIL))).

The descriptions (STATE)(MASS) and so on in the definition of "material," which lack the value part V, show that "material" may have arbitrary values of these attributes. In the definition of "liquid," there is a SP-link to "material," which means that "material" is a superset concept of "liquid," or that "liquid" is a subset or a lower concept of "material." The objects belonging to a subset are considered to have the same properties as the objects of the superset, in addition to the properties described explicitly in its definition.

By the above descriptions, we can see that the value of the attribute STATE of "liquid" is LIQUID, and that of SHAPE is the special value NIL. The value NIL means "liquid" can not have any value of SHAPE. By tracing up the SP-links, we can retrieve all the (A V) pairs of an object, where we assume the value of an attribute of a lower concept has precedence over that of the upper concept. For instance, we can obtain the following full description of "liquid":

liquid: ((ATTR (STATE LIQUID)(SHAPE NIL)(MASS)(COLOR)...)).

These upper-lower relationships among entity nouns are not expressed by a tree structure. Some nouns may have properties of more than one different noun. "Water" is such an example. "Water" has the properties of both "liquid" and "compound." We permit a noun to have several upper concepts. The relationships are then represented by a lattice as shown in Figure 1.

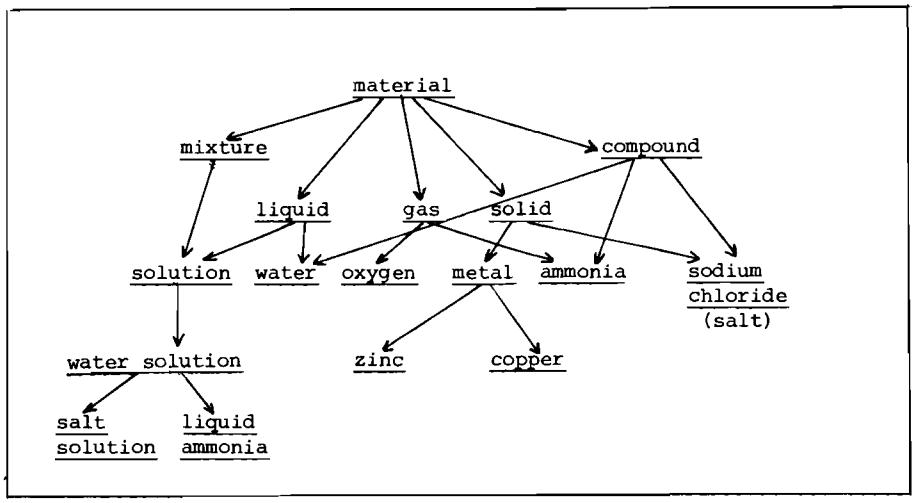


Figure 1. Upper-lower relationships among nouns.

Although most nouns are regarded as entity nouns, there are a few nouns which have relational functions. We call them relational nouns. "Father" is a familiar example of such a word. In order to identify a person indicated by the word, we have to know whose father he is. In the chemical field, we can easily find such nouns, for example "weight," "temperature," "color," "mass," and so on. They are called attribute nouns. Their meanings are described in different ways from that of ordinary nouns. Figure 2 shows some examples. Here, A-ST designates the standard name of the attribute that is expressed by the word.

```
( ONDO          ( (NF N-A)(A-ST  TEMPERATURE)(SP ZOKUSEI   RYOU   )))
  temperature                                    attribute quantity

( SHITSURYOU    ( (NF N-A)(A-ST MASS)(SP ZOKUSEI   RYO    )))
  mass                                  attribute quantity

( OOKISA        ( (NF N-A)(A-ST VOLUME MASS LENGTH AREA )
   size                          (SP ZOKUSEI    RYO    )))
                                     attribute  quantity

( IRO           ( (NF N-A)(A-ST COLOR)(SP ZOKUSEI   SHITSU)))
  color                                 attribute quality
```

An attribute noun may express more than one standard attribute OOKISA (size) expresses VOLUME, MASS, LENGTH, AREA, and which attribute it expresses depends upon what entity noun is used with it.

Attribute nouns are further classified into two groups--quantitative and qualitative. A qualitative attribute noun cannot be a case element of the verbs which require quantitative nouns. The verbs "FUERU" (increase) and "HERU" (decrease) are such examples.

Figure 2. Attribute nouns.

"Liquid" is another example of relational nouns. The Japanese word which corresponds to "liquid" is "EKITAI." Though "liquid" in English represents two parts of speech, that is, noun and adjective, "EKITAI" in Japanese is categorized syntactically as a noun. But semantically "EKITAI" has two different meanings, one corresponding to the noun usage of "liquid," the other corresponding to the adjective usage. The noun "EKITAI" in the adjective usage is called a value noun of the attribute STATE. Another Japanes word "AKAIRO," which corresponds to "red color" in English is also a value noun of the attribute COLOR. Figure 3 shows the description of these nouns in the noun dictionary.

```
( EKITAI   ( (NF N-E)(SP BUSSHITSU)(ATR (STATE LIQUID)(SHAPE NIL)))))
  liquid                material

           ( (NF N-V)(V-DESCRIPTION (STATE LIQUID)))

( AKAIRO   ( (NF N-E)(SP  IRO )))
  red                   color

           ( (NF N-V)(V-DESCRIPTION (COLOR RED)))
```

Figure 3. Value nouns.

There are other kinds of relational nouns in Japanese, e.g., action noun, prepositional noun, anaphoric noun, and function noun. Action noun is the nominalization of a verb. For example, "KANSATSU" (observation) is the nominalization of the verb "KANSATSU-SURU" (observe.) We describe this in the dictionary by giving the link to the original verb and other additional information.

There are no words in Japanese which correspond to prepositions in English, but some special nouns play the role of prepositions. We call such nouns prepositional nouns. Because a prepositional noun usually has more than one meaning, just as a English preposition has, we attach semantic conditions in order to disambiguate them. Figure 4 shows examples of lexical descriptions of prepositional nouns. Corresponding to each meaning of them, we give a triplet. The first element of it is the semantic condition. If the condition is satisfied, the corresponding second element is adopted as the meaning. If not, the next triplet is tried. The second element of a triplet represents the whole meaning of the phrase. For example, the whole meaning of the phrase "TSUKUE (desk-entity noun) NO (of) UE (on--prepositional noun)" (on the desk) is PLACE. The third element of a triplet expresses the relationship by which the other noun in the phrase may specialize the whole meaning.

```
( MAE      ( (NF N-P)(F-DESCRIPTION)))
  before
  in front of   ((CAT ACTION)     TIME BEFORE))

                ((AND (CAT  N-E)(LOWER DOUGU     )))
                                      instrument
                ((BUSSHITSU)   PLACE IN-FRONT-OF)
                  material

( NAKA     ( (NF N-P)(F-DESCRIPTION)))
  in
                ( OR (LOWER YOUKI    )(LOWER EKITAI) PLACE   IN)
                          container    liquid
```

Figure 4. Prepositional nouns.

2.2 Verb Description

Verbs, adjectives, and prepositions in English have relational meanings with nouns. A verb represents a certain activity, while the agent who causes the activity is not inherent to the meaning of the verb, the same with the object which the activity affects, and the other components. They appear in a sentence with certain loose relations to a verb of the sentence. In our system, the meaning of a verb is described by setting up several relational slots which will be filled in by nouns. In this sense, the meaning of a verb is not confined to itself, but is related to nouns.

We describe these relations by using the case concept introduced by C.J. Fillmore. Case may be looked upon as a role which an object plays in an activity. Because several objects usually participate in an activity, there are several cases associated with an activity. An object is expressed by a noun phrase, and an activity by a verb in a sentence. A sentence instantiates an activity by supplying noun phrases to the cases associated with the activity. We call such instantiated activity an event. The problem is to decide what case a noun phrase takes to a verb in a sentence.

Though there are usually some syntactic clues in a sentence as to how it instantiates an activity, they are not enough to decide the case relationships between noun phrases and a verb. To do it, we need not only syntactic information but also a semantic one. A verb has its own special usage patterns. That is, certain kinds of cases are necessary for the activity, and certain kinds of objects are preferable as the case elements. We call these patterns case frames of verbs, and express them by a list of case pairs such as (CASE-ELEMENT: NOUN). A verb usually has more than one case frame corresponding to the different usages of it. A typical description of a verb is shown in Figure 5.

```
( TOKASU    (CF
  melt      (( ACT NINGEN      )( OBJ KOTAI)(IN EKITAI))))
  dissolve         human being        solid     liquid

            ((ACT NINGEN       )(OBJ KOTAI)(INST))
                  human being        solid

            ((ACT SAN )(OBJ  KINZOKU  ))
                  acid       metal
```

Figure 5. A typical description of a verb.

According to this description, we understand that the verb "TOKASU" (melt, dissolve) has two different usages. In one usage, the verb takes the ACTOR case, and prefers to take the subconcepts of the noun "NINGEN" (human being) as the case element. In such a way, case frame descriptions are closely connected with the noun descriptions, especially of the upper-lower concept relationships among nouns.

There are two kinds of cases, intrinsic case and extrinsic case. The intrinsic cases of a verb are essential ones for the activity, but extrinsic cases are not. For example, the cases of TIME and PLACE, which express when and where an event occurs, are extrinsic ones for ordinary verbs. Most activities can be modified by these extrinsic cases, but the kinds of nouns preferred for these case elements do not strongly depend on the kinds of activities. Therefore, we describe only the intrinsic cases in the verb dictionary. We set up fourteen cases, as shown in Table 1, for the analysis of sentences in the textbook of elementary chemistry.

In order to resolve the syntactic ambiquities of a sentence, it is also necessary to utilize contextual information obtained by the preceding sentences. When one knows a certain event has occurred, he can expect what events will occur in succession, and what changes the objects participating in the event will suffer. This kind of expectation plays an important role in understanding sentences. Various kinds of associations cluster richly around an individual activity such as human knowledge. One can perform contextual analysis of language by using these associations.

We append this kind of individual knowledge to a case frame of a verb. The following two items are described for each verb in the verb dictionary.

1) CON: this means the consequent activities which may probably follow the activity of the verb, but not necessarily.

2) NTRANS: this means the resultant relations which may hold between the activity and the objects in view of how the objects will be influenced by the activity. In our system, the influence on the objects is described by the following three expressions.

   a) (ADD case a-set-of-(A V)-pairs)

   b) (DELETE case a-set-of-attributes)

   c) (CREAT lexical-name-of-an-object a-set-of-(A V)-pairs)

By a), we mean that the object in the case indicated by the second element comes to have a set of properties indicated by the third element; b) is for the deletion of a set of properties

Table 1.

1) ACT: ACTor is responsible for action.

KARE- GA IOU -O SHIKENKAN-NI IRERU.
(he)-(ACT) (sulfur)-(OBJ) (test tube)-(IN, PLACE, etc.) (put in)

(He puts sulfur in a test tube.)

In the chemical field, a chemical object is often regarded as ACTor of an action, though it does not exercise intention to do the action. For example, the underlined word in the following sentence is regarded as ACT.

ENSAN -WA DOU -O TOKASU.
(hydrochloric acid)-(all cases) (copper)-(OBJ) (melt)

(Hydrochloric acid melts copper.)

2) SUBJ: SUBJect is one which is the main object of talk in a sentence.

a) KITAI-NO TAISEKI - GA FUERU.
(gas) (volume)-(SUBJ) (increase)

(The volume of the gas increases.)

b) IOU - WA KIIROI.
(sulfur)-(SUBJ) (yellow)

(Sulfur is yellow.)

c) KITAI- GA HASSEI-SURU[1]
(gas)-(SUBJ) (be created)

(Gas is created.)

[1] In Japanese, this sentence is not in passive voice.

3) OBJ: OBJect is the receiving end of an activity. It is affected by the activity.

a) KARE- GA MIZU - O NESSURU.
(he)-(SUBJ) (water)-(OBJ) (heat)

(He heats the water.)

b) ENSAN - GA AEN - O TOKASU.
(hydrochloric acid)-(ACT) (zinc)-(OBJ) (melt)

(Hydrochloric acid melts zinc.)

Table 1 (continued).

4) IOBJ: This case is semantically the most neutral case. It is an object or concept which is affected by an activity, and which is not OBJect. This case is usually specialized by other cases such as PLACE, TO, IN, and so on, depending on the semantic interpretation of the verb itself.

DOU - O	ENSAN-NI	TSUKERU.
(copper)-(OBJ)	(hydrochloric acid)-(IOBJ) (IN)	(dip)

(Someone dips copper in hydrochloric acid.)

5) FROM: FROM describes a former position or state. By the physical objects with verbs: put, give, receive, and so on, FROM is the previous location of the OBJECT.

a) SHIKENKAN-KARA BEAKER-E EKITAI-O
(test tube)-(FROM) (beaker)-(PLACE) (liquid)-(OBJECT)

UTSUSU.
(pour)

((Someone) pour the liquid from the test tube into the beaker.

b) KAGOUBUTSU-KARA SUISO -GA HASSEISURU.[1]
(compound)-(FROM) (hydrogen)-(SUBJ) (be generated)

(Hydrogen is generated from the compound.)

[1] In Japanese, this sentence is not in the passive voice.

6) RESULT: RESULT is to the future as FROM is to the past.

MIZU - GA	SUIJOUKI-NI	NARU.
(water)-(SUBJ)	(steam) (RESULT)	(become)

(The water becomes steam.)

7) INST: INSTrument is an object used as a tool.

GAS-BURNER-DE	MIZU - O	NESSURU.
(gas burner)-(INST)	(water)-(OBJ)	(heat)

((Someone) heats water by a gas burner.)

Table 1 (continued).

8) TO: This is a destination of something in the action.

a) SUIBUN - GA NAKUNARU TOKI MADE NESSHI TSUZUKERU.
(water)-(SUBJ) (be gone) (time) (till) (heat) (continue)

(Continue to heat (it) till water is gone.)

b) MIZU-O 10°C-NI HIYASU.
(water)-(OBJ) (10°C)-(TO) (refrigerate)

((Someone) refrigerates the water to 10°C.)

9) FACT:

KORE - O SHITSURYOUHOZON-NO HOUSOKU- TO IU.
(it) -(OBJ) (the conservation of mass) (law) -(FACT) (call)

(We call it the law of conservation of mass.)

10) PLACE:

ALCOHOL-LAMP-NO YOKO -NI BEAKER-O OKU.
(alcohol lamp) (side)-(PLACE) (beaker)-(OBJ) (put)

((Someone) puts a beaker on the side of an alcohol lamp.)

11) IN: This is the case which is more specific than PLACE.

MIZU - O SHIKENKAN-NI IRERU.
(water) (OBJ) (test tube)-(IN) (pour)

((Someone) pour water in a test tube.)

12) SOURCE: This shows materials of compounds.

ENSOSANNATRIUM -WA ENSO, SANSO , NATRIUM -KARA
(sodium chlorate)-(SUBJ) (chlorine) (oxygen) (natrium)-(SOURCE)

DEKITEIRU.
(consist)

(Sodium chlorate consists of chlorine, oxygen, and natrium.)

Table 1 (concluded).

13) CAUSE: This shows a reason or cause of the activity.

NESSHITA-TAME - NI HAGESHIKU KAGOUSURU.
(heat) -(reason)-(CAUSE) (violently) (react)

(Because (someone) heats (them), (they) react violently.)

14) TIME:

NESSHITA-TOKI -NI SANSO-GA HASSEISURU.
(heat) (time)-(TIME) (oxygen)-(SUBJ) (be generated)

(Oxygen is generated when (someone) heats (it).)

from the object; c) shows that some objects will be created by the activity.

A typical example using CON expression is shown in Figure 6. When we have completed the analysis of the sentence:

IOU-O SHIKENKAN-NI IRERU.
sulfur - (object) test tube-(in, result, etc.) put in

(Someone puts sulfur in test tube.)

each case of the case frame of the verb "IRERU" (put in) is instantiated by an object referred to in the sentence. Then we can instantiate the expression of CON, and we conclude, "the sulfur is in the test tube." Figure 7 shows an example using NTRANS expression. From this expression, one can see the verb "tokasu" has two different meanings. One corresponds to "melt," and the other to "dissolve in" in English. When we analyze the sentence:

DOU-O TOKASU.
copper-(object) dissolve, melt

(Someone melts copper.)

we adopt the first case frame of "TOKASU" (melt), because it gives the highest matched value against the sentence (See section 3.4). As the result of evaluating the NTRANS expression in the case frame, we conclude that the copper is now in the liquid state. In the lexical description, copper is a lower concept of solid, so that copper in general behaves as a solid object.

```
( IRERU    (CF
 put in

           ( ACT    NINGEN) (OBJ   BUSSHITSU) (IN   YOUKI)))
                (human beings)    (material)     (container)

           ( CON (ARU (SOBJ ( * OBJ) (PLACE   (IN ( * IN))))))
```

The function * retrieves the designated case-element of the
of the current frame.

Figure 6.

```
(TOKASU       (   CF
 melt
 dissolve
                ( ACT    NINGEN) (OBJ   KOTAI) (INST)))
                     (human beings)   (solid)

                (NTRANS   (ADD    OBJ   (STATE   EKITAI)))
                                              (liquid)

              ( ACT    NINGEN) (OBJ    KOTAI)  (IN EKITAI)
                    (human beings)   (solid)    (liquid)

                (NTRANS    (CREAT   YOUEKI))
                                   (solution)


                                (   SOLVER   ( * IN))

                                (   SOLVENT  ( * OBJ))
```

Figure 7.

But the copper in the above sentence has the attribute value pair (STATE LIQUID) and will behave as liquid in the succeeding sentences.

On the contrary, when we analyze the sentence:

```
SHIO-O          MIZU-NI                    TOKASU.
(salt-(object)  water-(in, place, etc.)    melt dissolve
```

(Someone dissolve salt in water.)

the second case frame of "TOKASU" (dissolve in) gives the highest value. After the sentence instantiates the case frame, a new object--that is, a mixture which consists of salt and water--will be created.

CON and NTRANS are thus important in the contextual analysis of sentences. The detailed analysis procedure using these expressions is described in section 4.2.

3. Analysis of Noun Phrase

3.1 Properties of a Noun Phrase

In Japanese, two or more nouns are often concatenated by the postposition "NO" to form a noun phrase. Because there are many different semantic relationships among nouns concatenated by "NO," we must decide what relationships may be held among the nouns. Typical examples are shown in Figure 8. The phrase "NOUN+NO" can modify, in principle, all the succeeding nouns in Japanese, so that many different patterns of modification relationships are syntactically permitted. We must decide which one is correct, by considering semantic restrictions.

```
EKITAI -NO    JOUTAI -NO   SANSO-NO      TAISEKI
(liquid)-NO   (state)-NO   (oxygen)-NO   (volume)

(the volume of the oxygen in the state of liquid)

HANNOU    -NO   ATO    -NO  NATRIUM  -NO  TAISEKI -NO   HENKA
(reaction)-NO   (after)-    (natrium)-NO  (volume)-NO   (change)

(changes of the natrium's volume after the reaction)

SANKADOU        - NO   KANGEN
(oxidized-copper)-NO   deoxidization

(deoxidization of oxidized copper)
```

Figure 8.

We extracted sixteen semantically acceptable combinations of two nouns. They are shown in Table 2. Corresponding to these relationships, we prepared sixteen primitive functions. These functions are applied in turn to a noun phrase to decide what relationship is held between two nouns. The order in which these functions are applied is based on the frequency and the tightness of the relations. The scope within which the relations are checked is relatively short and each function checks only one semantic relation. In order to illustrate how these functions perform their tasks, we will show an example of "noun + prepositional noun" phrase.

Table 2. Admissible noun-noun combinations.

---

1) (value noun) + (attribute noun)

(ex) 100gr - NO SHITSURYO
(mass)

(ex) KOTAI - NO JOUTAI
(solid) (state)

2) (value noun) + (entity noun)

(ex) AOIRO - NO KOTAI
(blue--noun) (solid)

(ex) EKITAI - NO IOU
(liquid) (sulfur)

3) (entity noun) + (attribute noun)

(ex) DOU - NO SHITSURYOU
(copper) (mass)

(ex) EKITAI - NO IRO
(liquid) (color)

4) (noun) + (prepositional noun)

(ex) SHIKENKAN - NO NAKA
(test tube) (in)

(ex) HANNOU - NO MAE
(reaction) (before)

5) (anaphoric noun) + (noun)

(ex) MOTO - NO BUSSHITSU
(former) (material)

Table 2 (continued).

---

6) (attribute noun) + (entity noun)

(ex) (100gr -NO) SHITSURYOU - NO DOU
(mass) (copper)

(TAKAI) ONDO - NO EKITAI
(high) (temperature) (liquid)

In this usage, the attribute noun should be modified by another noun or adjective, which specifies the value of the attribute.

7) (noun) + (action noun)

(ex) SANKADOU - NO KANGEN
(oxidized copper) (deoxidization)

IRO - NO HENKA
(color) (change)

8) (time) + (noun)

(ex) (HANNOU - NO) MAE - NO DOU
(reaction) (before) (copper)

The noun-noun combination, "(reaction)-NO (before)" expresses the "time" before the reaction.

9) (place) + (noun)

(ex) (SHIKENKAN - NO) NAKA-NO EKITAI
(test tube) (in) (liquid)

The noun-noun combination, "(test tube)-NO (in)" expresses the "place" in the test tube.

10) (noun) + (conjunction noun)

(ex) SANKA - NO TAME
(oxidization) in order to
by reason of

(ex) HANNOU - NO TOKI
(reaction) (when)

In Japanese, some nouns are used to elucidate the case relationships between a noun phrase and a verb. The noun "TAME" in the first example expresses the cases such as CAUSE, PURPOSE, and so on, and the noun "TOKI" in the second example expresses the case TIME.

Table 2 (concluded).

11) (entity noun) + (entity noun)

(ex) Natrium - NO KAGOUBUTSU
(compound)

The first entity noun expresses the element of the object expressed by the second noun.

12) (entity noun) + (entity noun)

(ex) SANKADOU - NO SANSO
(oxidized copper) (oxygen)

The second noun is the element of the object expressed by the first noun.

13) (entity noun) + (entity noun)

(ex) SHIKENKAN - NO SOKO
(test tube) (bottom)

The second noun is the part of the first noun.

14) (entity nouns) + (entity noun)

(ex) Karium, Natrium - NADO - NO KINZOKU
(etc.) (metal)

The nouns "Karium" and "Natrium" are the lower concept nouns of the last noun "metal."

15) (name) + (noun)

(ex) SHITSURYOUHOZON - NO HOUSOKU
(the conservation of mass) (law)

16) Others

(ex) $1cm^2$ ATARI - NO CHIKARA
(per $1cm^2$) (pressure)

The noun "MAE" is a prepositional noun, and the semantic description of it is shown in Figure 4. It is easily seen that this word has two different meanings.

```
JIKKEN-NO       MAE
experiment      time:    before
                place:   in front of
```

The function for the analysis of this kind of phrase checks at first whether the second noun "MAE" is a prepositional noun. If it is not, then this function fails, and returns the value NIL. In this example, because the word "MAE" is a prepositional noun, the checking proceeds further. The description in Figure 4 shows that if the preceding noun is an action noun, that is, if it is a nominalization of a verb, then "MAE" has the first meaning. Because the noun "JIKKEN" (experiment) satisfies this condition, the checking succeeds, and the function returns the value T. The result of the analysis is shown in Figure 9,a. On the other hand, if the input is:

```
TSUKUE-NO     MAE
desk          before, in front of  ,
```

then the word "TSUKUE" (desk) satisfies the condition of the second meaning, and the result is such as shown in Figure 9,b.

In this way, the sixteen checking functions not only test whether a certain semantic relationship holds among input words, but also disambiguates the meanings of input words.

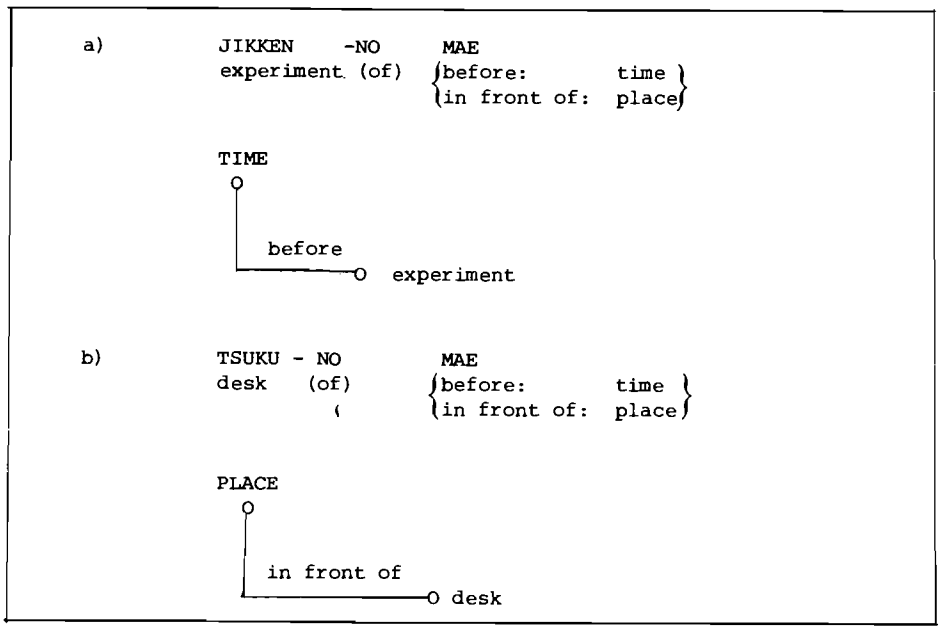


Figure 9. Results of analyses.

3.2 Analysis of a Noun Phrase

We analyze a noun phrase by using the above sixteen checking functions and the projection rule. As stated before, the phrases "noun + postposition NO" and adjectives can modify only the succeeding nouns. We stack in the temporary pile the noun phrases and adjectives which require the noun to be modified by them. The analysis of a noun phrase is carried out by scanning words one by one from left to right. If we scan an adjective or a determiner, we stack the word in the temporary pile. If we scan a noun, we pick up a word from the stack and check whether it can modify the noun. This checking is done by the above functions if the picked-up word is a noun. We also have the checking functions between a noun and an adjective or a determiner. The dictionary content of an adjective is just the same as that of a value noun. The semantic checking function between an adjective and a noun will test whether the noun can have the attribute which is modifiable by the adjective.

The checking of the determiner is a little different and is explained later. The checking process will stop when there are no words in the temporary pile or a picked-up word fails to modify the noun. Then the noun is stacked in the temporary pile. If the temporary pile contains only one noun and there are no words to be scanned in the noun phrase, the analysis succeeds and returns the noun in the stack as the result. The returned noun is called the head noun of the noun phrase. These processes are illustrated in Figure 10.

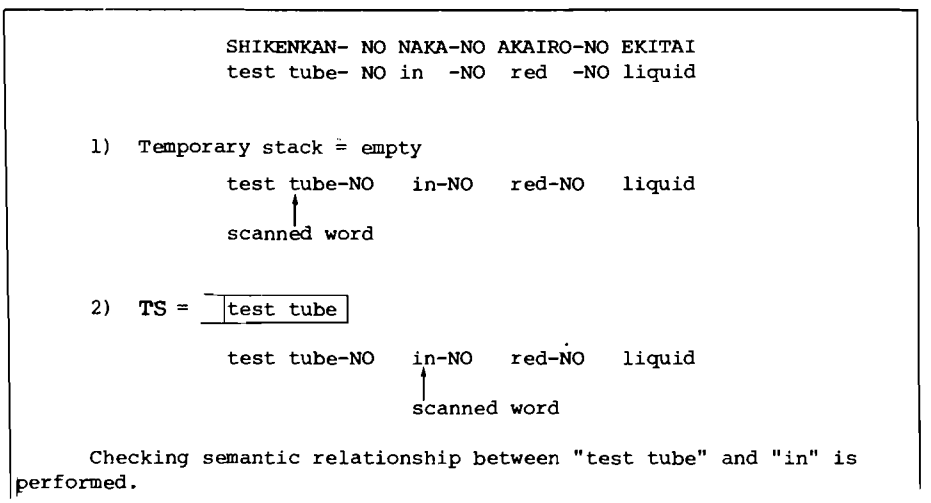


Figure 10.

3) TS = | place |

test tube-No in-NO red-NO liquid
↑
scanned word

The phrase "test tube-NO in" is transformed into the form

PLACE
in
test tube

Checking semantic relationship between "place" and "red" is performed.

4) TS = | | red | place |

test tube-NO in-NO red-NO liquid
↑
scanned word

The checking between "red" and "place" failed to establish a new concept. Therefore, the TS contains the two words--"red" and "place."

5) TS = | liquid |

test tube-NO in-NO red-NO liquid
↑
scanned word

There are no words to be scanned, and the TS contains only one word. So the analysis of this noun phrase succeeds.

The result is as follows: The head noun of this noun phrase is "liquid."

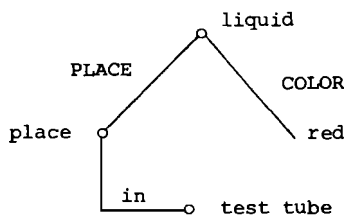


Figure 10 (concluded).

If there are no words to be scanned next and the temporary pile contains more than one word, then the analysis fails and backtracks to the decision points of the program. A decision point in the analysis of a noun phrase is the point at which two nouns have been related by a certain semantic relationship. The established relationship between two nouns during the analysis is by the function which succeeds first. Because the order of checking functions is somewhat arbitrary, in some cases the relationship which has not been checked may be more preferable than the established relationship. We will show such examples.

EKITAI- NO	JOUTAI-NO	HENKA
liquid (of)	state (of)	change

(the change of the state of the liquid)

EKITAI- NO	JOUTAI-NO	SANSO
liquid (of)	state (of)	oxygen

(oxygen in the liquid state)

In the first example the word "JOUTAI" (state) designates the attribute of "EKITAI" (liquid). And the "EKITAI" corresponds to a visible, real object. On the other hand, "JOUTAI" (state) in the second example designates the attribute of "SANSO" (oxygen), and the word "EKITAI" does not correspond to a real object. It is used to specify the attribute "state" of the oxygen. These examples show that the word "EKITAI" (liquid) has two different usages. According to these usages, there are two different semantic constructions of the phrase "EKITAI-NO JOUTAI" as shown in Figure 11.

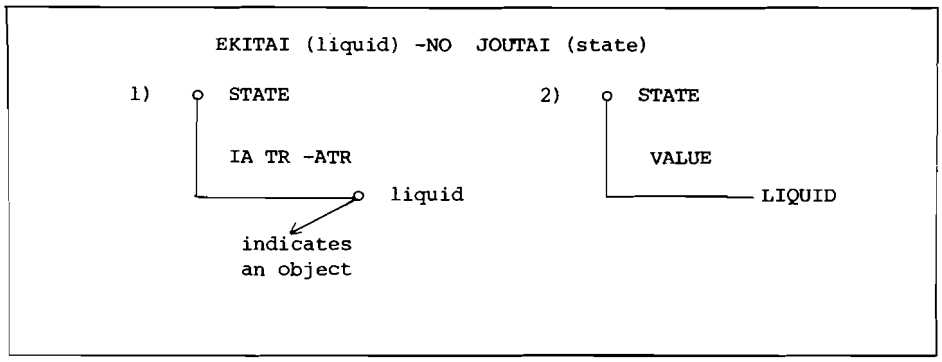


Figure 11. Two different deep structures for the phrase EKITAI NO JOUTAI.

Because we analyze a noun phrase from left to right, we cannot determine which one is correct until we recognize the next word "HENKA" (change, transition) or "SANSO" (oxygen) in the above examples. However, a semantic checking function disambiguates the multiple meanings of the word "EKITAI." If the disambiguation is recognized to be incorrect in the following processing, we must be able to backtrack to the decision point at which this temporary disambiguation was made. We implemented such a process by using PLATON's backtracking facilities. This process is illustrated in Figure 12.

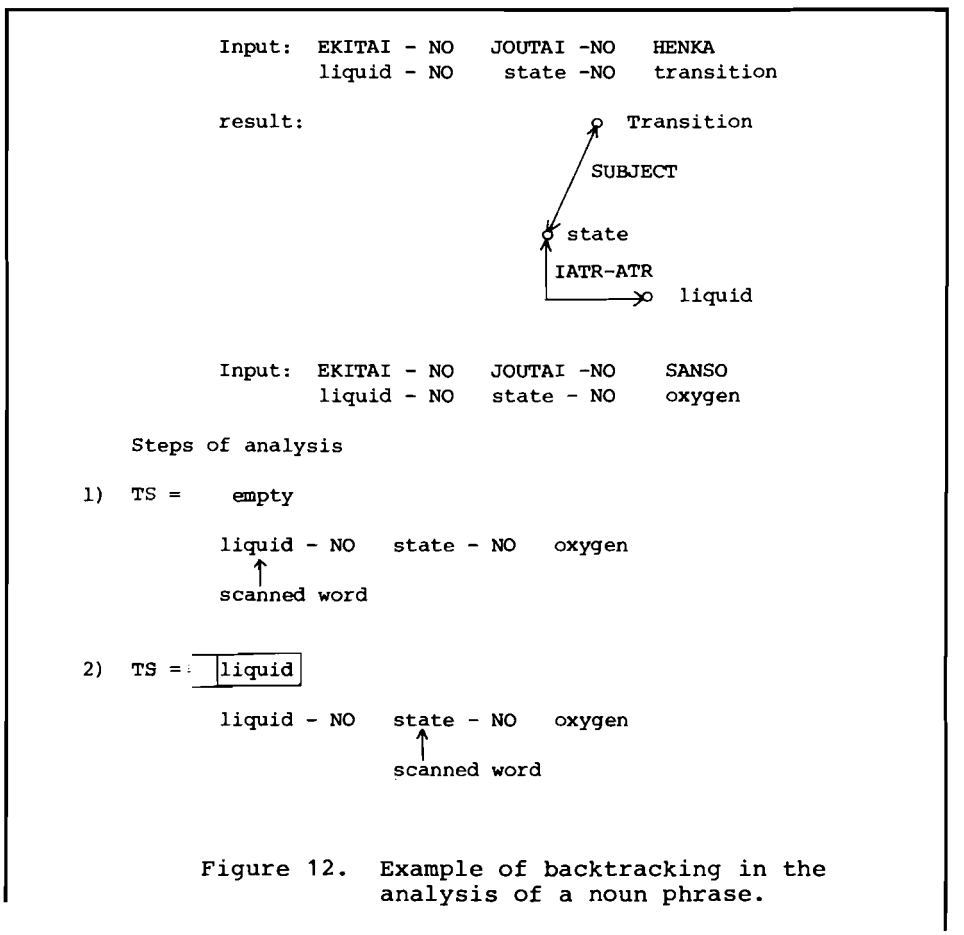


Figure 12. Example of backtracking in the analysis of a noun phrase.

3) TS = state

liquid - NO state - NO oxygen
↑
scanned word

At this point, the first meaning of "liquid" is adopted. That is, the word "liquid" indicates a physical object.

The semantic check between "state" and "oxygen" fails because the attribute noun "state" has been linked to the liquid by the relation IATR-ATR.

So the program will go back to step 3.

4) TS = liquid

liquid - NO state - NO oxygen
↑
scanned word

The semantic check between "liquid" and "state" proceeds further. The semantic checking function "VALUE ATRCHECK" succeeds. This function adopts the second meaning of "liquid."

5) TS = state

liquid - NO state - NO oxygen
↑
scanned word

At this time, because the noun "state" is linked to the liquid by the relation VALUE, the check between "state" and "oxygen" succeeds. The result is as follows. Notice that the noun "liquid" does not express a real object but the value of the attribute "state."

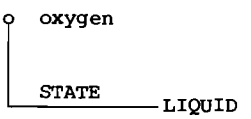


Figure 12 (concluded).

3.3 Analysis of Conjunctive Phrase

The words in Japanese which correspond to "and" and "or" in English are categorized as special postpositions shown in Table 3. We call these postpositions conjunctive postpositions.

Table 3. Conjunctive Postpositions in Japanese.

Postposition	Corresponding English
TO	and
YA	and
MO	and
KA	or
...	...

In Japanese as well as in English, it is difficult to determine the scope of a conjunction. There are some phrases which have the same syntactic structure but semantically different constructions. Some examples are shown in Figure 13. On the other hand, some phrases have different surface structures but convey the same meaning. Examples are shown in Figure 14. As there are few syntactic clues in these examples, we must analyze them by using semantic relationships among words.

At the first stage of the analysis of a noun phrase, we try to find out conjunctive postpositions in the noun phrase. If we cannot find them, the normal analysis sequence described above is applied on the noun phrase. If there is a conjuncitve postposition, the following steps are performed.

Step 1: Because the conjunctive postposition "TO" often has another corresponding postposition "TO" in the succeeding part (Figure 14), we search for this latter postposition in the succeeding part when we find a postposition "TO." If the corresponding postposition "TO" is found, then the noun before the first postposition and the noun phrase interposed between the first and the second postpositions are parallel. The normal noun phrase analysis is applied on this interposed phrase. Go to step 4. If we cannot find out such a postposition, then go to step 2.

Step 2: If the postposition which is found in the phrase is not "TO," or there is not the corresponding "TO," we will undertake the following substeps. (In the following explanation, we use

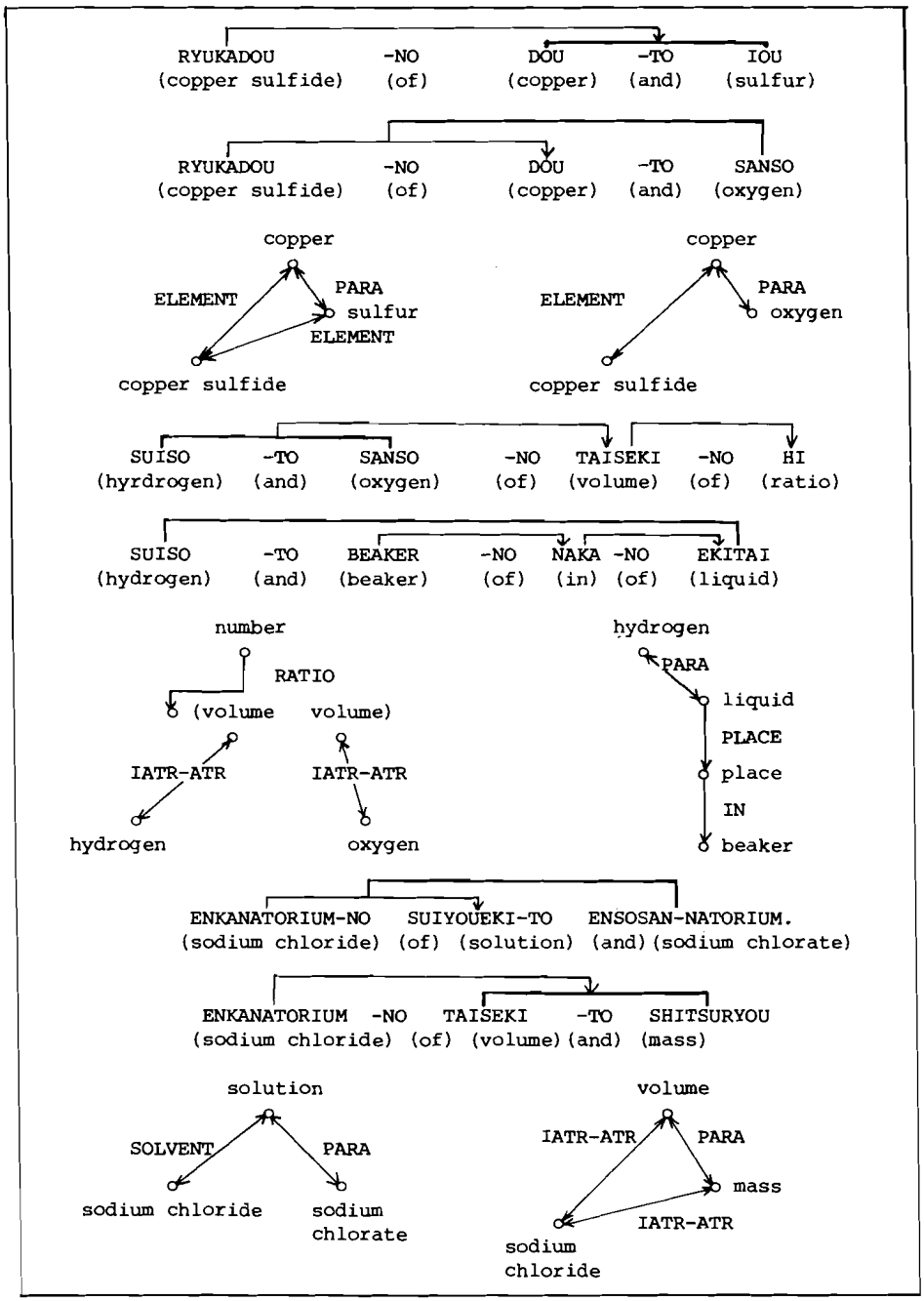

RYUKADOU (copper sulfide) -NO (of) DOU (copper) -TO (and) IOU (sulfur)
RYUKADOU (copper sulfide) -NO (of) DOU (copper) -TO (and) SANSO (oxygen)
copper
ELEMENT
PARA
sulfur
ELEMENT
copper sulfide
copper
ELEMENT
PARA
oxygen
copper sulfide
SUISO (hyrdrogen) -TO (and) SANSO (oxygen) -NO (of) TAISEKI (volume) -NO (of) HI (ratio)
SUISO (hydrogen) -TO (and) BEAKER (beaker) -NO (of) NAKA (in) -NO (of) EKITAI (liquid)
number
RATIO
(volume volume)
IATR-ATR
IATR-ATR
hydrogen
oxygen
hydrogen
PARA
liquid
PLACE
place
IN
beaker
ENKANATORIUM-NO (sodium chloride) SUIYOUEKI-TO (of) (solution) ENSOSAN-NATORIUM. (and) (sodium chlorate)
ENKANATORIUM (sodium chloride) -NO (of) TAISEKI (volume) -TO (and) SHITSURYOU (mass)
solution
SOLVENT
PARA
sodium chloride
sodium chlorate
volume
IATR-ATR
PARA
mass
IATR-ATR
sodium chloride

Figure 13.

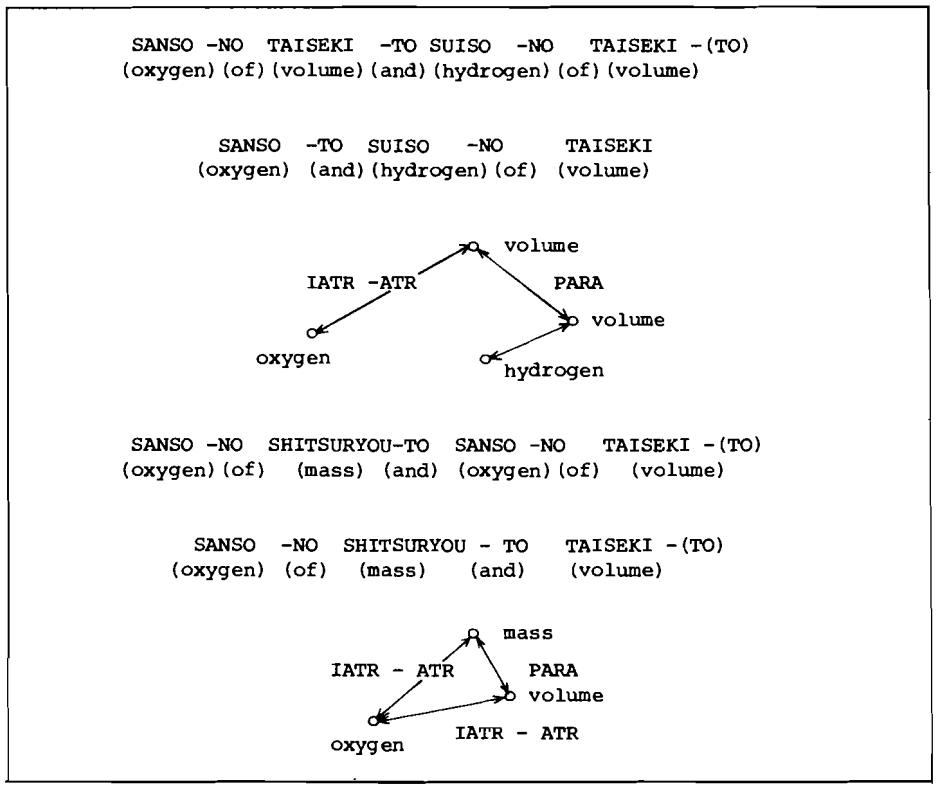


Figure 14.

Noun-1 to designate the noun to which the first conjunctive postposition is attached.)

a) We try to find out the same noun in the succeeding part as Noun-1. If we can find it, let it be Noun-2, and go to step 3.

b) If Noun-1 is not an entity noun, then we try to find out the noun which belongs to the same category as Noun-1. If we can find out such a noun, let it be Noun-2, and go to step 3.

c) We try to find out the noun which as an upper concept in common with Noun-1. If we can find out such a noun, let it be Noun-2 and go to step 3.

Step 3: The phrase between the postposition and Noun-2 are analyzed by the normal sequence of the analysis of a noun phrase. This is the second phrase which constitutes the parallel phrases.

Step 4: The phrase before the postposition is analyzed.

Step 5: We have not determined the left end of the first phrase (Figure 15). In order to determine it, we pick up words one by one from left to right, and check whether each word can modify Noun-2. If we find the word which cannot modify Noun-2, it is considered as the left end of the phrase.

For example, we show the analysis of the following phrase in Figure 16.

RYUKADOU-NO	DOU-TO	IOU-NO	SHITSURYO-NO	HI
copper sulfide(of)	copper(and)	sulfur(of)	mass (of)	ratio

(The ratio between the mass of the copper and the sulfur which constitute copper sulfide.)

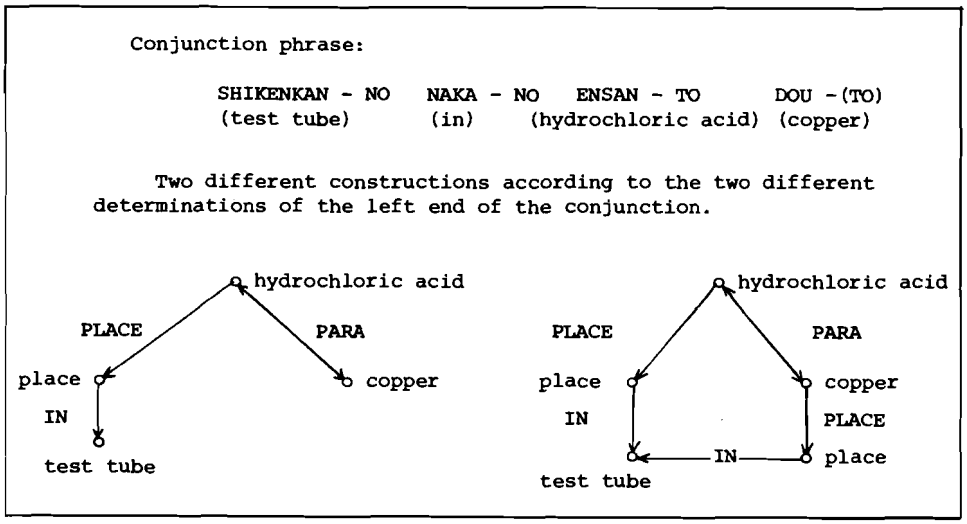


Figure 15.

Phrase:

RYUKADOU - NO DOU - TO IOU -NO
(copper sulfide) (copper) - (conjunctive pp--and) (sulfur)

SHITSURYOU -NO HI
(mass) (ratio)

meaning: the ratio of the masses of copper and sulfur of copper sulfide

sequence of analysis:

1) Find the conjunctive postposition "TO."

RYUKADOU - NO DOU TO IOU -NO SHITSURYOU -NO HI
(copper sulfide) (copper) (sulfur) (mass) (ratio)

former part latter part

2) Find from the latter part the noun which belongs to the same category as "copper." In the above phrase, the noun "sulfur" is found.

RYUKADOU - NO (DOU - TO IOU -NO) SHITSURYOU - NO HI
(copper sulfide) (copper)

temporarily determined scope of the conjunctive phrase

3) Analyze the part before the postposition "TO."

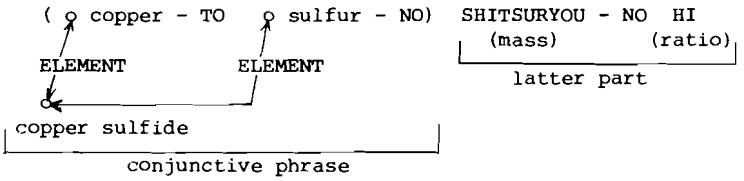
(♀ copper = TO IOU - NO) SHITSURYOU - NO HI
(sulfur) (mass) (ratio)

ELEMENT

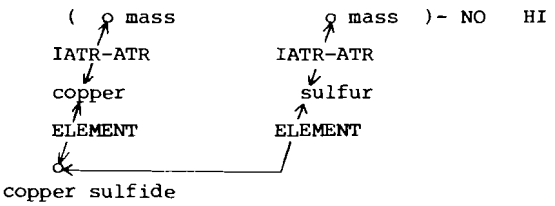
♂ copper sulfide

Figure 16 (continued).

4) The second noun of the conjunctive phrase, "sulfur" is checked against the noun "copper sulfide," which is related to the first noun of the conjunctive phrase, "copper;" we know "copper sulfide" is also closely related to "sulfur." We obtain,



5) The two nouns, "copper" and "sulfur," in the conjunctive phrase are checked against nouns in the latter part. Because the noun "mass" can be related to only one physical object, we produce the new noun "mass" for "sulfur."



6) The noun "ratio" can be related to a conjunctive phrase. So we obtain the following result.

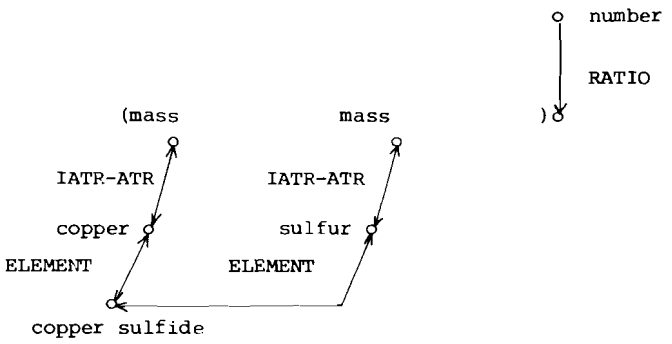


Figure 16 (concluded).

3.4 Analysis of a Simple Sentence

Japanese is a typical example of SOV languages, in which ACTOR, OBJECT, and other case elements of a verb usually appear before the verb in a sentence. This makes Japanese very different from English and European languages. A typical construction of a Japanese sentence is shown in Figure 17. A verb may govern several noun phrases--case elements--preceding it. A relative clause modifying a noun may appear in the form "- - - verb + noun - - -." The right end of the scope of the clause is easily identified by finding out the verb, but the left end of it is syntactically ambiguous. In Figure 17, the noun phrase NP_{i+1} is a case element of the verb V_1. On the other hand, the noun phrase NP_i is governed by the verb V_2. Because the projection rule is kept in Japanese as in other languages, all noun phrase between NP_{i+1} and V_1 are governed by V_1, and the noun phrases before NP_i are governed by V_2. However, in the course of analysis, such boundaries cannot be determined uniquely by syntatic clues alone. To determine them, we must use semantic relationships such as case relationships between noun phrases and verbs.

In English, the case marker in a surface structure is the order of phrases. In Japanese, the order of phrases in a sentence is arbitrary, except that the main verb of a sentence comes at the end of the sentence. A postposition attached to a noun phrase usually shows the case which the noun phrase plays in the sentence. The postpositions usually used in Japanese and the deep cases corresponding to them are tabulated in Table 4. From this table, one can see that a postposition in a surface structure does not necessarily correspond to a unique deep case. In the course of analysis, we must choose an appropriate case by considering the case frames of the main verb and the head noun of the noun phrases.

A postposition also plays the role of a delimiter which shows the right end of a noun phrase. The outline of the analysis of a simple sentence is as follows.

1) At first the program tries to find a verb in the input sentence. Because there may be embedded sentences, which modify nouns in the main sentence, there are usually more than one verb in the input sentence. The program picks up the leftmost verb of the sentence.

2) The part before the verb is segmented into several parts by finding postpositions.

3) Because each segment is supposed to form a noun phrase, they are passed to the program which analyzes a noun phrase.

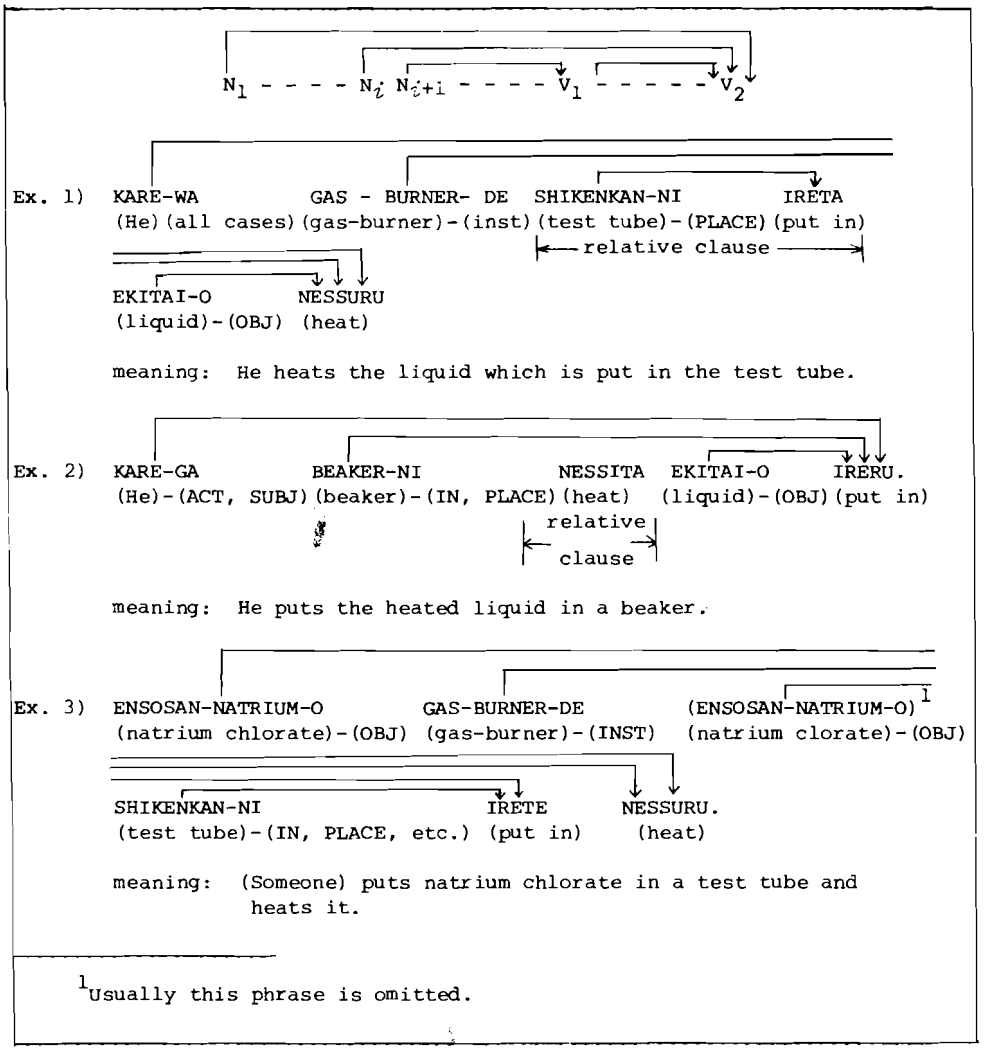

N_1 - - - - N_i N_{i+1} - - - - V_1 - - - - - V_2
Ex. 1) KARE-WA GAS - BURNER- DE SHIKENKAN-NI IRETA
(He) (all cases) (gas-burner)-(inst) (test tube)-(PLACE) (put in)
relative clause
EKITAI-O NESSURU
(liquid)-(OBJ) (heat)
meaning: He heats the liquid which is put in the test tube.
Ex. 2) KARE-GA BEAKER-NI NESSITA EKITAI-O IRERU.
(He)-(ACT, SUBJ) (beaker)-(IN, PLACE) (heat) (liquid)-(OBJ) (put in)
relative clause
meaning: He puts the heated liquid in a beaker.
Ex. 3) ENSOSAN-NATRIUM-O GAS-BURNER-DE (ENSOSAN-NATRIUM-O) 1
(natrium chlorate)-(OBJ) (gas-burner)-(INST) (natrium clorate)-(OBJ)
SHIKENKAN-NI IRETE NESSURU.
(test tube)-(IN, PLACE, etc.) (put in) (heat)
meaning: (Someone) puts natrium chlorate in a test tube and heats it.
1 Usually this phrase is omitted.

Figure 17.

Table 4. Postpositions in Japanese and case relationships.

Postposition	Case
-GA	ACT. SUBJ
-NO	NMOD (ACT. SUBJ)
-O	OBJ
-NI	RESULT, IN, IOBJ, TO, PLACE, CAUSE, TIME
-HE	TO
-TO	FACT, RESULT, TAISHO
-KARE	FROM, SOURCE, CAUSE, METHOD, PLACE, TIME
-YORI	FROM, SOURCE
-MADE	TO
-WA	all cases
-DAKE, -MO, -SHIKA, ---	all cases

4) When all the segments are analyzed and the head nouns are determined, the program checks each noun phrase against the verb, asking whether a case relationship will be satisfied between the noun phrase and the verb. The checking is carried out backward from the phrase nearest to the verb.

5) When there are no noun phrases to be checked, or we find a noun phrase which cannot be a case element of the verb, the checking is finished. If there remains an intrinsic case of the verb which has not been filled in yet, we search for an appropriate noun to fill in the case in the preceding or succeeding sentences. This searching process will be explained in section 4.

We determine, based on the following syntactic and semantic clues, whether a noun phrase can be a case element of a verb.

1) The postposition, which follows the noun phrase. This is a case marker in the surface structure.

2) The case frames of the verb.

3) The head noun of the noun phrase.

The postposition gives a set of possible cases by which the noun phrase is related to a verb. We must choose an appropriate one from this set, by using the second and third information. The case elements in a case frame of a verb are relatively upper concept nouns. Because a real sentence is considered to be an instantiation of a case frame, the nouns in a sentence are lower concept nouns of the nouns in the case frame.

Suppose we analyze the sentence:

SHOKUEN-O	MIZU-NI	TOKASU.
salt-(object)	water-(in, result, time, etc.)	melt, dissolve

(Someone dissolves salt in water.)

We check whether the sentence matches against the case frame of "TOKASU:"

TOKASU: ((ACT human)(OBJ material)(IN liquid)) .

The checking is performed by considering whether "salt" is a lower concept noun of "material," and whether "water" is lower concept noun of "liquid."

Because a case frame contains only intrinsic cases of a verb, we check extrinsic ones when a noun phrase is found not to be an intrinsic case element of the verb. That is, we check whether the postposition can be TIME or PLACE cases, and whether the noun phrase is an instance of the noun "place" or "time."

The above process is somewhat straightforward. But a real sentence has several ambiguities according to the following reasons.

1) A verb may have more than one different usage, i.e., a verb may have several case frames.

2) A postposition can indicate more than one case. Some postpositions can take almost all the cases. "WA" is such an example.

3) The noun modified by an embedded sentence is usually a case element of the sentence, but we have no syntactic clues as to what case the noun phrase takes in the sentence. Therefore, the program derives all possible relationships between nouns and the verb. We choose the most preferable one by using an evaluation function which is empirically established in the following form.

$$f(CFN, C1, C2, C3) = \frac{6 \times C1 + 2 \times C3}{CFN} + \frac{C2}{2}$$

CFN: numer of intrinsic cases in a case frame

C1: number of intrinsic case elements which are filled in by the noun phrases in the sentence

C2: number of extrinsic case elements which are filled in by the noun phrases in the sentence

C3: number of intrinsic case elements which are filled in by the noun phrases in the preceding sentences

The value of this function indicates the degree of matching between sentence and a case frame. Among possible case relationships between noun phrases and a verb, we choose the one which gives the highest matched value, and proceed to the analysis of the remaining part. If it is found to be wrong during the succeeding analysis, control comes back to the point at which the decision was made, discards it, and chooses the one which gives the next highest value.

4. Contextual Analysis

4.1 Basic Approach to Contextual Analysis

A man reads sentences from left to right, and understands them in succession. When he cannot understand a sentence satisfactorily, he goes back to the preceding sentences in order to obtain the keys for understanding. If he cannot discover the keys, he puts this pending question in his memory and proceeds to the next sentence. If he discovers a phrase or a sentence which seems to solve the question, he checks whether it can really resolve the question. If so, he properly organizes it into the previous context and deletes the question from his memory. However, this pending question does not stay in his memory very long. As time passes, the question disappears from his memory.

We think this understanding process of language is not so complicated. It can be realized in an artificial intelligence approach. Though we recognize that some kinds of problems can be solved only by using complicated logical operations, we think most problems in language understanding can be solved by relatively simple operations. Logical operations can be applied only on the complete data base in which all the necessary axioms (corresponding to the human knowledge) are declared and no contradictory axioms exist. In the course of reading sentences, a man has only partial knowledge about the context, and, therefore, his knowledge is not complete. However, he can understand the meaning of sentences before he reads through all of them. This means that a man always does incomplete deductions. Because of this reason, we use, instead of logical operation, heuristically admissible operations which use a memory structure similar to that of human intermediate term memory, and various semantic relationships described in the dictionary.

We can summarize our approach as follows.

1) We memorize context in the form similar to the intermediate term memory of human beings.

2) Two kinds of memories are prepared. One is to represent the current content, and the other is to sustain the pending problems. The former is further divided into the noun stack (NS) and the hypothetical noun stack(HNS). The latter is called the trapping list(TL).

3) Contextual analysis will be performed when a syntactic unit, such as a noun phrase and a sentence, which converys a unit of certain definite idea, has been extracted.

4) NS has the organization from which the theme words of the sentences can be easily retrieved. Here "theme words" mean the key subjects mentioned in the sentence.

5) Sometimes we have to refer to the succeeding sentences in order to understand a sentence. In such cases, we do not immediately refer to the succeeding sentences, but instead, reserve a pending question in TL, and the question will be resolved in the course of analyzing the succeeding sentences.

4.2 Memory Structure for Contextual Information

The analysis of a sentence is primarily guided by the semantic description--case frame--of a main verb, while the contextual analysis is mainly guided by the information about nouns. What objects or concepts are the themes of the sentences, and what has been described about them are usually reflected by the nouns which appear in the sentences and offer important clues for the contextual analysis.

We assign a different LISP atom (produced by the LISP function "gensym") to each noun in the sentences, and put various information about the noun on the property list. We call this LISP atom a "noun atom." The flags tabulated in Table 5 are used. We can retrieve all the descriptions about an object expressed by a noun. We stack these LISP atoms on NS and HNS.

Table 5. Information attached to a noun atom.

Relation	Content
LEX	link to the lexical descriptions
SATR	(A V) pairs which specify this object
CASE	link to the case-frame in which the object appears
PRE	link to the noun atom which appears in the previous sentence, and which represents the same object as this atom
POST	the inverse relation of PRE
SMOD	link to the relative clause which modifies this object
PARA	link to the noun atoms which appear in a conjunctive phrase together with this object

4.2.1 Noun Stack(NS)

When we start to analyze a sentence, we stack a list of noun atoms which are assigned to the nouns in the sentence. These noun atoms are reordered according to their degree of importance. NS has the construction shown in Figure 18.

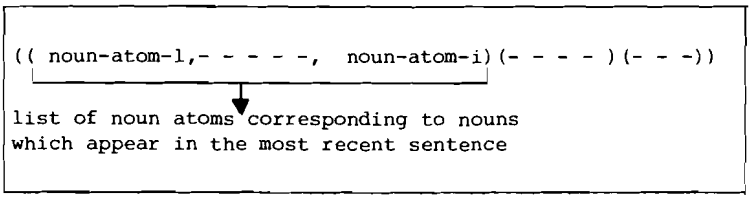


Figure 18. Construction of NS.

To decide how a word is important, we use the following heuristics:

1) In Japanese, a theme word is often omitted or expressed by a pronoun in the succeeding sentences after it appears once in a sentence. In other words, the word which is omitted or expressed by a pronoun is an important word for the understanding of a sentence.

2) A theme word is also used as a "subject" in the surface structure. If we want to emphasize a word in OBJ-case of the deep case structure, or if a word in ACT-case is not worth mentioning, we express a sentence in the passive voice in order to put the stressed word in subject position of the sentence.

3) The importance of a head noun in a noun phrase is greater than that of other nouns.

A simple example is shown in Figure 19. By seeing this example, one can understand that the copper appears in all the sentences, and it is the theme word in these sentences.

```
Input sentence:

          N1                                        N2          N3
        RUTSUBO     -NI                          lOOgr-NO SHITSURYOU-NO
      (melting pot)-(PLACE, TIME, IOBJ, etc.)          (mass)

          N4                              N5
        AEN -O              IRETE  , GAS-BURNER-DE
      (zinc)-(OBJ, IOBJ) (put in)  (gas burner)-(PLACE, INST, etc.)

        NESSHI,    TOKASHITA,
        (heat)     (melt, PAST TENSE)

Meaning of input sentence:

     S1:  (someone) put lOOgr of zinc in a melting pot.

     S2:  (someone) heated it by a gas burner.

     S3:  (Someone) melted it.

Changes of NS

     Beginning of the analysis of S1:  ((N4  N3  N2  N1))

     End of the analysis of S1:  ((N4  N1  N3  N2))

     Beginning of the analysis of S2:  ((N5)(N4  N1  N3  N2))

     End of the analysis of S2:  ((N4  N5)(N4  N1  N3  N2))

     Beginning of the analysis of S3:  (NIL  (N4  N5)(N4  N1  N3  N2))

     End of the analysis of S3:  ((N4)(N4  N5)(N4  N1  N3  N2))
```

Figure 19. Changes of NS.

4.2.2 Hypothetical Noun Stack(HNS)

We first show examples which cannot be properly analyzed without HNS.

a) SUISO-TO SANSO-O 2:1-NO WARIAL-DE
hydrogen-(and) oxygen (object) two to one-(of) ratio-(by)

KONGO-SHI
intermix

(Someone intermixes hydrogen and oxygen by the ratio of two to one.)

KONO KONGOUKITAI-NI - - - - -
this blend gas (place)- - - -

(- - - - in this blend gas.)

b) SHOKUEN 5gr-O MIZU 100cc-NI TOKASU.
salt 5 gr -(object) water 100cc-(in) dissolve

(Someone dissolves the salt of 5 grams into the water of 100cc.)

KONO SUIYOUEKI-WA - - - - -
(the) solution

In these two examples, though the word "KONO," which corresponds to the determiner "the" or "this" in English, is used, the referred object does not exist explicitly in the preceding sentece. The referred object is produced as the result of the event, which is expressed by the preceding sentence. As mentioned before, we attach to a case frame in the verb dictionary what object may be created if the case frame is used.

"TOKASU" (dissolve) has the case frame

((ACT human)(OBJ material)(IN liquid)),

and this case frame has an additional description

(NTRANS (CREAT 'solution ('solvent (*IN))
('solute (*OBJ)))).

The symbol " * " in this description is a LISP function, which retrieves the case element indicated by the argument from the current specialization of the case frame. The sentence,

SHOKUEN 5gr-O MIZU 100ccNI TOKASU ,
salt (object) water (in) dissolve

invokes a specialization of the above case frame as the following, i.e., a new object has been produced, a solution whose solvent is water and whose solute is salt. We put this newly produced object in HNS instead of NS because of the following two reasons.

a) As the description is based on uncertain knowledge, it is likely, but not necessary, that the object is produced in the real world. If we find out some descriptions about this object in the succeeding sentences, we will decide it really exists and transfer it from HNS to NS.

b) Because the newly produced object is referred to in the succeeding sentences sometimes by different words or by syntactically different forms, it is convenient to stack them individually in HNS.

4.3 Estimation of the Omitted Words

In the analysis of a Japanese sentence, it is important to supply the omitted words from the preceding or succeeding sentences. To do this we must be able to, a) recognize that a word is omitted, and b) search for an appropriate word to supply the omitted part.

We think that a certain syntactic unit such as a noun phrase and a simple sentence conveys a definite idea; a noun phrase may designate a certain definite object, a concept, and so on, and a simple sentence may express a definite event. In order that a simple sentence expresses a definite event, each case element of the case frame must be specified by the objects in the sentence. So we can detect an omitted word by finding an unspecified case element in a case frame. Moreover, we can suppose from the case frame what kind of noun should be supplied to the vacancy.

In such a manner, we can detect and fill in an omitted word properly by using the semantic descriptions in the dictionary.

4.3.1 Omitted Word in a Simple Sentence

1 When we have finished the analysis of a simple sentence, we check whether there remain some intrinsic cases to be supplied. If there remain some, we search for appropriate words through the preceding sentences. The searching process is carried out in the following way.

a) We search through HNS first, because the newly created object by the preceding event is often the theme object of the present event.

b) In Japanese, the sentences in succession are apt to omit the same case elements. So we search for the same case in the previous sentence as the omitted case in the present sentence through NS.

c) If the above processes fail, then we check the words in NS one by one until we find out a semantically admissible word or all the words appeared in the three previous sentences.

d) If we cannot find out a suitable one, we set up a problem in the trapping list TL (mentioned in the next section.)

Some results of the processing are shown in Figure 20.

a) Input sentence:

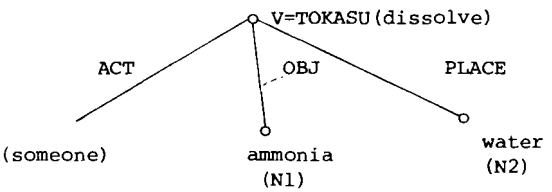
AMMONIA - O MIZU - NI
(ammonia)(OBJ, IOBJ)(water)-(PLACE, TIME, IOBJ, etc.)

TOKASHI, RITOMASUSHI - O TSUKERU
(dissolve, melt)(litmus paper)(OBJ, IOBJ)(soak, put, etc.)

meaning: (Someone) dissolves ammonia in water, and puts a litmus paper (in it).

Analysis process:

result of the analysis of the first sentence



NS= ((N1 N2))
ammonia water

HNS= ((N3))
mixture
solution-N2
solvent-N1

intermediate result of the analysis of the second sentence

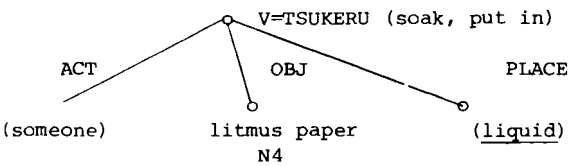
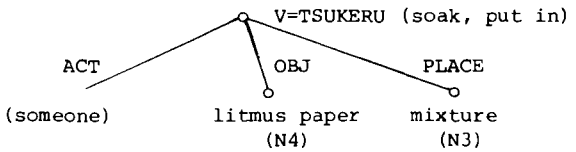


Figure 20 (continued).

final result obtained after searching process



b) Input sentence:

NAPHUTHALINE -O (naphthalene)-(OBJ, IOBJ)
SHIKENKAN - NI (test tube)-(PLACE, IOBJ, IN, etc.)

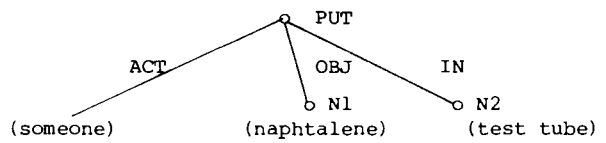
IRE (put in), GAS-BURNER -DE (gas burner)-(INST, METHOD)
NESSHITE, TOKASHI, (heat) (melt)

KANSATSUSURU. (observe)

meaning: (Someone) puts naphthalene in the test tube.
(Someone) heats (it) by a gas burner.
(Someone) melts (the naphtalene).
(Someone) observe (the naphtalene).

Analysis process:

result of the analysis of the first sentence



NS = ((N1 N2)); HNS = NIL

temporary assertion:

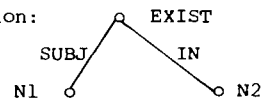
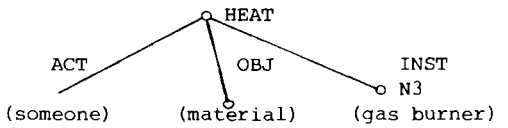
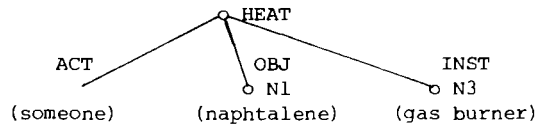


Figure 20 (continued)

intermediate result of the analysis of the second sentence



final result after searching process



NS = ((N1 N3)(N1 N2))

Through the third and fourther sentences have also blank cases, they are properly filled in. The following result is obtained.

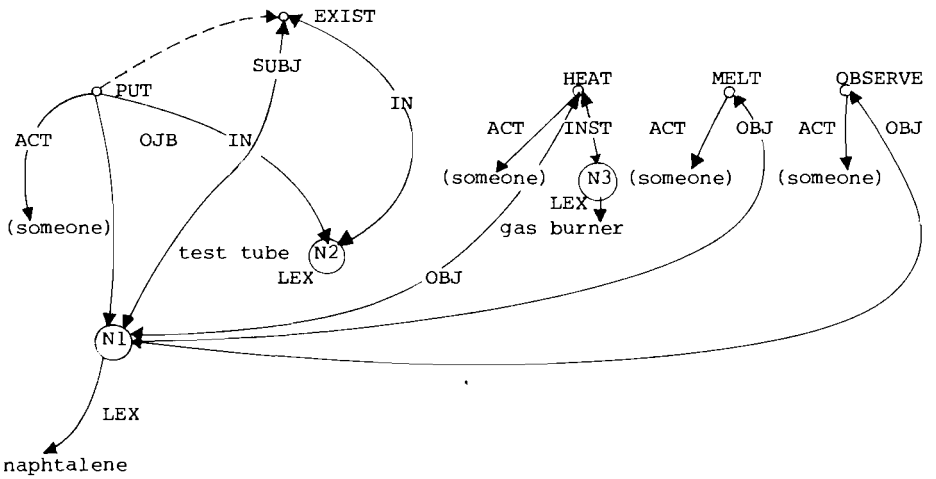


Figure 20 (concluded).

4.3.2 Omitted Word in a Noun Phrase

A noun is classified as either an entity word or a relational word. Most nouns have definite menaing by themselves, and are regarded as entity words. However, some kinds of nouns have relational meaning. That is to say, they have some slots in their meaning to be filled in by other words, in order that they express definite ideas. Sometimes a relational noun is used alone in a noun phrase. In this case, the relational noun must be semantically connected with other words which are omitted in the noun phrase. Such examples are shown in Figure 21.

1) IOU - O NESSURU TOKI IRO - GA HENKASURU.
(sulfur)-(OBJ, IOBJ) (heat) (when) (color)-(SUBJ) (change)

meaning: When (someone) heats sulfur, the color (of sulfur) changes.

The phrase "IRO - GA" is a noun phrase, but it is an incomplete (color) (SUBJ) one by itself. We can easily understand the color mean "the color of the copper."

2) ENSAN SHIKENKAN-NI
(hydrochloric acid)-(OBJ) (test tube)-(PLACE, TIME, etc.)

2Occ IRERU.
(put in)

meaning: (Someone) puts 2Occ of hydrochloric acid in a test tube.

The word 2Occ is put on a separate position from the ENSAN (hydrochloric acid) in the sentence. It, however, specifies the attribute of the acid, VOLUME.

Figure 21.

At the final step of the analysis of a noun phrase, we check whether there remains a relational noun which does not have a definite meaning. If there is, we search through NS for a word which is suitable to fill in the slots of the noun. The searching process is the same as the omitted words in a simple sentence. Sometimes the omitted words exist in the succeeding sentences. So we can set up a problem in TL, if we cannot find out appropriate word in the preceding sentence.

4.3.3 Detailed Description of Trapping List (TL)

Most of anaphoric expressions and omitted words are well analyzed by searching through the preceding sentences. However, we need sometimes to refer to the succeeding sentences in order to analyze a sentence properly. The sentences shown in Figure 22 are such examples. Because the preceding sentences have already been analyzed and both HNS and NS have been set up, it is easy to refer to the preceding sentences. To the contrary, we cannot refer to the succeeding sentences immediately when it is necessary.

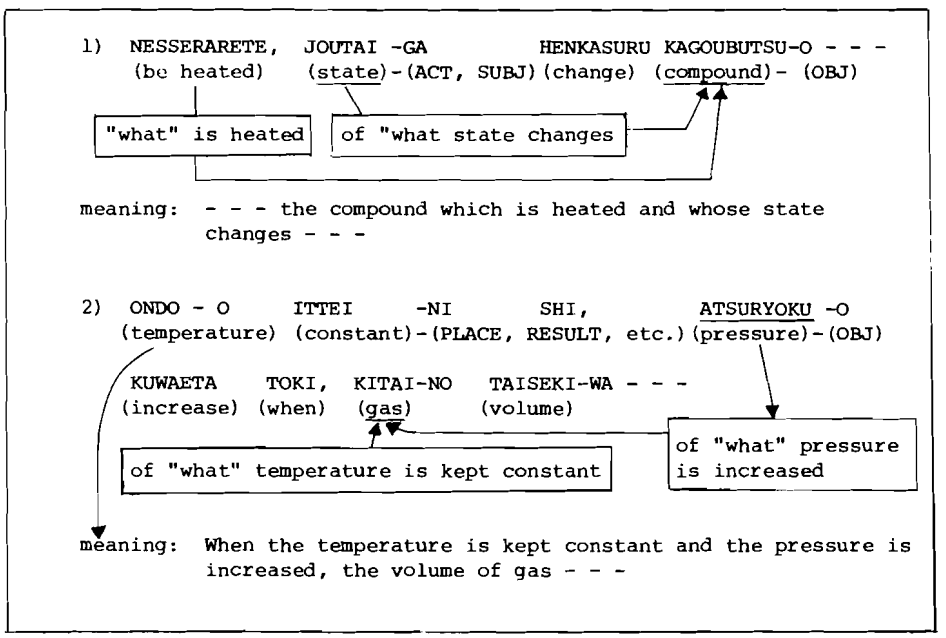


Figure 22. Examples that omitted words appear in the succeding sentence.

To solve this we set up a trapping list TL. The basic organization of TL is shown in Figure 23. A trapping element is a triplet and corresponds to a pending problem. When we cannot find out an appropriate word in the preceding sentences for an omitted word or an anaphoric expression, we put a new trapping element in TL. At this time the first of the triplet, N, is set to zero. When a noun phrase in the succeeding sentences is analyzed, we pick up a noun from the noun phrase

one by one and check whether the present noun can resolve a pending problem in TL by evaluating the function FNI in a trapping element.

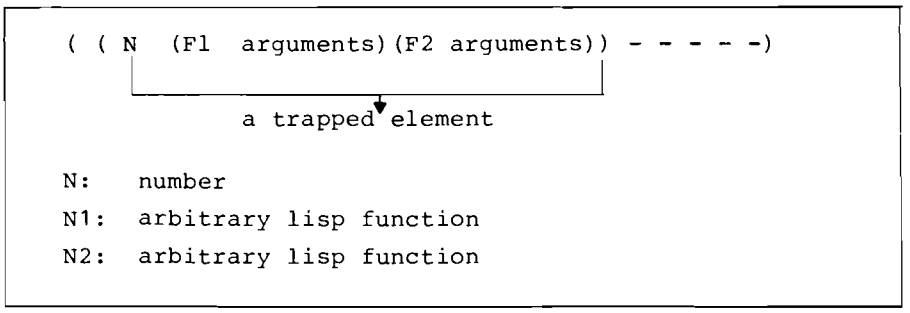


Figure 23. Construction of TL.

We defined several LISP functions for the function FNI. These functions work as follows.

a) They check whether a noun at hand can answer the problem in TL.

b) If it can, they update the data (for example, if the function FNI is the function which searches the words in TL for filling in the omitted case element, then the function will put the present noun in the case frame), and return the value "DELETE." Then the system will delete the trapping element from TL.

c) If it cannot, the system adds 1 to N, the first element of the trapping element. When N exceeds five, the trapping element is deleted from TL. That is, the problem corresponding to the trapping element can not be solved at all. Before the deletion of a trapping element, the third element of it, the function FN2, is evaluated.

By using the idea of TL, we can separate various checking mechanisms from the main program. They can be invoked automatically when a noun appears in a sentence.

4.4 Processing of Anaphoric Expressions

In Japanese, an anaphora is expressed by using the articles "KONO," "KORE," or "KORERA," which correspond to the definite

article "the," "this," or "these" in English. The pronoun "KORE" is used to designate a single object in the preceding sentences, and the pronoun "KORERA" is used to designate plural objects. The article "KONO" is used as a constituent of a noun phrase. Though the article "the" in English modifies the noun directly following the article; "KONO" often modifies a noun far removed from it. An example is shown in Figure 24.

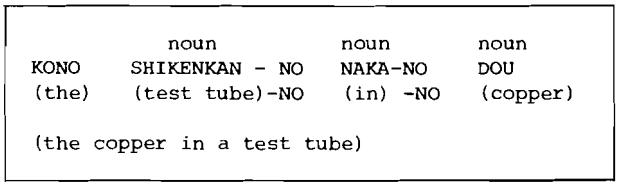


Figure 24.

In this example, three nouns following the article can be modified syntactically by it. We must decide by using contextual information which noun is modified by the article. As stated in the analysis of a noun phrase, we scan the words in a noun phrase one by one from left to right. When we find the article "KONO," we put it in the temporary stack. The word in this temporary stack will be checked as to whether it can modify a noun in the following noun phrase. In the above example, when we scan the noun "SHIKENKAN" (test tube), we check whether the object indicated by it was already mentioned in the preceding sentences. If it was, then the article "KONO" is regarded as modifying the noun "test tube." If not, the article is stacked again. In this way, the article will be check against the nouns in the noun phrase until the noun modified by it is found.

The article "KONO" is used in the following two different ways.

a) SANSO-GA ARU. KONO SANSO-O - - - -
oxygen-(sub. act) exist oxygen-(object)

(There is oxygen.) (The oxygen ------)

The noun "SANSO" modified by the article "KONO" is the same entity noun which is already mentioned in the first sentence.

b) SANSO-GA ARU. KONO TAISEKI-O
volume-(object)

(There is oxygen.) (The volume of the oxygen----)

In this case, "KONO" itself designates that the entity noun "SANSO" appeared in the first sentence. The category of this usage is permitted only if the noun modified by it is a relational one. If the noun has only a relational meaning, the second usage appears more often than the first.

The meaning descriptions of these articles and pronouns are procedurally expressed by LISP functions. The functions in the dictionary will be evaluated if we find such words in a sentence. The function for "KONO" performs its task in the following way.

1) It checks whether a succeeding noun is a relational one. If the noun has only a relational meaning, the function regards at first that the article "KONO" is of the second usage. Go to step (3). If not, go to step (2).

2) The first usage has the following three varieties.

   a) SANSO - GA ARU KONO SANSO-O -----

      (There is oxygen) (The oxygen------)

   The noun modified by the article is the same noun which appeared in the preceding sentence.

   b) SANSO-GA ARU. KONO KITAI-O

      (There is oxygen) (The gas - - - - -)

   The noun "gas" modified by the article is an upper concept noun of the referred noun "oxygen."

   c) SANSO - TO SUISO - O KONGOUSURU.
      (oxygen) (and) hydrogen (obj) mix

      (Someone mixes oxygen and hydrogen.)

      KONO KONGOUKITAI-O ....
           blend gas (obj)

      (The blend gas - - - -)

After the first sentence is analyzed and it instantiates the case frame of the verb "mix," we evaluate the NTRANS description of the case frame and obtain a new inferenced object "mixture," whose element is the oxygen and the hydrogen. The noun "blend" modified by the article is a lower concept noun of the inferenced noun (mixture) in HNS.

According to these three varities, we perform the following three check routines. The order of checking is shown in Figure 25.

check 1: whether there is in the list the same noun as the noun modified by "KONO."

check 2: whether there is in the list the lower concept noun of the noun modified by "KONO."

check 3: when the list is from HNS, whether there is in the list the upper concept noun of the modified noun, and the properties of it are consistent with those of the modified noun.

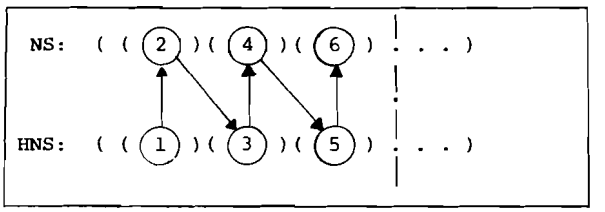


Figure 25.

If we can find out the noun which satisfies one of these three conditions, we decide that it is the referred noun. If we cannot, the function for "KONO" returns the value NIL.

3) If the noun which follows the article has a relational meaning, the meaning description of the noun has slots which must be filled in by other words. What kind of noun is preferable for the slots is described in the meaning description. We search through NS and HNS for the object which satisfies the description.

For example, suppose the input is,

SANSO-GA ARU. KONO TAISEKI - - - .
oxygen (act. subj.) exist volume

The noun "TAISEKI" is an attribute noun. So we look for the noun which may have the attribute and recognize the oxygen is appropriate. Another example is,

SHIKENKAN - GA ARU. KONO NAKA-NI - - - .
test tube (act. subj.) exist in (place, result)

(There is a test tube.) (In the test tube - - -)

The noun "NAKA" (in) is a prepositional noun which requires a "container." We can easily recognize that "test tube" is a lower concept noun of container. Therefore we regard that the word "KON" is used for the test tube. If we cannot find out such nouns, we suppose that the article "KONO" is not of the second usage but of the first. So we will go to step 2.

The pronoun "KORE" (this, it) is used in a sentence as a case element. We can expect the kind of objects designated by the pronoun by using the case frame description of the verb in a sentence. The postposition attached to the pronoun indicates a set of possible cases. By means of taking out from the case frames the cases which belong to the set, we can obtain the semantic descriptions which are satisfied by the object designated by the pronoun. So we search through HNS and NS for the object which satisfies the descriptions. A simple example is,

MIZU	500cc - GA		ARU.	KORE - NI		SHOKUEN
water		(act, subj)	exist		place, result time, - - -	salt

2gr	-O	IRERU .
	(obj)	put in

(There are 500cc water.) (In this water (someone) put in salt of 2 grams.)

The set of possible cases of the postposition "NI" is (PLACE, RESULT, TIME, BENEFICENT,- - - -), and the case frames of "IRERU" (put in) have the case "PLACE." We can expect the pronoun "KORE" (this, it) fills in the PLACE case in the sentence. The semantic description says that a lower concept noun of "container" or "liquid" is preferable as the PLACE case of the verb "IRERU" (put in). The object "water" which is a lower concept noun of "liquid" is found in NS, and is determined to be the object designated by the pronoun.

We have some other pronouns and articles in Japanese which are analyzed in the same way. We provide different LISP functions according to different pronouns and put them in the dictionary definitions of these words.

5. Analysis of Complicated Sentences

In the previous sections, we described the semantic and contextual analysis procedure of our system. In this section, we explain, by using example sentences, how these functional units are organized in order to analyze fairly complicated sentences.

Suppose the input sentence is,

ASSHUKU-SARETE	TAISEKI-GA		HENKA-SURU	TOKI-NO
be compressed	volume (subj. act.)		change	time, when
SANSO-NO	JOUTAI-O	KANSATSUSHI,	SONO	ATSURYOKU-O
oxygen	state (obj)	observe	the its	pressure (obj)
SOKUTEISHI,	SORE-O	GRAPH-NI		ARAWASU.
measure	it (OBJ)	graph (place result)		express.

(Someone observes the state of the oxygen when it is compressed and the volume of it changes, measures the pressure, and expresses it by a graph.)

The sentence is analyzed by the following procedure.

a) The program first tries to find out the leftmost verb, and analyzes the sentence part governed by the verb. The phrase "ASSHUKU-SARETE" (be compressed) is analyzed first. This has an irregular structure in the sense that there are no explicit case elements before the verb. All case elements are omitted in this sentence part. By seeing the inflection of the verb ("ASSHUKU-SURU" (to compress)--"ASSHUKUSARE" (to be compressed)), we recognize that the sentence is expressed in passive voice. The lexical description of the verb in the word dictionary indicates that it takes two intrinsic cases in the field of chemistry, ACTOR and OBJECT. In a Japanese sentence, especially in the field of chemistry, the case element of ACTOR is apt to be neglected. Therefore, we adopt the dummy assignment for the ACTOR as the author of the sentence or some other human being. As there are no preceding sentences, we cannot fill in the OBJECT case immediately. So we set up the pending problem in TL which will watch the analysis of the succeeding sentence part to fill in the case.

b) The sentence part "TAISEKI-GA HENKA-SURU" will be analyzed next. The verb "HENKA-SURU" (change) requires only SUBJ case. The postposition "GA" attached to the noun "TAISEKI" (volume) possibly implies the case "SUBJ." The noun "TAISEKI" is a lower concept noun of "attribute," which satisfies the semantic condition of the case element. So this sentence is analyzed in a straightforward manner. However, because the noun "TAISEKI" is an attribute noun, we must find out the entity noun which corresponds to the noun "TAISEKI." That is, we must identify the object whose volume is meant by the word. As we cannot find out such an object in the preceding sentences, we set up a pending problem in TL. By

checking the inflection of the verb "HENKASURU" (change) and the word order "--- verb + noun ---." it is syntactically recognized that this sentence is an embedded sentence and modifies the noun "TOKI" (time, when). We then connect this sentence part with the noun "TOKI" by using the relation "SMOD" (MODified by a Sentence).

c) When we analyze the next sentence part,

TOKI - NO SANSO-NO JOUTAI-O KANSATSU-SURU
time oxygen-(of) state-(obj) observe
when - (of)

we first perform the analysis of the noun phrase "TOKI-NO SANSO-NO JOUTAI." The combination of the two nouns "TOKI" and "SANSO" is semantically permissible because "oxygen" is a lower concept noun of "material," and can be modified by a word which designates a special point of time. The noun "TOKI" is modified by the sentence part analyzed at step (b), and designates the time when the event expressed by the sentence part occurs. The combination of "SANSO" (oxygen) and "JOUTAI" (state) is also permissible.

The nouns "TOKI," "SANSO," and "JOUTAI" in the noun phrase activate the trapping elements in TL. The noun "SANSO" satisfies the conditions of the two trapping elements set up by step (a) and (b). That is, "SANSO" fills in the case OBJ of the first sentence part. "TAISEKI" (volume) in the second sentence is regarded as the volume of the oxygen.

d) The next sentence part "ATSURYOKU-O SOKUTEISHI" can be processed easily. However the noun "ATSURYOKU" (pressure) is used alone without "of what." We must find out the corresponding entity noun in the preceding sentences. "Oxygen" is easily found to satisfy the condition, and "ATSURYOKU" (pressure) means "of the oxygen."

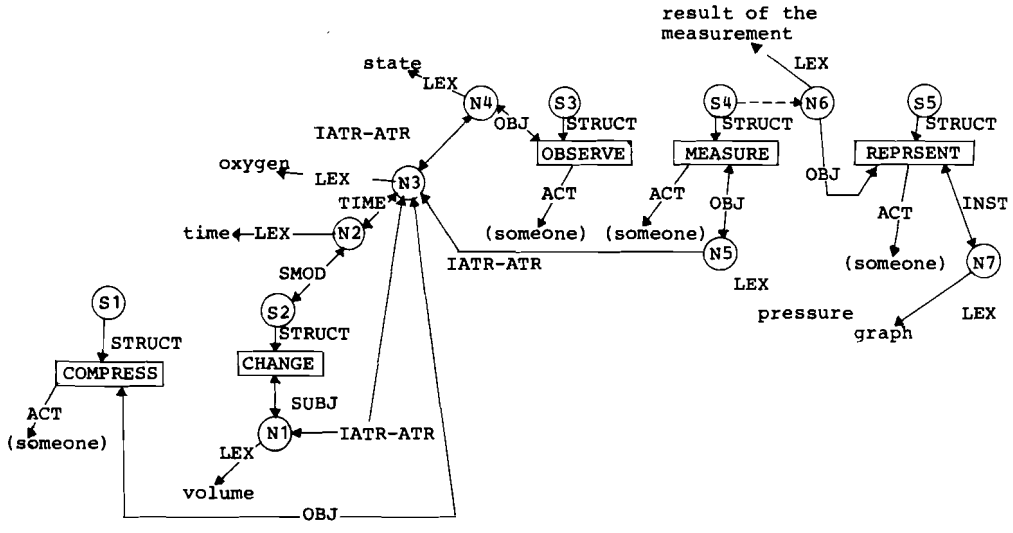
e) The remaining steps will be easily understood. We show the result of the parsing of the whole expression in Figure 26.

The next example shows how HNS is used. Suppose the input sentence is,

SUISO - TO SANSO -O KONGOUSHI, KONO KONGOUKITAI-NI
hydrogen-(and) oxygen-(obj) mix the blend gas

TENKASURU-TO BAKUHATSU-SHI, MIZU-GA DEKIRU.
fire (if, when) explode (water (subj, act)be made

(If someone mixes hydrogen and oxygen, and fires the blend gas, then (it) explodes and water is made.)



Input sentence:

ASSHUKUSARETE, TAISEKI - GA HENKASURU TOKI -NO
(be compressed) (volume)-(ACT, SUBJ) (change) (when)

SANSO - NO JOUTAI - O KANSATSUSHI, SONO ATSURYOKU-O
(oxygen) (state)-(OBJ) (observe) (the) (pressure)-(OBJ)

SOKUTEISHI, SORE-O GRAPH-NI SURU.
(measure) (it)-(OBJ) (graph)-(IOBJ, RESULT, etc.) (represent)

meaning: (Someone) observes the state of oxygen which is compressed and whose volume changes. (Someone) measures the pressure and represents it as a graph.

Figure 26.

The following steps are performed.

a) When the analysis of the first sentence part, "SUISO-TO SANSO-O KONGOUSHI" is finished, the case frames of the verb "KONGOUSHI" are instantiated. We will evaluate the NTRANS expression of the case frame which obtains the highest matched value. As the result, a new object "mixture" is created and the elements of the mixture are the hydrogen and the oxygen. This newly created object is put into HNS.

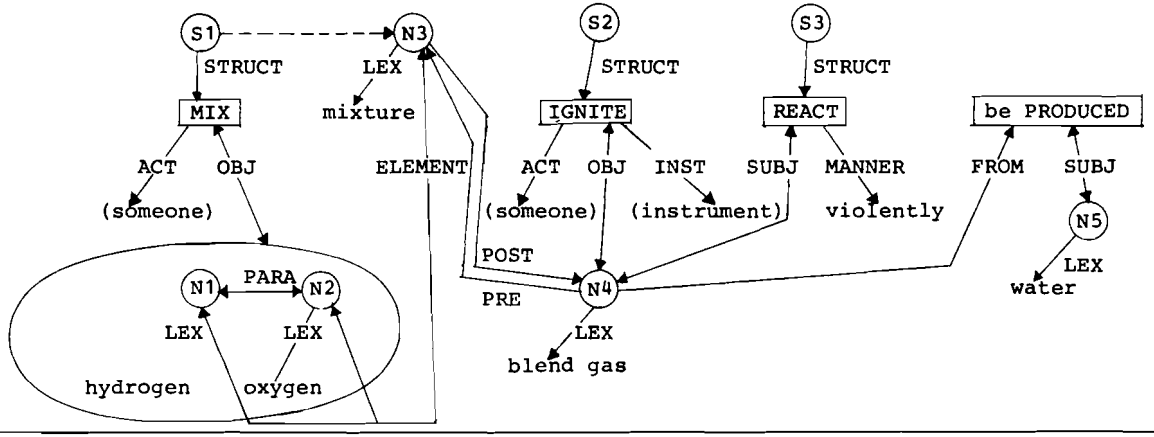
b) Because the noun phrase "KONO KONGOUKITAI-NI" (to the blend gas) in the sentence part is modified by the anaphoric determiner "KONO" (the), it requires the object which is designated by this phrase. The noun "KONGOUKITAI" (blend gas) is a lower concept noun of "mixture," and the components of it are gaseous objects. We search for it in HNS and NS and find the object "mixture" in HNS, whose elements are the hydrogen and the oxygen.

c) The object "blend gas" is the theme object in the succeeding sentences. It fills in the omitted case ACT of the third sentence part and FROM case of the fourth sentence. Figure 27 shows the result of the parsing.

Table 6 shows the score of the results obtained by applying our analysis scheme to the sentences in a textbook of junior high school chemistry.

Table 6. Score of result.

	Total number	Success	Failure
Noun phrase	312	286	26
Conjunctive phrase	372	349	23
Sentence	280	254	26



Input sentence:

SUISO - TO SANSO - O KONGOUSHI
(hydrogen)-(conjunctive pp - - -and) (oxygen)-(OBJ) (mix)

KONO KONGOUKITAI-NI TENKASURU -TO
(this) (blend gas)-(OBJ, IOBJ, PLACE, etc.) (ignite)-(conjunctive pp
- - - if, when)

HAGESHIKU KAGAKUHENKASHI, MIZU - GA DEKIRU.
(violently) (react) (water)-(ACT, SUBJ) (be produced)

meaning: If (someone) mixes hydrogen and oxygen and ignites it,
then the gas violently reacts and water is produced.

Figure 27.

6. Conclusion

We can summarize the above procedure of our language analysis in the following way.

a) By introducing the idea of case, we described the patterns of activities in the verb dictionary. The descriptions also contain the information as to how activities are connected with each other and how an activity changes the objects.

b) The meaning descriptions of nouns are based upon the upper and lower concept relationships and the attribute value pairs. Some kinds of nouns are regarded as having relational meanings. Their meaning descriptions are similar to those of verbs, adjectives, and prepositions. By using these descriptions, we can analyze fairly complicated noun phrases where there are few syntactic clues.

c) We do not use logical expressions to represent context. Instead, we represent contextual information in the memory structure similar to that of human intermediate term memory. The combination of it with the semantic descriptions of words has enabled us to perform efficient contextual analyses.

d) We have developed a programming language which makes it easy to write natural language grammar and to control the analysis procedure. By using this language, we can incorporate semantic and contextual analyses with semantic ones. Semantic and contextual checking functions are rather simple and small.

Our analysis program has obtained a fairly good result. However, the contextual analysis program can treat only a local context. In order to treat a more global one, we should improve our program in the following ways.

a) Corresponding to human long-term memory, we must provide our system with an appropriate scheme to represent the state of the world. The system must have frameworks to express spacial relationships among objects, time relationships among events, and so on.

b) At the present stage, we have only one relationship "CON" to connect one activity with others. However, human knowledge of the world contains various kinds of relationships among activities, such as cause, purpose, reason, and so on. These relationships may play an important role not only in the analysis process of sentences, but also in the inference process in answering a question.

c) In order that a system can communicate with a man in a flexible and natural manner, it must be able to perform inferences from incomplete data bases. Therefore, we will not use a uniform proof procedure such as the resolution proof procedure.

d) It is necessary to apply our method on fields other than chemistry, and to test whether our semantic description method should be changed or not.

References

[1] Charniak, E. "Toward a Model of Children's Story Comprehension." Tech. Rep. MAC TR-266, Massachusetts Institute of Technology, 1972.

[2] Fillmore, C. "The Case for Case." In Bach and Harms, eds., Universals in Linguistic Theory. New York, Holt, Rhinehart, 1968.

[3] Hunt, E. "The Memory We Must Have." In Schank and Colby, eds., Computer Models of Thought and Language, 1973.

[4] Martin, W. "A System for Building Expert Problem Solving Systems Involving Verbal Reasoning." OWL Notes, Massachusetts Institute of Technology, 1974.

[5] Sandwall, E. "PCF-2, A First-Order Calculus for Expressing Conceptual Information." Computer Science Report of Uppsala Univ., 1972.

[6] Schank, R.C. "Margie: Memory, Analysis, Response, Generation, and Inference on English." Proc. 3rd IJCAI, 1973.

[7] Simmons, R. "Semantic Networks: Their Computation and Use for Understanding English Sentences." In Schank and Colby, eds., Computer Models of Thought and Language, 1973.

[8] Wilks, Y. "Understanding without Proofs." Proc. 3rd IJCAI, 1973.

[9] Winograd, T. "Procedures as Representation for Data in a Computer Program for Understanding Natural Language." Tech. Rep. MAC TR-84, Massachusetts Institute of Technology, 1971.

Parsing in QAS

W.H. Paxton

The activity of the parsing system can be described as the step-by-step construction of "interpretations" of utterances. An interpretation is a phrase of the root category of the language that spans the utterance and includes attributes such as semantic representation. Phrases are created either by (a) recognizing a word in the input, or (b) applying a composition rule to constituent phrases. In the parser's search for an appropriate interpretation, phrases are incrementally formed, evaluated and combined. As this process goes on, the parser builds a data structure called the parse net, representing the growing collection of phrases, and maintains another structure, called the task queue, encoding the alternative operations available for taking another step toward understanding the input. Each entry in the task queue specifies a procedure to be performed at a particular location (node) in the parse net. The performance of such a procedure typically entails both modifying the parse net and scheduling new tasks to make further modifications. By factoring the parsing process into tasks that first make incremental changes and then spawn other tasks to be performed at unspecified later times, the parser is given a means of controlling the overall activity of the understanding system. Other components of the system, such as semantics, may carry out large portions of a task, but it is the responsibility of the parser to decide when the task will actually be performed. Thus, instead of having a separate "control" component in the system, decisions regarding what to do next are made by the parser on the basis of a complex, heuristic parsing strategy described at length below.

The control aspect of the parser's role is of great importance because only a subset of the scheduled tasks will actually prove to be necessary to understand the input; the others will be "false steps" leading toward potential interpretations but proving to be inappropriate for the particular utterance being parsed. Ideally, in deciding which task to perform next, the parser would always choose one of the necessary tasks and never take a false step. The utterance would be understood with the unnecessary tasks still left in the queue. To approach this ideal, the actual system must spend some of its effort deciding which task to perform next. Such effort is well spent if it produces a net decrease in processing time. In other words, the efficiency of the system will be improved by decisions regarding the order in which tasks are performed if the cost of the decisions is less than the cost of the false-step tasks that would have otherwise been performed. Since the

potential for wasting effort on unnecessary operations is particularly large in natural language understanding, the system can afford to carry out rather complex computations in deciding what to do next, and still get a big improvement in overall efficiency. In the current system, the decisions are based on the relative priorities assigned to the various tasks waiting in the queue.

In establishing priorities, the parser gets important guidance from the "values" the language definition assigns to different interpretations. In addition to defining the possible phrases, the language definition also associates with each phrase a set of factors to be used in establishing its score with respect to particular inputs and contexts. In particular, each interpretation, being a root category phrase, gets a score in this manner. The interpretation value is a simple function of this root score. Other things being equal, a task will be favored if it appears to lead toward an interpretation with a higher value. To achieve this ranking, task priorities assigned by the parser tend to reflect the maximum value of the interpretations whose construction the task would lead to.

In addition to interpretation value, response time is also an important concern. The parser must balance the goal of finding the interpretation with the highest value against the goal of making a prompt response. Our approach in dealing with these conflicting goals is to maintain a set of phrases, in the parser called focus phrases, that have been constructed in the parse and to concentrate on finding ways to extend them to a complete interpretation. This focusing of activity is brought about by inhibiting tasks looking for replacements for any of the focus phrases, unless the potential replacement promises to lead to a significant improvement in value for the final interpretation. Tasks conflicting with the focus of activity have their priority temporarily lowered so that the parser is biased toward building up a complete interpretation using phrases in focus rather than exploring competing interpretations that would not use focus phrases. If the focus is wrong, then the attempts to extend it to a complete interpretation will be unsuccessful. Eventually a task that conflicts with the focus will become the highest priority operation for the parser to perform in spite of the bias against it. As a result, the focus set will be modified so that it is consistent with the new task, and the parser will then concentrate on using the revised set of phrases.

In addition to calculating priorities of tasks on the basis of interpretation values and focus of activity, the parser must ensure that the information gained through the performance of the tasks is used effectively. This is done by structuring the parse net and the tasks that operate on it in a way that brings together related activities and coordinates them to eliminate duplication of effort. By avoiding duplication, the system reduces the ill effects of the false steps it will inevitably take. Work done on a false path is not necessarily wasted, since

it may produce a phrase that can be used in some other way. For example, a phrase constructed as part of an unsuccessful search for one type of sentence may later appear in the final interpretation as part of different kind of sentence. Also, false steps are not repeated, since the system only makes one attempt to build a particular type of phrase in a particular location in the utterance, regardless of how many large phrases might include it. Mistakes are inevitable, but at least the system will not make the same mistake twice in one parse.

To summarize, the parser balances the desire to find the highest value interpretation of an utterance against the need to make a prompt response. In a step-by-step manner, phrases are created, evaluated, and combined. The choice of the next operation to carry out takes the form of assigning priorities to alternative tasks. Priorities reject both the expected values of interpretations toward which the task would lead and the relation of the task to the current focus of activity. Finally, the entire process is organized so that information gained in performing a task is shared and recorded in such a way that it does not have to be rediscovered.

Acknowledgment

This report has been extracted from D.E. Walker et al., Speech Understanding Research, Annual Report, Project 3804, Artificial Intelligence Center, SRI, Menlo Park, California,1975. Work reported herein was supported at SRI by the Advanced Research Projects Agency.

Input Processing in a German Language Question-Answering System

Egbert Lehmann

1. Introduction

For fact retrieval, it seems desirable to develop a truly multipurpose question-answering system (QAS) accepting as input written natural language text. By providing different factual data bases and dictionaries to such a system, it could be adapted to satisfy the needs of very different groups of users. User-questions and also a great part of the stored factual information (including, if necessary, pragmatic information and definitions of field-specific notions) should be given to the system in a uniform and convenient manner, at least very similar to a natural language. The system should be able to provide the user, on request, the factual information he needs.

Because ultimately a lot of knowledge must be incorporated in such a system, which may become enormously complex, an evolutionary approach starting with a minimum of linguistic and world knowledge seems appropriate in developing it. Development (design, coding, testing, debugging) of a carefully predefined general knowledge base as well as an application-oriented data base is a formidable task. It can be considerably simplified if the QAS is already principally able to work in a simple manner when only a very limited knowledge is available. It has to understand literally the meaning of ordinary, not too complicated sentences (at the risk of sometimes seeming a little stupid in doing so!). In this way, the system would be able--by storing facts and finding answers to posed questions by inference from stored facts--step by step to extend and adapt its knowledge base. So, mainly by its own experiences, it would in time become more and more qualified. Working with the system in an experimental way can be extremely valuable for the designer because he gains a vivid impression from the actual working capabilities and shortcomings of the system at each stage of development. I considered evolutionary principles important in designing the input language and the language processor of our QAS. The program for input analysis was programed by the author in LISP and has been running on a RJAD R-40 computer with 512K byte core memory. It is part of a complete German language question answering system. The ideas of Sandewall, Palme, Woods, Winograd, Schank, and Simmons have strongly influenced this work.

2. The Input Language: An Evolutionary Approach for Developing the Input Language

Because in the beginning the input processor has only a very poor semantic and factual knowledge, is has to rely strongly on syntactic considerations. First, in the spirit of Montague (1970), a simple but sufficiently powerful formal language was developed, which corresponds to a certain extent to the German language and constitutes a possible skeleton of it. This so-called core language has a rather small vocabulary of structural words or particles (comparable to the reserved symbols of a programing or logical language). These words are of fundamental importance for language analysis and interpretation and must be known absolutely to the language processor.

This strictly formal language layout decreases the risk of misunderstanding (which otherwise would be very large in absence of a voluminous knowlege base) and permits a straightforward, very efficient process of language analysis. Our core language is comparable to and of the same level as the language of Sandewall's (1972) PFC-2.

By successive extensions of the grammar, the dictionary, and the parser, we then tried to make our input language more and more similar to the German language. We obtained what we call a stylized written German language. By continuing this effort, it ultimately seems possible to cover the full natural language. In comparison with the German language, our sylized fact input language FES (Fakten-Eingabe-Sprache) has some not so severe restrictions (concerning some complicated or extravagant language constructs, which cause difficulties in language processing) and a few additional language elements for avoiding lexical or syntactic ambiguities (for instance, parentheses for explicit structuring of complex subordinated sentences or complex noun phrases). All these extensions and restrictions together only lead to a moderate deviation from German. So the input language FES remains comprehensible to a native speaker/listener of German and has nearly the same expressive power. Imbedded in the FES is the more restricted and artificial looking core language, which is not only easier to implement, but is also valuable in many communicative situations for avioding ambiguities. Because both languages can be intermingled, there are, at the disposal of the user, very flexible language tools covering a wide range between the core language and nearly the full natural language.

3. Elements of the Core Language

Most German content words used in sentences of the core language are unknown to the language processor. Therefore, the user has to mark the lexical category of word class of each unknown word by putting one of the following reserved artificial morphemes in front of it:

: (for proper names of individuals),

V (for infinitives of verbs),

* (for nouns characterizing countable objects),

+ (for nouns characterizing substances (mass terms)),

/ (for adverbs and adjectives),

F (for substantives with a functional meaning, as "father," "capital"),

R (for substantives denoting relations, as "part," "property").

All German concept words are used in the core language only in the basic form; inflected forms are not permitted. The frequently used particles are very important for understanding the core language. They are mostly adopted from German, and they are partly designated by a modified or artificial name. In the syntactic position of determiners, we distinguish the following five particles:

D (analogous to the English "the"), definite determiner,

E (English "a"), specific indefinite determiner,

C (not explicitely available in German and English), acts as a placeholder in NP's that describe concepts instead of individuals),

IRG ("any"), unspecific determiner; quantifier, producing existential quantified variables, and

JED ("each"), quantifier, producing universal quantified variables.

Semantically, "E" introduces new individuals in the semantic representation; "D" is very important as a frequently used reference-establishing mechanism, crossing the sentence boundaries in the discourse to refer back to a previously mentioned (or known) individual. All determiners can be considered formally as functions acting on concept descriptions. The basic auxiliaries are the particles:

IST (English "is"), the basic auxiliary expressing predication by attaching concepts (properties) to individual objects (e.g., "Peter is teacher," "the girl is pretty"),

TUT (engl. "does"), the basic auxiliary for describing events, and

PASS (describing events in passive mood.)

The basic model auxiliaries are: KANN ("can"), WILL ("will"), SOLL ("shall"), DARF ("is allowed"), MUSS ("must"), MAG ("may"). Prepositions in the core language mainly characterize, in a nonambiguous way, deep-case relations, e.g.,

MIT ("with"), the INSTRument;

IN ("in"), AUF ("on"), AN ("at"), the LOCation;

NACH ("to"), the DIRection,

BEI the CIRCumstances, and

UEBER the THEMA of an action.

This set of German prepositions is extended by modified preposition names (e.g., ALS-T (SIMULTaneous), NACH-T ("after")) and by the original names of the appropriate deep-case relations (e.g., INSTR, LOC, DIR, CIRC, THEMA as well as AG, OBJ, DAT, etc.) appearing in the semantic representation. Conjunctions and interrogative particles also characterize deep-case relations. Numerals always are written as ordinary numbers.

In the core language, there already are a few pronouns, which are coreferent in a simple way with language constructs mentioned before (usual NP's):

DIESØ ("this), is coreferent with the whole preceding sentence,

ERØ ("it"), is coreferent with the first NP on the top level of the preceding sentence (normally the grammatical subject of this sentence),

DØ is coreferent with the last NP on the top level of the preceding sentence,

D1 is a kind of relative pronoun refering to the NP appearing immediately left of it,

DORT ("there"), is coreferent with the last location mentioned in the preceding text,

DANACH ("then"), establishes a relation of temporal order between the time moments of both events described in the actual and in the preceding sentence.

SICH ("oneself"), is coreferent with the first NP (subject) of the same sentence it appears in, if it is not dominated by a reflexive verb.

Moreover, there is the pronoun ESØ (unpersonal "it"), which is only a syntactic placeholder for the subject position in a sentence. MAN or JEMAND ("someone," for persons), and ETWAS ("something," for impersonal entities) with the corresponding

sentence-negating forms NIEMAND ("nobody") and NICHTS ("nothing") are unspecific (variable-generating) pronouns.

Some unique artificial morphemes are used for explicit characterizing of some special grammatical phenomena:

PLUR (plural of nouns, appropriate only for compound individuals, i.e., finite sets of single individuals),

PRAET (past tense of verbs),

FUT (future tense),

KONJTV (subjunctive mood)

INF (infinitive constructions),

NICHT ("not", negation).

Because it is not possible here to describe the core language in greater detail, some additional remarks and an example must suffice for the moment. (For a formal description of the syntax see Figure 1, in section 8, page 124.

An input text is a sequence of main sentences. A main sentence may be a prepositional sentence, an open question, a closed question, or a command. Subordinate sentences (subsentences) appear normally at the end of other (main or subordinate) sentences. They can be recursive up to an arbitrary depth. Subsentences, which are part of other subsentences, are surrounded by a pair of parentheses in order to facilitate the interpretation of the whole sentence. When more than one noun phrase (NP), not preceded by prepositions appears on the top level of an arbitrary sentence, normally the first one will be considered as describing the syntactic SUBject, the last one the direct OBJect, and the middle one (if available) the DATive (indirect object) case of the sentence.

4. Syntax of the Core Language

```
Text ::=  (main sentence...)

main sentence ::= ({[praep]{np|?w|?welch subst}aufz|aufz np|
                                                    |BITTE}[NICHT]
                    {[praep]{np|advp}}...[vi...|praep]
                     [,{ss|infc}]...{.|?|!})

ss ::= subkonj{[praep]{np|advp}}...[vi]...aux[({ss{infc})]...

infc :: = [{UM|INF}]{[praep]{np|advp}}...[vi]...ZU vi
                  [({ss|infc})]...
```

```
np ::= proper name|refword|{det|number}[PLUR][adjp]...{subst|vi}|
       [substanzmod|det|number{unitofmeas|subst}][adjp]...substanz|
       <np-{[praep]np|rels}>

rels ::= ([praep]relpron{[praep]{np|advp}}...[vi]...aux
         [({ss|infc})]...)

adjp ::= [grad|quant]adj

advp ::= [grad|quant]adv

quant ::= number unitofmeasure
```

Meaning of the syntactic metavariables (written with small letters, angle brackets are avoided): ss = subordinated sentence introduced by a conjunction, infc = infinitive construction, np = nominal phrase, rels = relative sentence, adjp = adjective phrase, advp = adverbial phrase, quant = quantity, praep = prep, aux = auxiliary; ?w = interogative0 (as "who", "where", "when"), ? welch = interogative 1 (as "which", "how many"), vi = verb infinitive, subkonj = subordinating conjunction, refword = referential word, det = determiner, subst = substantive (count term), substanz = substantive (mass term), substanzmod = modifier for portions of substances (as "much"), unitofmeas = unit of measurement, relproun = relative pronoun, grad = gradual modifier of adjectives/adverbs (as "very").

As metalanguage, we utilize a BNF notation extended by square brackets (surrounding facultative consituents), special parentheses "{" and "}" for factorization of substrings and for the Symbol "...", describing the possibility of iterated appearance of the constituent left of it. Look, for example, at the following compound sentence, reexpressing the German sentence, "Columbus glaubte, dass er Indien erreicht hätte, als er 1492 Amerika entdeckte." ("Columbus believed that he had arrived in India, when he discovered America in 1492.")

```
:  COLUMBUS TUT PRAET V GLAUBEN . DASS ERØ : INDIEN
   Columbus does      believe   that he    India

V  ERREICHEN PRAET KONJTV TUT . ALS-T ERØ : AMERIKA
   arrive               does  when he    America

MOM 1492 V ENTDECKEN PRAET TUT .
           discover        does
```

The reserved words of the core language are underlined in this example.

Complex NP's consisting of other NP's also are allowed. Their recursive structure is explicitly expressed by use of angle brackets

and by inserting a dash behind the first NP-constituent:

```
np ::=  < np - rels > | < np - [praep] np >  .
```

Apparent by this notation, the greatest source of ambiguity in German sentences is removed. For instance, the core language expression:

```
< D F ANZAHL - < D * EINWOHNER - ( D1 MOM 1964
  the number    the  inhabitant   that

IN < D F HAUPTSTADT - VON : ÖSTERREICH > V LEBEN
in  the  capital     of    Austria        live

PRAET TUN ) > >  ,
```

reexpresses the German NP, "die Anzahl der Einwohner, die 1964 in der Hauptstadt von Österreich lebten" ("the number of inhabitants living in the capital of Austria in 1964").

5. The Stylized Fact Input Language FES

If we proceed from the artificial core language to the stylized natural language, many new problems appear. Ambiguous syntactic constructions will become possible if some grammar rules are relaxed (e.g., by deleting some artificial language elements used in the core language for explicit structuring) and many new rules are added. Therefore, a backtracking mechanism will be needed for input analysis. Semantic ambiguities also appear by use of homonymous words. Therefore, it is important to distinguish different word meanings of homographs in the dictionary. For inflected word forms (in German more multiform and frequent than in English), the language processor has to find the basic word form with the aid of the dictionary and eventually also find procedures for morphological analysis.

Because the vocabulary of a natural language grows with time and therefore is considered as potentially unlimited, we need mechanisms for handling words, so far unknown to the system, at least in a preliminary and inductive way. Sentences containing such words must be understood by the system, and the position of the unknown word in the sentence and its morphological properties must be observed.

Mechanisms for resolving anaphoric references are of great importance for a language for the description of arbitrary facts. More sophisticated devices of backward reference not only enable the language user to formulate his ideas in a shorter and more economical way, but are also necessary to express the unknown word, so that the same object, which was not identified previously by a proper name, is indicated by descriptions in different places in the text or discourse.

The following fragment of a simple text in FES may serve as an illustration. We have (a) the text in FES for processing with an extended dictionary, (b) the English translation, and (c) the text translated into the core language):

1.

(a) WIEN IST DIE HAUPTSTADT VON ÖSTERREICH.

(b) Vienna is the capital of austria.

(c) : WIEN IST C F HAUPTSTADT - VON : ÖSTERREICH.

2.

(a) ES LIEGT AN DER DONAU UND IST EINE SEHR SCHÖNE STADT.

(b) It is located on the Danube and is a very beautiful city.

(c) ERØ TUT AN : DONAU V LIEGEN.

ERØ IST C SEHR / SCHÖN * STADT.

3.

(a) IN WIEN GIBT ES EINE ALTE UNIVERSITÄT - (DIE

(b) In Vienna there is an old university, which

(c) < E / ALT * UNIVERSITÄT - (D1 MOM 1365 V GRÜNDEN

(a) 1365 GEGRÜNDET WURDE).

(b) was founded in 1365.

(c) PRAET PASS) TUT IN : WIEN V EXISTIEREN.

4.

(a) WANN WURDE DIE UNIVERSITÄT - IN WIEN GEGRÜNDET?

(b) When was the university in Vienna founded?

(c) WANN PASS PRAET < D * UNIVERSITÄT - IN : WIEN > V GRÜNDEN?

6. The Semantic Representation

General principles

Before we begin to design special procedures for input processing, we have to set up general principles for the (essential language-free) semantic representation of the content of factual information. Before we ask how to represent and implement the conceptual meaning of texts, we first have to ask

what to represent at all. This means, we have to reflect on the diverse nature of the parts of reality described by natural language and to classify them along general lines. In doing so, our thoughts were strongly formed by the valuable work of Sandewall (1972) and Schank (1972). We believe that, in a semantic description, objects of the physical world as well as concrete situations and events and also arbitrary complex conceptual structures, characterizing classes of each kind, should be explicitly representable. This makes possible the formal description of all the different interrelations among and properties of the phenomena dealt with in a discourse.

Our most important distinction of these phenomena is between individuals and concepts. As individuals, we consider not only things and persons but, in an extended sense, also each kind of particular; entities existing in reality or in the human imagination. Special portions of substances, locations (places), and ensembles (consisting of several single individuals), institutions, particular situations, and events, and also objects of human thought or feeling mostly will be handled as individuals. Concepts are essentially abstract in nature. They are used as instruments of human cognition for grasping and ordering the parts of the reality by establishing their properties and interactions. Most things (and also most concepts in use) are not designatable by proper names (or other single words of a language), but must be described with the aid of more complex language constructs by use of concept words. Logical interpretation of concept words as names of predicates or relations is obvious and can be consequently done if the notion of individuals is used in an extended sense as outlined above. The same should be also true of more complex concepts. Concepts can (analogously as individuals) be classified to characterize classes of individuals (as countable objects, situations, events, etc.) as well as functions, relations, properties of arbitrary phenomena, numbers and measurable quantities.

For modeling and analyzing semantic structures, we tried to set up a rather small (but not minimal and thereby too unspecific) set of basic semantic relations. Deciding what sort of methodological framework should be used for the description and/or implementation of the semantic representation of natural language discourse was only the second step. Nowadays, candidates for such a framework are:

a) the clean and academic looking world of first order, predicate calculus strongly connected with resolution-oriented theorem proving,

b) the action- and heuristic-oriented world of high-level languages of artificial intelligence such as PLANNER and QA.4 (respectively, QLISP) where all can be programed in a pragmatic, sometimes rather violent, way and where nothing is impossible,

c) the more restricted and humanlike, intuitive, appealing, and efficiently manageable world of semantic networks.

The last of these was selected as a basis of our semantic representation for input texts.

I shall now describe what the available program actually produces as semantic representation, and not what it eventually will produce in the future. I am conscious that many important problems are open at present. Other problems seem to be understood from a theoretical point of view, but troubles arise in the practical application of some of the better ideas, because the general knowledge of the system is too restricted yet. Often it proves difficult to devise a satisfactory mechanism for extracting, from the given natural language text, all information necessary for a good meaningful representation. The difficulties only increase with the often diffuse, obscure, and fragmentary traits of human thought and language use.

For example, if we look at verbs as concepts usually representing classes of events or actions, it seems impossible to define how many (deep-case) arguments each verb must have and of what special sorts its arguments should be. Undoubtedly, such definitions are possible and useful within a fixed, rather limited universe of discourse such like Winograd's (1972) block world, or in dealing with texts describing vital actions of human actors in common situations, principally known to all human readers/listeners (as analyzed by Schank (1972, 1974)). But for a more general context (including, for example, the widely used jargon of scientific writing, and also newspapers which use many common words in a less literal sense), too severe restrictions on the use of words would be unwise and unrealistic.

The semantic representation of our input language will not be different from that of ordinary German and will also be essentially similar to that of other natural languages. Because the application area of our QAS will be the general area of fact retrieval, our input language and its semantic representation are focused on particular true facts, not on arbitrarily structured general prepositions. The semantic network paradigm seems to be a reasonable representational framework for such facts.

Our semantic network is a twofold labeled directed graph. Its nodes represent individuals and concepts that can both be subclassified into different sorts. The arcs of the network are labeled by the names of some basic two-place semantic relations, which we consider to be the most significant relations for human thought. More specific two-place relations as well as one-place relations, three-place relations, etc., usually will be represented by more complex network configurations, using concept nodes as representatives for such relations.

Our network stored in the computer is symmetrical insofar as for each stored link between two nodes n1 and n2 there also exists an (inverse) link between n2 and n1 in the network. Both links are labeled by the name of the same relation, but the first label is marked outgoing and the second incoming.

From a purely logical point of view, our network was a special schema for the efficient storage and retrieval of large masses of propositions in a heavily restricted predicate calculus (without variables and quantifier and without logical connectives different from the logical "and" ("$\wedge$") and the negation sign on the lowest level. In spite of its simplicity and small expressive power, this schema proved to be almost sufficient as a base for logical interpretation of simple facts. Recently, the expressive power was significantly increased by adding variables as well as some constructions of a higher-order logic to our network formalism. However, until now we have no fully general method for representing first order, predicate calculus formulas as network structures.

7. Semantic Relations

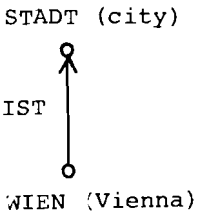
In selecting a limited number of basic semantic relations, we freely used or borrowed names and/or meanings of relations also used by other authors (for example by Simmons (1972), Sandewall (1972), Palme (1973), Rumelhard (1973)). To begin with, there is a single relation, IST ("is"), defined between individuals and concepts that acts as the usual device for predication. If x is an individual and P a concept, then IST (x,P) means that $\bar{P}$ (x) is true ("x has the property P").

IST (Vienna, city) means that an individual called "Vienna" is a member of the class of "cities."

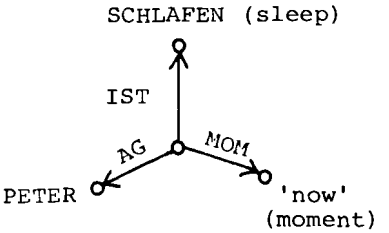
IST ("Peter is sleeping now", sleeping) means that the so described particular event is a member of the class of "sleeping-acts" (that are named by the verb "sleeping" or "sleep").

If a predication is not really true but only believed or supposed to hold, the relation IST has to be replaced by the relation SEI ("may be"). The negated variants of IST and SEI are the relations "-IST" and "-SEI".

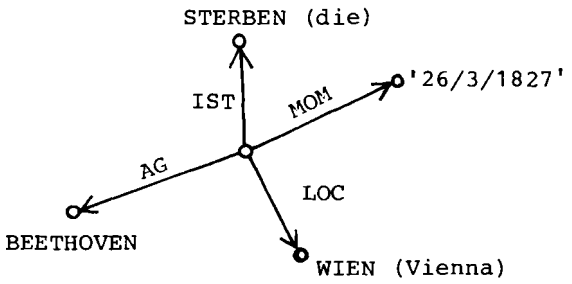
In drawing the semantic representation of simple facts, for each pair of arcs we only draw the outgoing arc and, for convenience, avoid the incoming one. So we obtain, for instance, the following representations:



("WIEN IST EINE STADT.")
("Vienna is a city.")



("PETER TUT NUN SCHLAFEN !")
("Peter is sleeping now.")

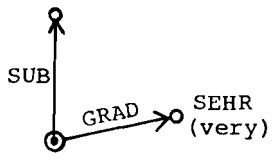


("BEETHOVEN STARB AM 26 TEN MÄRZ 1827 IN WIEN.")

The relation SUB links two concept nodes (as "cat" and "animal"), if the first concept is included in (subsumed by) the second one. Modification of established concepts by other ones leading to new conceptions is very important for human reasoning. For concept modification three relations exist: GRAD (GRADually modifying, e.g., by adverbs as SEHR ("very"), KAUM ("scarcely")), QUANT (QUANTitative modifying by quantities (e.g., 30 METER)), and MOD (general MODification of concepts, mostly by adding of adjectives or adverbs (e.g., "beautiful city," "great composer")).

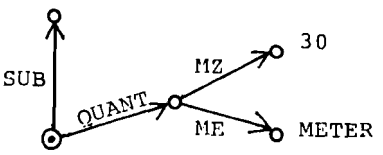
In the following examples, we always symbolize the explicit representatives of the language units under consideration by two small concentric circles.

(interesting)
INTERESSANT

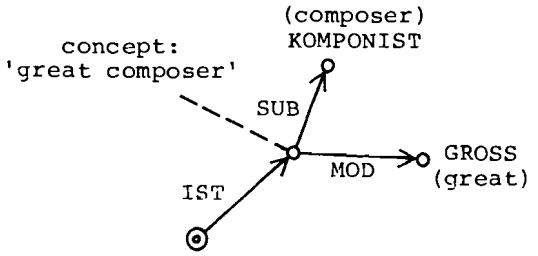


"SEHR INTERESSANT"
("very interesting")

LANG(long)

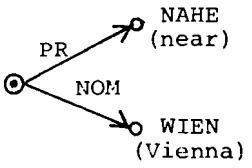


"30 METER LANG"
("30 meters long")

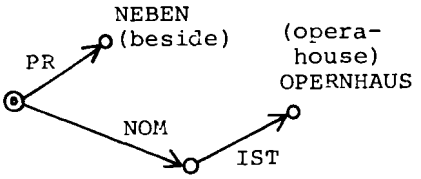


"DER GROSSE KOMPONIST : BEETHOVEN"
("the great composer Beethoven")

Locations in the simplest case can be considered as identical to physical objects or are defined in natural language (using propositional phrases) in relation to such objects, utilizing prepositions with a local meaning as "above," "near," "beside." For the semantic representation of locations, we use the relation NOM (between locations and physical objects) and the relation PR (between locations and special relational concepts named by local prepositions), e.g.,

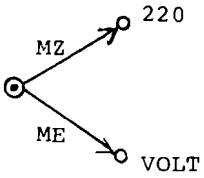


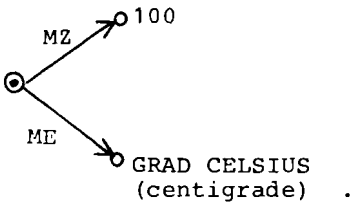
"NAHE WIEN"
(near Vienna")



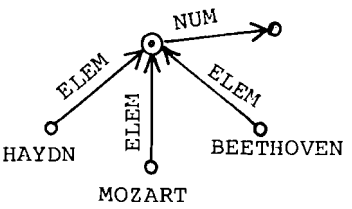
"NEBEN DEM OPERNHAUS"
("beside the opera house")

Quantities are concepts defined by utilizing the relations MZ (number of measurement) and ME (unit of measurement) as in "100 GRAD CELSIUS" or "220 VOLT," represented by:

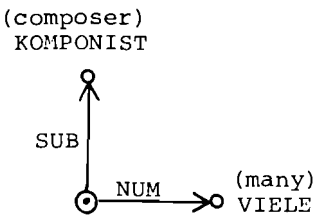




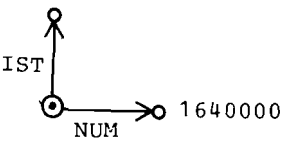
Finite sets consisting of several single individuals can be represented as compound individuals (ensembles). The number of their members is characterized by utilizing an undefined numeral (as "many," "some," etc.) or stated exactly by a natural number as the second argument of the relation NUM. If single members of such a set are explicitly represented as network nodes, they are linked to the compound individual by the relation ELEM. Examples are:



"HAYDN, MOZART UND BEETHOVEN"

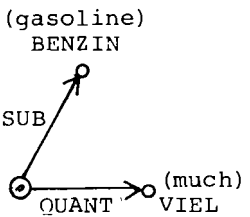


"VIELE KOMPONIST(EN)"
("many composers")

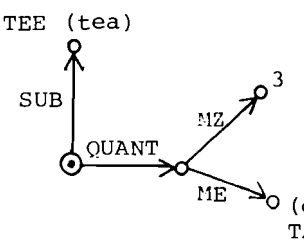


"1640000 EINWOHNER" ("1640000 inhabitants")

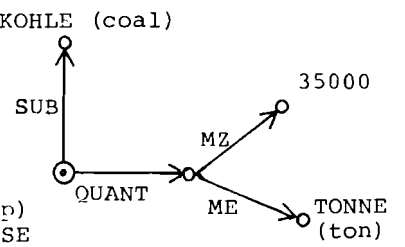
For representation of substances, look at the following examples:



"VIEL BENZIN"
("much gasoline")



"DREI TASSE(N) TEE"
("three cups of tea")



"35000 TONNEN(N) KOHLE"
("35000 tons of coal")

Some often used primitive relations characterizing individuals by immediately relating them to other entities are:

ELEM: member of a finite set of physical objects,

IN : relates an (immovable) object to the place where it exists,

MAT : relates a physical object to the material (substance) it consists of,

NAM : relation between an individual and his (not unique) name,

PART: relation between two single physical objects whereof the first is part of the second one,

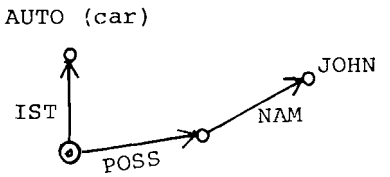
POSS: characterizes an object as POSSessed by a person or institution,

PROD: relation between an individual (perhaps of abstract nature, e.g., a work of literature) and the person who produced it,

UTIL: characterizes an individual as utilized mainly or exclusively by a special person,

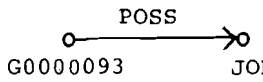
ATTR: characterizes a (mostly abstract) individual (e.g., a particular attribute or feature) as belonging to or marking another individual (e.g., "the illness of Napoleon," "the history of Vienna").

For instance, the fact, "There is a car possessed by John," could be implicitely represented as:

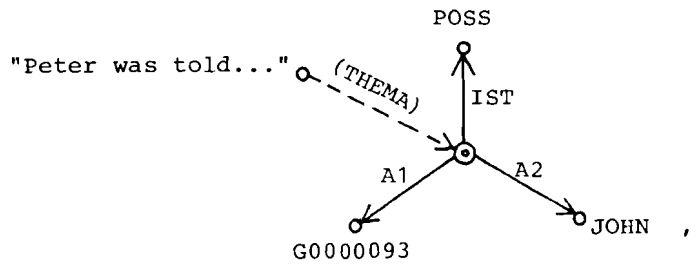


Because a car customarily has no (external) proper name, an artificial name, for instance G0000093, would be created by the program for the node representing "the car of John" in the semantic network. In this way, all nodes without external names in the network will get an artificial internal name.

Avoiding for convenience the relation NAME and directly identifying the individual John by the name JOHN, we get as implicit representation of the fact considered above,

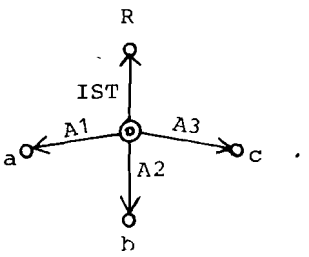


If we want to represent facts on facts, we need an explicit representation for facts. Network nodes representing intensions of factual prepositions can be created by the system and can play the role of arguments of other prepositions. For example, to represent the fact, "Peter was told that this car is possessed by John," for the fact, "this car is possessed by John," an explicit representation such as,

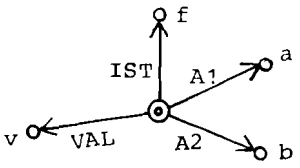


is needed.

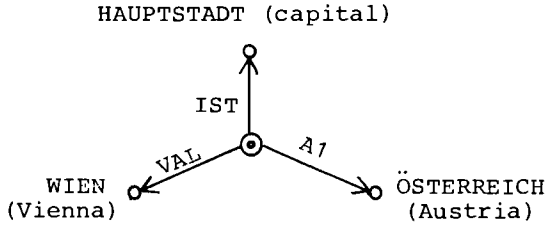
Arbitrary n-place relational expressions of the kind R(a,b,c) can be explicitly represented, using the primitive relations A1, A2, A3 for successively marking the arguments, by structures of the kind,



For attaching a value v to a functional term f(a,b) the implicit representation of f(a,b) = v is:



Example: "Vienna is the capital of Austria."



"WIEN IST DIE HAUPTSTADT - VON ÖSTERREICH."

Definite time moments (dates) of events are represented by numbers, which are to be interpreted in a special way. For instance, the date "26 th March 1827" is represented by the number 1827032600.

The relations EQ (equality), GR ("greater than"), and GREQ ("greater than or equal") are defined for two numbers, quantities or time moments. For instance, time moments, if not explicitly known, can be interrelated by these relations. Two concepts, differently described and represented before, can be defined to be equivalent (EQUIV) or to characterize two disjointed (DISJ) sets of individuals. Two individuals, differently described and represented before, are identical, if linked by the relation IDENT.

Events are explicitly represented by nodes that obtain their meaning by subordination under an event class concept (named mostly by a, perhaps modified, verb) and are specified by a number of deep-case arguments. Actions are events, (usually consciously) performed by animate individuals as actors. If no restrictions (e.g., depending on the dominating verb) are known to the system, an arbitrary number of many deep-case arguments is permitted. We did not try to restrict the number of utilized deep-case relations so far as possible (as other authors did). At present we use the following deep-case relations (the first argument is always an event, the second one we characterize in parentheses):

AG (AGens, actor, causal actant; causing or performing an action; not restricted to persons),

DAT (DATive-deep-case, specifying an entity (mostly a person) the event is directed or related to),

DEST (DESTinated person, not present on the place of an event, the event is directed or related to this person)

DIR (DIRection or intended end-point (place) of an event, particularly specifying a movement of a physical object)

INSTR (INSTRument of an action)

LOC (LOCation of an event)

MATC (MATerial used for an action of producing something)

MOM (time-MOMent of an event)

OBJ (OBJect relation, its meaning is strongly dependent on the special properties of the dominating verb; the deep-case object is usual an individual that is affected by the described event).

ORIG (place of ORIGin or direction a moved object is coming from)

ROL (ROLE an actor (or, in passive sentences, object) is playing in the event)

Related to deep-case relations are the following relations between two events:

CAUS (CAUSal relation between two events)

CIRC (CIRCumstances, surrounding conditions)

CONC (CONCordance; for events, occurring according to a fixed or prescribed event pattern)

CONDIT (CONDITional relation; "if-then-relation" between two classes of events)

DAN ("then-relation", temporal ordering of events)

INT (INTention or purpose of an event)

MENT (emotional condition or state of mind of the actor in an event)

MODC (MODe or manner in which an event takes place)

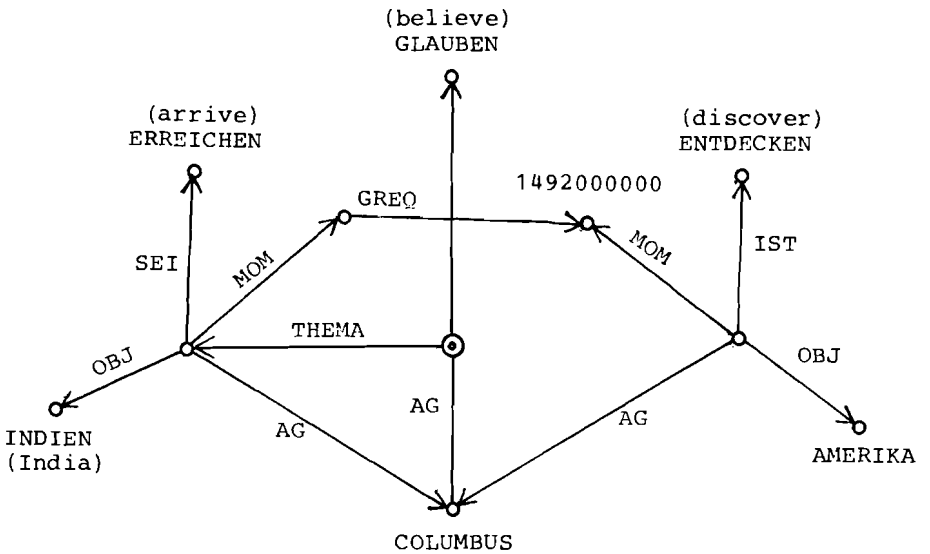
METH (method of an action)

SIMULT (temporal relation between events occurring SIMULTaneously)

THEMA (THEMA or topic of a cognitive, communicative or perceiving activity).

Significant difficulties arise in interpreting natural language sentences with regard to the theoretically important distinction between events (individuals) and event classes (concepts). Mostly the decision is highly context dependent, and, until now, the difference is not always sufficiently regarded by the program. Our network also allows the representation of variables produced, for instance, in processing the quantifiers IRG and JED or the pronouns MAN, JEMAND, NIEMAND, ETWAS, NICHTS of the core language. There are two additional sorts of nodes representing respectively existential and universal quantified variables. So some not too complicated and often used general prepositions which can be handled by the deductive mechanisms of the system, can be represented. What has not been possible to represent as network structures until now are general prepositions with many quantifiers different in scope. Difficulties will also be caused by the logical connector V (inclusive or).

Finally, look, for example, at the semantic representation (consisting of three event nodes) of the compound sentence concerning Columbus ("Columbus believed he had arrived in India, when he discovered America in 1492.").



8. The Language Processing Program

The overall structure of the input-processing program is shown in Figure 1.

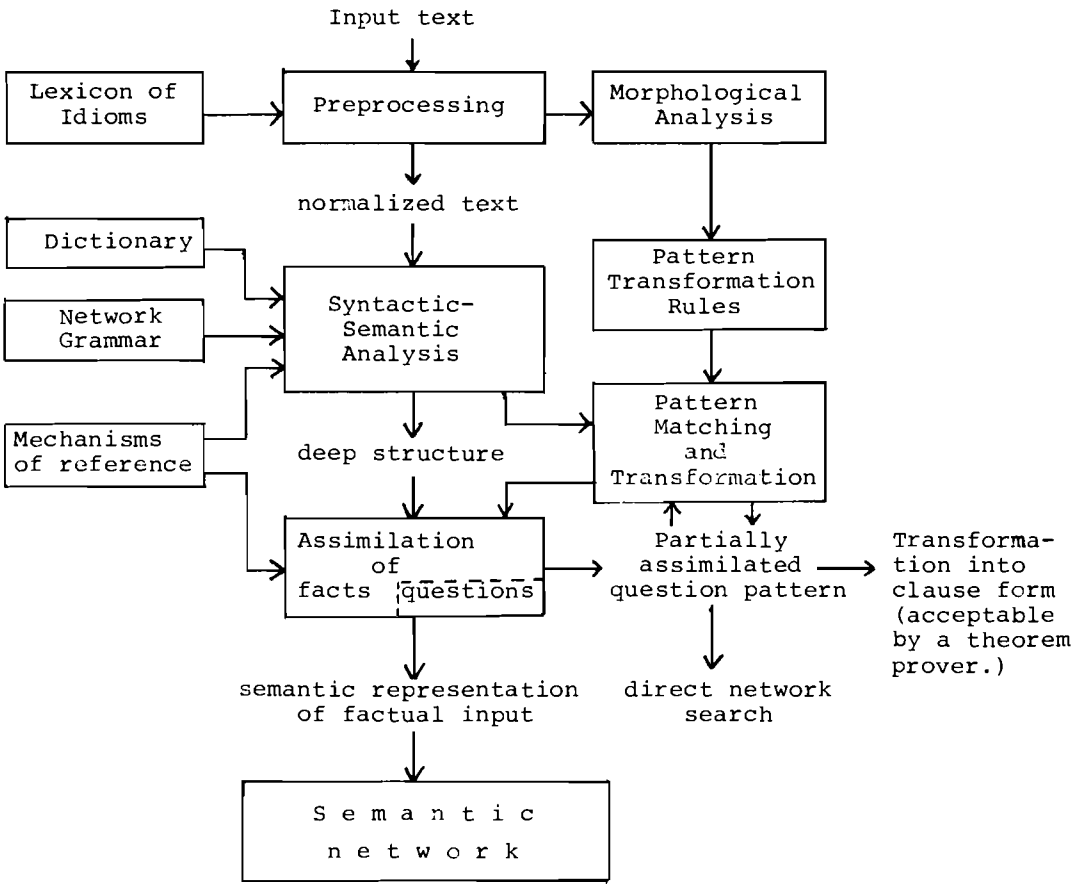


Figure 1.

9. The Network Parser

Our network parser is a recursive LISP program. It takes as arguments the top-level grammar net for main sentences (a voluminous data structure) and the normalized sentence to be analyzed. Sentences are represented as simple lists consisting of atomar words only. This parser can be considered as a kind

of general interpreter of network grammars. The analysis could be done more efficiently by working in a compilerlike manner (first compiling the network grammar to a LISP program, then compiling this program once more), but, for purpose of grammar testing and continuing modification of the analysis process, the interpretative mode seems favorable.

The parsers has different working modes selectable by switches. It can proceed

a) in a straightforward manner completely without backtracking,

b) in a smooth backtracking mode, terminating its work, if a first apparently acceptable analysis result has been found,

c) in a strong backtracking mode for exhaustively discovering ambiguities, searching through all possible ways of analyzing the given sentence.

In analyzing FES texts as input, we normally used mode (b) as most appropriate; for the core language mode (a) is sufficient.

The parser also has many other test facilities for tracing the flow of analysis, for producing (as a luxurious by-product of the complex analysis process) a kind of plain surface structure of analyzed sentences (which can be printed out for aesthetic reasons, but is not practically used anywhere in further processing).

The often very complex deep structure produced by the analysis of a sentence (without assimilation) can be pretty-printed, showing the treelike structure in a satisfying two-dimensional arrangement easy to survey. If the analysis of a given sentence was not successful, the parser tries to localize the place in the sentence where the trouble occurred by printing out the shortest nonanalyzed remainder of the sentence as well as additional information characterizing the temporal content of the network registers and the state of the analysis at the time when the parser had most deeply penetrated into the sentence.

10. The Preprocessor

Before the text is analyzed sentence by sentence with the network parser, a preprocessor for common idioms that do not fit into the input language grammar (or would be semantically misunderstood) is activated. In processing a sentence word by word from left to right, the preprocessor tries to apply replacement rules that are stored in a lexicon of idioms. These replacement rules are basically similar to the well-known rewriting rules appearing in type 0 Chomsky grammars. But the power of our rules is substantially enlarged by mechanisms for substitution of variables and additional sophisticated facilities for pattern matching.

11. The Network Grammar

For the analysis of our input language, we use an augmented recursive transition network grammar specially designed along the lines described by Woods (1970). This network grammar is always rather large and sophisticated and, in fact, accepts a great part of normally formed German sentences.

According to my own experience, this kind of grammar model seems well suited for the task of language understanding because it has a sufficiently transparent syntactic skeleton and, at the same time, offers unlimited computational power. Arbitrary actions for normalization of inflected word forms by morphological analysis, for context-dependent handling of unknown words, for resolving references, for performing grammatical transformations, and for building complicated structures as analysis results can be incorporated.

Our network grammar consists of eight single networks for different types of language units (see Figure 2). In traversing a network, if the analysis proceeds along a PUSH-arc, another (or the same) network will be (recursively) called (analogous to a procedure call). The top-level network accepts all kinds of main sentences (prepositions, questions, and commands).

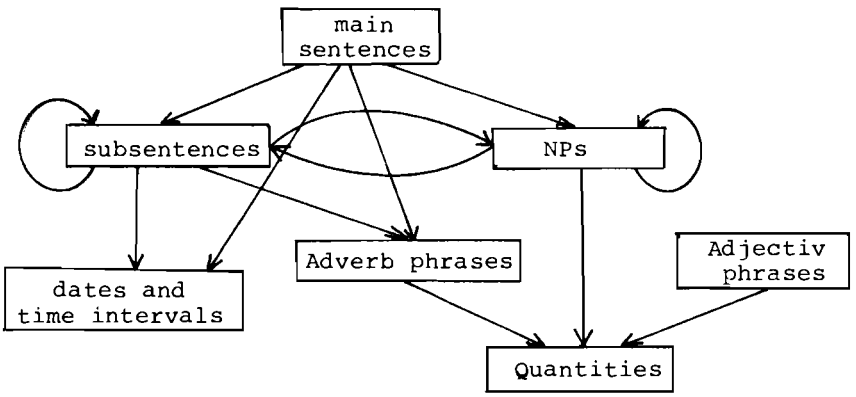


Figure 2. Hierarchical structure of the network grammar.

Although our network grammar is a large data structure, it can be considered as a nondeterministic analysis program. Therefore, the algorithmic properties of these grammars can be utilized to increase the efficiency of the analysis process. Structure building actions described in the network grammar (and invoked by the parser) produce the treelike deepstructures for successfully

analyzed parts of the input text. We want to use the term deep structure without special linguistic connotations (as in the sense of interpretative semantics) simply for a kind of intermediate representation of single sentences.

It should be stressed that our approach to formalizing a grammar seems a little odd from a theoretical point of view. The grammar describes only an acceptor (and in no way a generator) for sentences. It is very liberal in accepting sentences because the system reasonably can suppose that from common users who are vitally interested in obtaining answers to their questions, no senseless or strongly ungrammatical questions will be posed. Also the input texts will be at least sequences of correct German sentences. So the program tries so far as possible to understand what is given to it and accepts some slightly ungrammatical sentences.

12. The Deep Structures

The produced deep structures can be syntactically described as having the form:

```
<deep strucure> ::= <word>|<variable>|
                    (<sort functor> (<relation><deep structure>)
                                                          ...)|
                    (<special functor>[<deep structure>]...)  .
```

Sort functors characterize the special sort of the semantic representant and mostly produce concepts. Special functors are for instance the functors named by the determiners of the core language: E, D, C, IRG, JED. Special functors will be evaluated during the assimilation and then produce representants of individuals. E produces new individuals, D searches individuals (already stored in the network) sometimes by complicated mechanisms, IRG and JED are producers of quantified variables, and C delivers concepts.

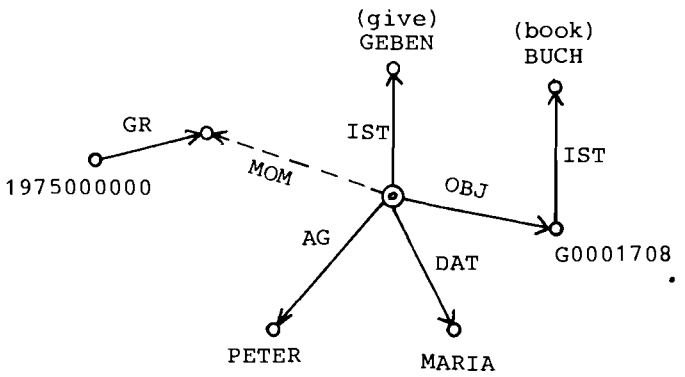
For example, for the sentence:

"DAS BUCH WURDE IHR VON PETER GEGEBEN"
("The book was given her by Peter"),

the analysis would produce the deep-structure representation

```
(AUSSAGESATZ
   (IST GEBEN)
   (TEMP PRAET)
   (AG PETER)
   (DAT MARIA)      (if the referent of the pronoun IHR
   (OBJ             ("her") in the actual context was MARIA!)
      (D BUCH)))  .
```

After complete assimilation, this deep structure will be explicitly represented by a node that obtains its meaning by the following small part of the semantic network:



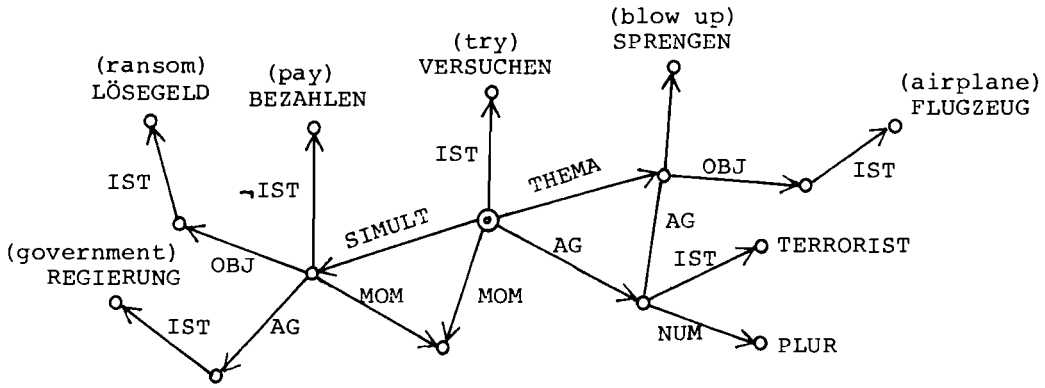
Now, a more complicated sentence is considered:

"DIE PLUR TERRORISTEN VERSUCHTEN (DAS FLUGZEUG ZU SPRENGEN), ALS DAS LÖSEGELD NICHT VON DER REGIERUNG BEZAHLT WURDE." ("The terrorists tried to blow up the airplane when the ransom was not paid by the government.")

As deep structure representation is pretty-printed:

```
(AUSSAGESATZ
   (IST VERSUCHEN)
   (AG
      (D
        (MENGE
           (IST TERRORIST)
           (NUM PLUR) )))
   (TEMP PRAET)
   (THEMA
      (SUBSATZ
         (SEI SPRENGEN)
         (AG
            (D
              (MENGE
                 (IST TERRORIST)
                 (NUM PLUR))))
         (OBJ
            (D FLUGZEUG) )))
   (SIMULT
      (SUBSATZ
         ( IST BEZAHLEN)
         (TEMP PRAET)
         (AG
            (D REGIERUNG) )
         (OBJ
            (D LOESEGELD) ))))  .
```

The following semantic representation results:



13. The Dictionary

Of great importance for our dictionary and the recognition of word classes is the distinction between open and closed classes of words. The analysis of the FES is done on the assumption that all particles of the language were known to the system (i.e., stored in the dictionary or specially handled by the program). Therefore, the different classes of particles are "closed."

So we developed a basic version of a dictionary containing only these particles and a few of the most frequently used content words. Particles appear with a high frequency, covering more than 50 percent of the words in a normal text and are most important for understanding the basic structure of utterances.

Because our classification of words is oriented by semantic as well as syntactic aspects and will simplify the task of analysis, we distinguish a greater number of word classes than the traditional grammarians did. Often our criteria for establishing special word classes resulted from pragmatic considerations. Because the utilized strategy for analysis permits dealing with word forms (lexemes) belonging to more than one main word class, the classes we obtained can overlap, and a finer subclassification was settled wherever feasible. The decision as to what class or meaning of a word was intended in an utterance always depends on the context of the questionable word.

Some kinds of properties can be inscribed in the dictionary for different classes of words and were marked by the following labels:

FEAT attaches word classes to words,

BASF attaches a basic word form to an inflected word form (mainly for irregularly formed inflected word forms of verbs and deviant plural forms of nouns),

CASE attaches possibly intended cases to a particle (these are to be considered as deep cases for prepositions, conjunctions, and interrogative particles, and as surface cases for determiners and pronouns),

TEMP attaches a tense specification to conjugated verb formes,

GEN specifies the gender of a substantive,

SYN specifies a synonymous word,

AMBIG attaches different word meanings to an ambiguous word.

14. Morphological Analysis

If, in the analysis process, a word is encountered not known before to the program, then it will always be supposed that it would be a member of an open word class. For uncommon content words, not supposed to be in the systems dictionary, the user can put an artificial morpheme in front of the unknown word for marking the intended word class.

For handling words that are not marked and are not included in the dictionary, the program has a rich arsenal of (partly heuristic and sometimes fairly risky) techniques for forming assumptions concerning the possible class membership of the questionable word. By morphological analysis, a word is divided into a string of letters. Suffixes (and, if necessary, also prefixes) are stripped and the resulting reduced forms are compared with the always stored words. Lists of common suffixes characterizing German adverbs, nouns, adjectives, and verbs are used for the morphological analysis.

A list of common German prefixes (although normally less significant for word class recognition) is available too. Particular procedures of morphological analysis are combined with considerations of the word context in a delicate way to form assumptions concerning the possible class membership of unknown words.

A Formal Framework for Unitary Approach to the Theory of Problem Solving

Giovanni Guida, D. Mandrioli, A. Paci, and M. Somalvico

1. Introduction

Computer science, considered as that unitary discipline which deals with all the basic problems tightly interconnected with the existence of the computer, has been developed in the last thirty years under the stimulating pressure of many theoretical and applicative exigences.

The rapidity and extension of the technological progress has, therefore, produced a quite tumultuous and badly organized development of computer science which, at the present moment, still lacks a well-structured understanding of the fundamental notions on which this new discipline is based.

It is, in fact, very surprising that only recently, i.e., in the last ten years, a theory of programs (also indicated as mathematical theory of computation) has been proposed and conspicuously developed over the basic notion of program.

Still more recently, the research work stimulated within the domain of artificial intelligence in general, and within the domain of problem solving in particular [1,3,4,8,9], has awakened the need for a new theoretical effort centered around the notion of problem which more and more appears to be of central importance within computer science.

This newly developing theory of problems provides a better understanding of the path followed by computer science in its development, which now seems to be coming out of its prehistoric phase [10-12].

The studies about the theory of problem solving to which belongs the algebraic approach that we shall present in this paper [2,5-7] are intended to achieve the following main goals:

a) a rather precise understanding of human behavior in problem solving;

b) a clear definition of what we mean by an automatic problem solver (APS);

c) a proposal of an internal structure of an automatic problem solver which can perform the three basic activities of selection, search, and learning;

d) a constructive comparison between the theoretical possibility of an absolutely general automatic problem solver and the practical requirement of a tool useful for the man;

e) the formulation of a theory of problems which can be helpful as a theoretical base in the design of an automatic problem solver;

f) further investigations about the automatic problem solver as a nondeterministic interpreter of a high-level representation language and as an automatic programmer.

The proposal of the unitary approach that we are going to present may be seen as a first step toward a complete formulation of the theory of problems and a better understanding of the basic concepts about the automatic problem solver.

This paper is devoted to present:

a) in section 1, a description of the role and of the implication of problem solving research in computer science, together with the outline of the basic functions and fundamental structure of problem solvers and representation languages interpreters;

b) in section 2, the basic definitions about the notions of problem and of solution, strictly related to the state-space approach to problem solving; these notions have been formalized in order to provide a general base upon which to develop successive investigations;

c) in section 3, the detailed investigation of the state-space S with the purpose of supplying it with an appropriate framework, utilized for developing considerations on the problem-solving methodology;

d) in section 4, the illustration of the problems which arise when costs are taken into account; in this way, the quality of the solutions obtained is illustrated as well;

e) in section 5, some preliminary results which are related to the problem of comparing different representations of problems; in particular, the conditions which allow us to consider two problems, although obtained by semantic domains of different nature, as substantially equivalent, are described as well;

f) in section 6, general notions and criteria designed to confront and compare different problems represented within various approaches to problem solving (namely state-space approach and problems-reduction approach), together with different types of relations between problems and properties based on these notions;

g) in section 7, the description of the unitary aspect of our theory presenting the main implications of the outlined formal approach on problem solvers and representation languages design criteria; a general and formalized description of the structure of a problem solver is proposed as well;

h) in section 8, a brief insight into the semantic description of our unitary approach which is now being investigated and which is not yet completely developed; other conclusive remarks and promising research directions are presented at the end of the section as well.

In this section, we are now going to illustrate the implication of problem-solving research in computer science. The standpoint of our considerations is the following one: computer science is an experimental discipline which is centered around a unitary and global goal: man-computer interaction in solving problems.

As any experimental discipline, therefore, computer science can be viewed as involved in the passage between two distinct worlds, namely, the world of reality, and the world of cognition of reality. Between these two worlds, a gap exists which can be overcome only by human ingenuity and creativity, but absolutely not by means of any mechanical or artificial technique. The activities required by any experimental discipline can be very well exposed within the framework of many philosophical models.

We will utilize, for our purposes, the Galilean inductive-deductive experimental method. The Galilean method is represented by a direct graph with three vertices and three arcs. In Figure 1, we illustrate such a graph in the case of physics as an example of an experimental discipline. In such an example, the three vertices of the graph correspond to the notion of phenomenon (in the physical reality), of model (of such a phenomenon), and of law (which can be derived within such a model). The three arcs represent three conceptual activities, which are related to the interaction between the world of reality and the world of cognition of reality.

Formalization is the first activity, which is exclusive of human creativity and invention and which enables man to substitute the informal and intuitive notion of phenomenon with the rigorous and precise concept of model within a selected formal framework.

Induction-deduction is the second activity, which lies within the world of cognition of reality, and which allows both man and mechanical tools to infer a law as a consequence or property of a formalized reality.

Matching is the third activity which allows man, again by means of human creativity, to confront the validity and the utility of the formally obtained law with the phenomenon, considered as the source of the whole experimental cognition process.

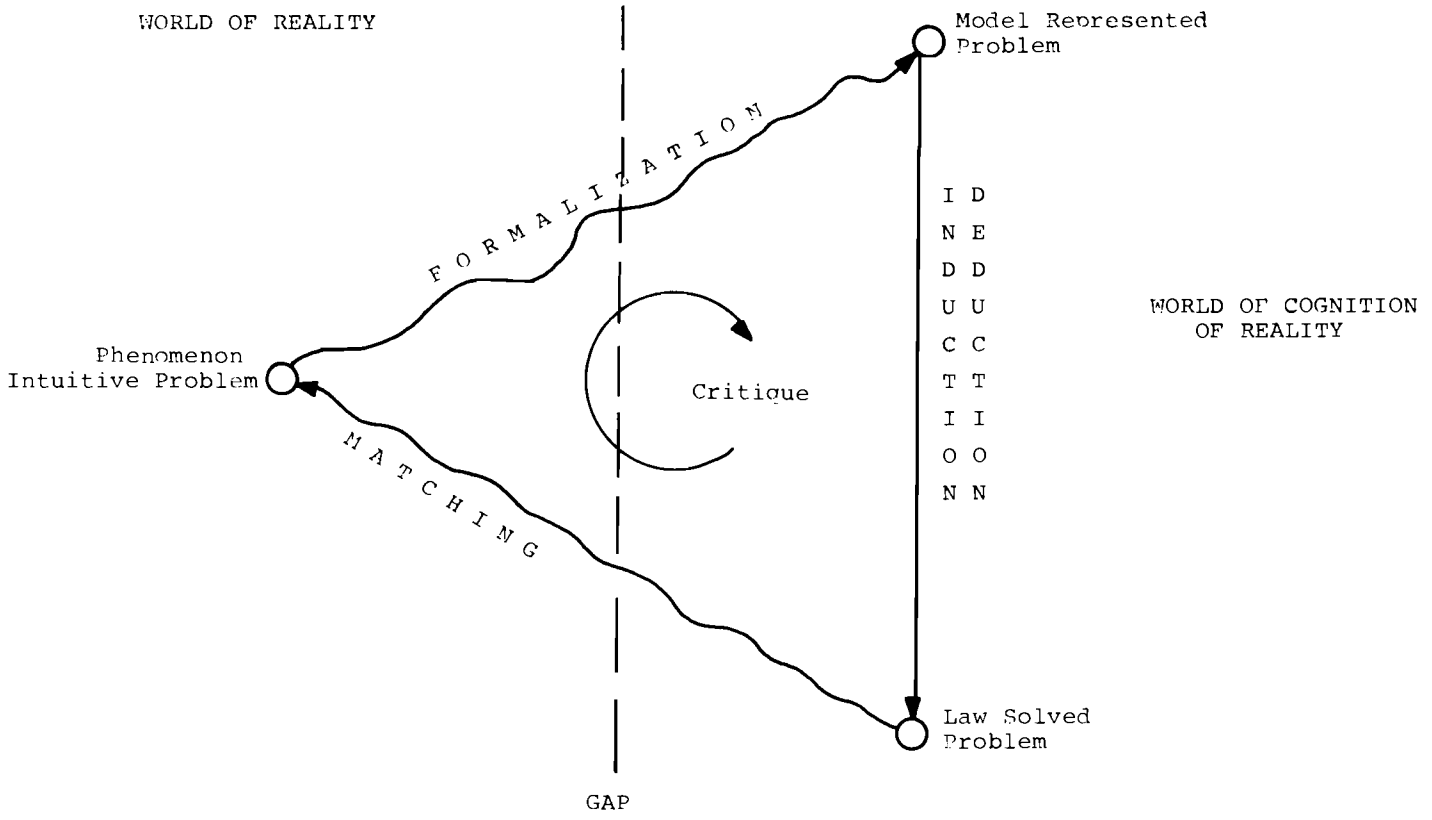


Figure 1. The Galilean inductive-deductive experimental method.

The whole cycle is under the scrutiny of the man by means of continuous critique which may bring up the convenience of an improved repetition of the whole cycle itself.

The new experimental discipline of computer science can be embedded in a similar way by simply introducing new conceptual notions in correspondence to each vertex of the graph. More precisely, with respect to the case of physics, we have the following substitutions:

a) for the notion of phenomenon, we substitute the notion of intuitive problem;

b) for the notion of model, we substitute the notion of represented problem;

c) for the notion of law, we substitute the notion of solved problem.

The intuitive problem is an entity which independently faces the man and can be viewed as an undefined and unlimited source of information.

From it, through the activity of formalization, the man operates an extraction of a finite and precisely described amount of information, namely the represented problem (i.e., the representation of the problem). This information is chosen as valuable and sufficient in order to provide--through mechanical, or interactive, computation--the construction of the solved problem (i.e., the solution of the problem).

It is very important to observe that, because of the overall meaning of the Galilean method, the whole passage between the represented problem and the solved problem lies within the world of cognition of reality, i.e., within a completely defined and formalized environment.

Hence, in principle, such a passage, i.e., the solution construction, can be performed in an artificial and completely automatic way. On the other hand, the two other activities of formalization and matching belong exclusively to the man. These are the fundamental conceptual considerations which illustrate the role of an automatic problem solver within computer science.

In the same way, it has been shown what can be considered as the ultimate expandibility of the role of the computer within the man-computer interaction in solving problems.

However, this ultimate goal can be considered only as the final target of the development of automatization within computer science. On the other hand, it is important to observe that at the present state of the art, the passage between represented problem and solved problem still requires, in the practical and available technological impact of computer science, a widely extended and intense cooperation between man and computer.

The role of the man in such cooperation, while avoidable in principle, bears on itself the responsibility of the conceptually most important activities. The trend of computer science research therefore can be considered, with an overall synthesis of its historical development, as based on the criterion of continuously reducing the impact of man, and increasing the ability of the computer. In particular, the trend of artificial intelligence research can be considered as centered on the aim of completely eliminating the role of man in such a construction of a solved problem for a represented problem. In Figure 2, we illustrate the present state of such a cooperation process. The responsibility of the man is shown to be:

a) Invention of the representation of the problem;

b) invention of an algorithm for the construction of the solution of the problem as it has been represented;

c) construction of a source program written in a symbolic and, generally, high-level programing language;

d) critical matching of the solution of the problem with the intuitive problem embedded in its semantic domain.

Please note that because of the previously exposed philosophical considerations while activities a) and d) belong always to the man, activities b) and c) can potentially be taken over by the computer.

The responsibility of the computer is shown to be:

a) translation of the source program into an object program written in machine language,

b) execution of the object program,

c) construction of the solution of the problem.

During its growth, computer science has been based and guided by appropriate theories in order to understand and organize the methodology followed by human activities and to construct specific artificial systems responsible for the computer activities.

Specifically, the following theories have been particularly important:

a) theory of computability, for understanding the limit and power of invention of algorithms;

b) theory of complexity, for providing a basis for critical evaluation of efficiency in the invention of algorithms;

c) theory of formal languages, for providing frameworks and tools useful in order to design and to construct such artificial systems as the translators (compilers, assemblers, interpreters) from source programs to object programs;

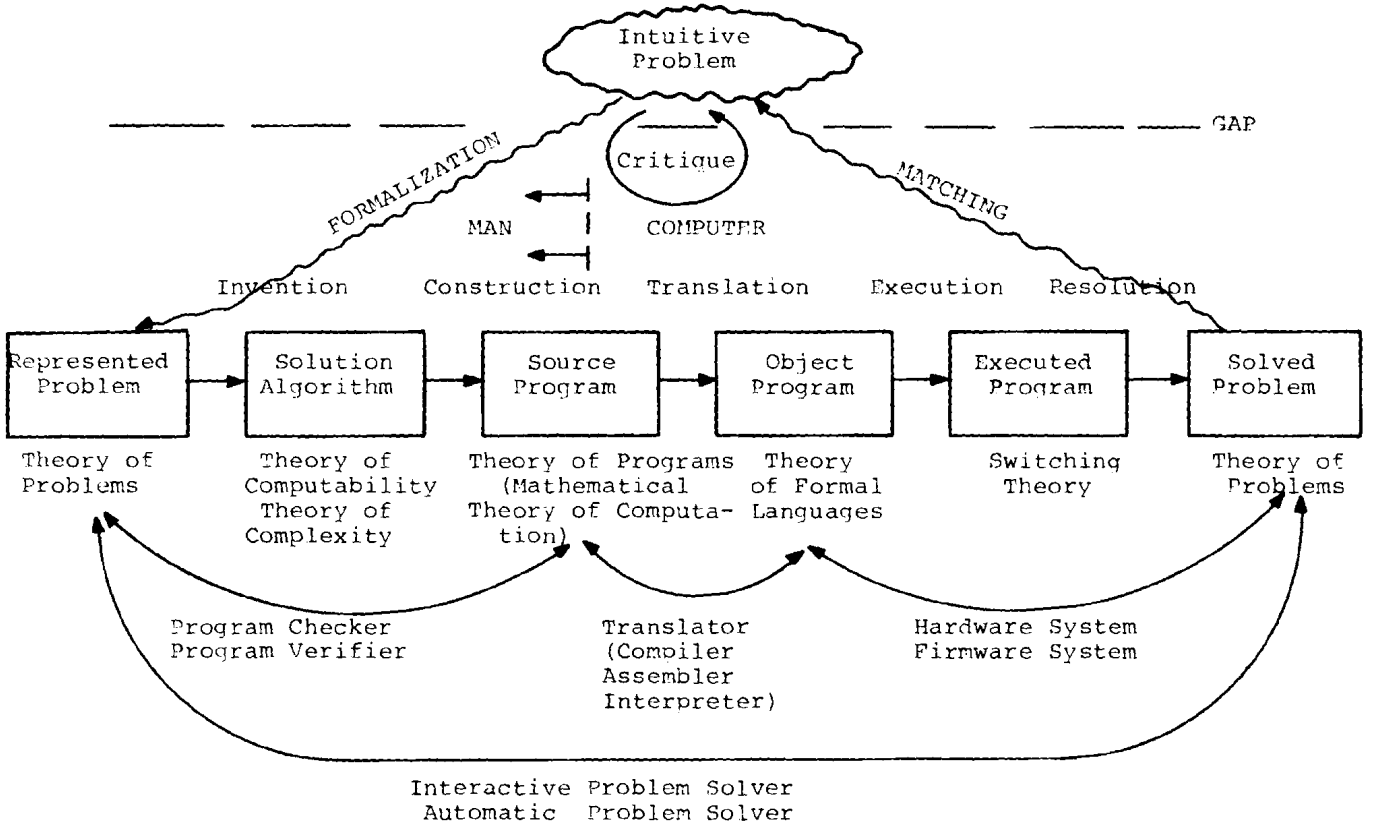


Figure 2. Present state of art in man-computer interaction in solving problems.

d) switching theory, for providing frameworks and tools useful in order to design and to construct such artificial systems as the hardware systems and fireware systems capable of the execution of object programs and the construction of the problem solution.

More recently, a new theory has been proposed and embryonically developed, namely:

e) theory of programs (mathematical theory of computation), for providing a basis for critical evaluation of invention of programs, and for providing frameworks and tools useful in order to design and to construct such artificial systems (though not yet available, at the present state of the art) as the program checkers and the program verifiers.

The recent results of problem solving within artificial intelligence research make the development of a new theory increasingly more necessary. In principle, a new theory can provide us with the bypassing of all the classical and also the quite recent theories. Such a new theory can be defined as:

f) theory of problems, for providing a basis for critical evaluation of the techniques capable of directly and automatically obtaining the solved problem from the represented problem, and for providing frameworks and tools useful in order to design and to construct such artificial systems (considered as becoming available in the not too distant future) as the interactive problem solvers and the automatic problem solvers.

In this way, we have achieved the purpose of illustrating the role of problem solving within computer science by means of an illustration of its meaning and importance in the scenario of development of computer science.

Let us now present a more detailed understanding of the functions and of the structure of an automatic problem solver. We shall illustrate first the basic conceptual functions involved in an automatic problem-solving activity, and we shall, by consequence, present a structure for an automatic problem solver.

We want to point out that all the considerations that we are going to illustrate can be more deeply understood in a critical way if one thinks of semantic domains from whence intuitive problems arise, which are not already set up by means of any rigorous description (e.g., such as semantic domains originated from the world of mathematics, or of logic).

Namely, one should think of such kinds of semantic domains (e.g., robotics, natural languages understanding, human situations, actions, behaviors, etc.) where:

a) the representation of any problem is a real invention for the man, requiring an effort of creativity and ingenuity;

b) any represented problem still leaves completely unidentified, also in an implicit way, the solved problem (that might eventually be obtained), which may as well be completely out of reach even of human imagination.

The formalization activity, performed by the man, provides, as it has been previously supposed, the represented problem as an artificial object which is obtained from, and is the substitute for, the intuitive problem.

The invention of the represented problem consists in the precise description of a finite quantity of information which the man formulates by means of the observation of two distinct entities, namely:

a) the intuitive problem, embedded in its semantic domain, and considered as an unlimited source of formalizable or representable information; this entity, considered as a "natural" entity, is provided by the world of reality;

b) the automatic problem solver, intended as a general-purpose tool which can deal with represented problems originating from various semantic domains; this entity, considered as an "artificial" entity, is provided by the artificial-intelligence scientist.

The invention of the represented problem requires that the man performs two basically different activities in its formalization process.

The first activity is devoted to the specification of the methods and ways which shape the automatic problem solver, considered as an originally general-purpose tool, into a well-precised special-purpose tool which is oriented by the semantic domain from which the intuitive problem is originated.

In other words, this activity is devoted by the man to "tune" the general-purpose tool into a special-purpose tool, by way of utilizing the human ingenuity and understanding of the best mode in which the artificial tool works more efficiently in attacking the solution process for the particular intuitive problem. By means of this activity, the general-problem solver is transformed into a special-problem solver.

The information described by consequence of this first activity is called control information, and it is the first part of the information contained in the represented problem.

The second activity is dedicated to the selection from the intuitive problem of a finite quantity of information, well defined, which is considered by the man as useful, and, hopefully, efficient and sufficient in order to allow the special-problem solver to achieve its goal of providing an automatic solution of the problem.

The information described by consequence of this second activity is called problem information, and it is the second part of the information contained in the represented problem.

It is conceptually important to observe that both the two previously described activities are done by the man conscious of being faced by ignorance of three different types, namely:

a) whether the control and problem information, contained in the represented problem, is sufficient in order to make the computer able to solve the problem;

b) what part of this information is actually relevant to the computer and should be utilized in order to solve the problem;

c) what is the actual way in which the relevant part of this information should be processed (possibly efficiently) in order to construct the (possibly optimum) solved problem, in the event that such a construction might be attainable.

It is important to observe that the first type of ignorance is a direct consequence of the existing gap between the world of reality and the world of cognition of reality.

In order to overcome this ignorance, the man needs to wait for the end of the whole experimental cycle; by means of the activity of matching, he will be able to check if an acceptable solution of the intuitive problem has been obtained, and, thus, he will be able to overcome this first type of ignorance.

The two other types of ignorances are very important because they are useful to point out two functions, performed by the automatic problem solver, which are intended to give artificial answers to this ignorance.

The first function, which is devoted to produce an automatic answer to the second type of ignorance, consists of an appropriate selection of one part of the information contained in the represented problem and considered, by the automatic problem solver, as useful and relevant for its activity of solving problems.

This activity is performed by a first part of the automatic problem solver, called selector, as it is shown in Figure 3, where all the structure of an automatic problem solver is illustrated.

Therefore, we will call the global represented problem the input of the selector, and the selected represented problem the output of the selector.

The second function, which is devoted to produce an automatic answer to the third type of ignorance, consists of a skillful search of the cooperation process, embracing the already selected information, which essentially makes up the solution algorithm and, thus, yields the solution of the problem.

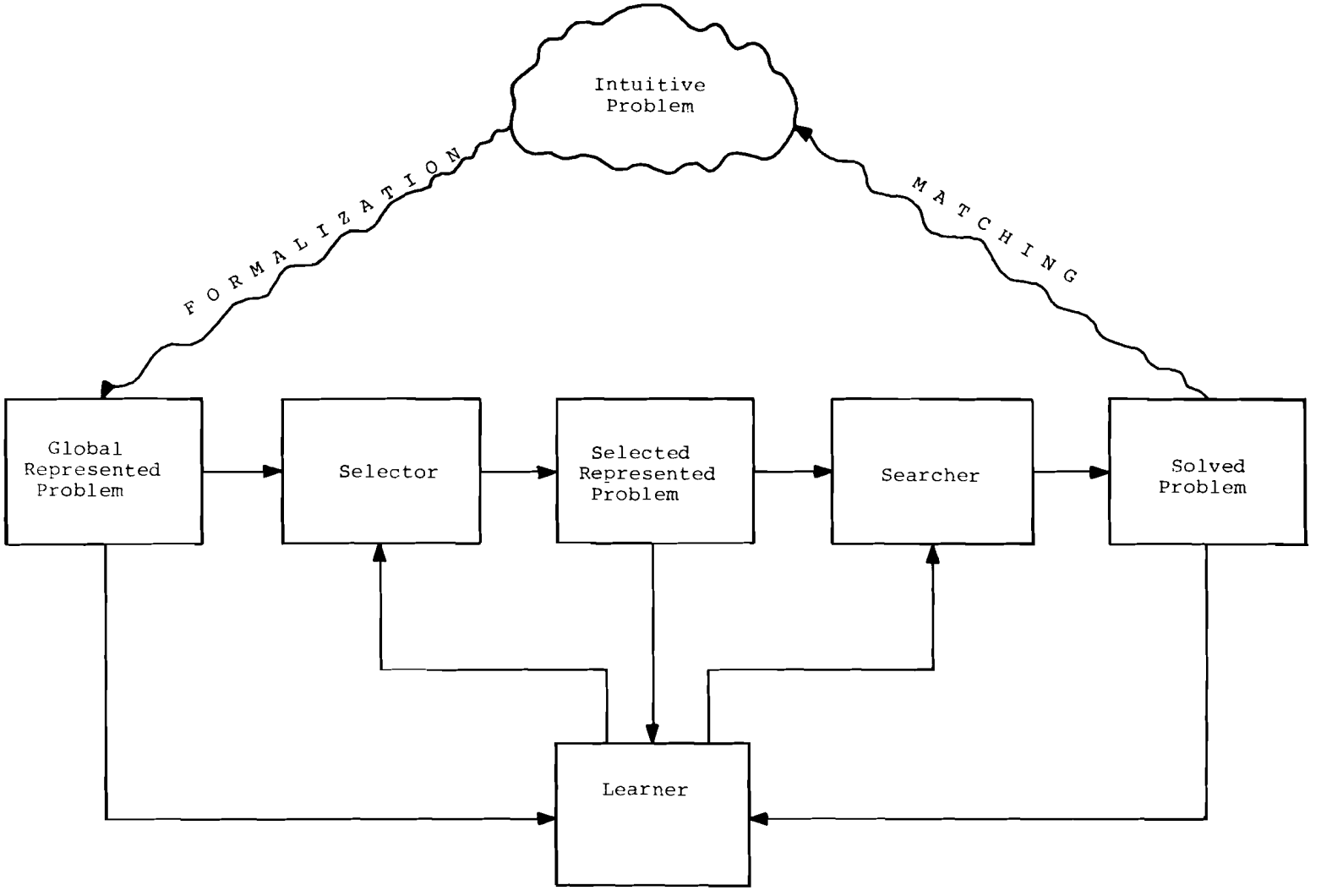


Figure 3. Structure of an automatic problem solver.

This activity is performed by a second part of the automatic problem solver, called searcher, as it is shown in Figure 3. Therefore, while the input of the searcher will be the selected represented problem, the output of the searcher will be the solved problem.

As it has been previously illustrated, the control information is the information which enables the man to specify the special configuration of the problem solver oriented toward a particular semantic domain. Therefore, by means of the control information, the structures of the selector and searcher are completely defined and specified.

In other words, the selector and searcher are two general purpose, artificial metasystems which, by means of the control information, are specialized into two special purpose, artificial systems.

This specialization of the structure of the selector and of the searcher by the man can be considered just as an initial specification which, during the ongoing solution process, can possibly be changed and improved.

This modifying and enhancing activity is the typical activity of learning which is able to provide a dynamic evolution of the structure of the selector and of the searcher.

This self-changing activity is performed by a third part of the automatic problem solver, called learner as it is shown in Figure 3.

Therefore, the inputs of the learner are constituted by the global represented problem, by the selected represented problem, and by the solved problem.

In this way, the inputs of the learner are obtained not only from the human activity of formalization of the intuitive problem into a represented problem, but also from the artificial activity of the problem solver itself; in this second case, these inputs can take account both of partial and of total results obtained from the artificial activity. The outputs of the learner are the automatically constructed and modifiable specifications of the selector and of the searcher.

Thus, also the learner can be considered as a general purpose, artificial metasystem which is able to specify the selector and the searcher into two special systems. Thus, the kernel of an automatic problem solver appears to be an artificial metasystem which is initialized by the man as an initial system, and, afterward, can evolve itself in a way appropriate to enhance its artificial performances in solving problems. Therefore, learning can be viewed as the ability of <u>self-awareness</u> of the whole automatic problem solver.

In fact, in principle, the learner itself can be considered as an automatic problem solver, which has been tuned on the semantic domain of problem solving, and which has to automatically solve the problems of constructing (or, better, specifying) selectors and searchers.

Thus, such sophisticated level in designing a learner can envisage automatic problem solvers acting completely as self-development artificial systems. In conclusion, we can assert that an automatic problem solver can operate on a represented problem and can provide a solved problem. Whichever has been the method followed by the man in performing the formalization task for the construction of the represented problem, it is necessary for him to choose an appropriate formalism adapted both to provide a "good" represented problem and to catalyze a "valid" artificial activity for the automatic problem solver.

Therefore, we can rightfully call such formalism a representation language which man needs for cooperating with the computer.

While the classic programing languages have been conceived to channel to the computer the human invention of solution algorithms, the representation languages can be conceived to channel to the computer the human invention of represented problems.

Therefore, we can look at automatic problem solvers as the interpreters of the representation languages in which the represented problems have been communicated to the computer.

Thus, it is natural to look at first-order predicate logic and at the PLANNER-like goal-oriented languages, as preliminary examples of representation languages. The interpreters of such representation languages (e.g., the theorem provers, in the case of first-order predicate logic) need, in the present state of the art, to be conceived and structured on the more formal basis of a theory of problems.

This paper is, indeed, centered around a part of this developing new theory, and the presented results provide more understanding for the design of both automatic problem solvers and representation language interpreters.

2. Basic Definitions

In this section, we present the basic definitions about the notions of problem and of solution which are strictly related to the state-space approach. These concepts are formalized here in order to provide a general base upon which to develop our successive investigations.

Definition 1

A (deterministic) problem schema M is a triple $M = (S, \Sigma, \Gamma)$ where:

$S = \{s_0, s_1, \ldots, s_{n-1}\}$ is a finite set of elements called the states of M;

$\Sigma = \{\sigma_0, \sigma_1, \ldots, \sigma_{m-1}\}$ is a finite set of elements called the inputs of M;

$\Gamma = \{\gamma\sigma_0, \gamma\sigma_1, \ldots, \gamma\sigma_{m-1}\}$ is a finite set of mappings of S into S called the operators of M. □

Definition 2

A (deterministic) problem P is a quintuple $P = (S,\Sigma,\Gamma,i,f)$, where (S,Σ,Γ) is a (deterministic) problem schema, and:

$i \in S$ is called the initial state;

$f \in S$ is called the final state. □

Definition 3

A (deterministic) extended problem $\tilde{P}$ is a quintuple $\tilde{P} = (S,\Sigma,\Gamma,I,F)$, where (S,Σ,Γ) is a (deterministic) problem schema and:

$I \subseteq S$ is called the set of the initial states;

$F \subseteq S$ is called the set of the final states. □

Definition 4

A solution of the problem $P = (S,\Sigma,\Gamma,i,f)$ is a string:

$$x = \sigma_{i_1}\sigma_{i_2} \ldots \sigma_{i_k} \in \Sigma^* \quad , \tag{2.1}$$

such that:

$$i\gamma_x = f \quad , \tag{2.2}$$

where:

$$\gamma_x = \gamma\sigma_{i_1}\gamma\sigma_{i_2} \ldots \gamma\sigma_{i_k} \quad , \tag{2.3}$$

(i.e., γ_x is made up by the composition of operators), and γ_ε is the identity function on S, if ε is the null string. □

Definition 5

A solution of the extended problem $\tilde{P} = (S,\Sigma,\Gamma,I,F)$ is a string $x \in \Sigma^*$ such that:

$$(\exists i)(\exists f)((i \in I) \wedge (i\gamma_x = f) \wedge (f \in F)) \quad . \tag{2.4}$$

□

Definition 6

The solution set of a(n) (extended) problem P ($\tilde{P}$) is the set $X_P \subseteq \Sigma^*$ ($X_{\tilde{P}} \subseteq \Sigma^*$) which contains all the solutions of P ($\tilde{P}$). □

We outline that the solution set X_P of a problem P is not necessarily finite. We are now able to introduce some initial formal properties of these notions.

Theorem 1.

Given an extended problem $\tilde{P} = (S,\Sigma,\Gamma,I,F)$ we have:

$$X_{\tilde{P}} = \bigcup_{P_i \in E_{\tilde{P}}} X_{P_i} \quad , \tag{2.5}$$

where

$$E_{\tilde{P}} = \left\{ P_i \,\middle|\, (P_i = (S,\Sigma,\Gamma,i,f)) \wedge (i \in I) \wedge (f \in F) \right\} \quad . \tag{2.6}$$

Proof. We have that:

$$X_{\tilde{P}} = \left\{ x \,\middle|\, (x \in \Sigma^*) \wedge ((\exists i)(\exists f)((i \in I) \wedge (i\gamma_x = f) \wedge (f \in F))) \right\} \quad . \tag{2.7}$$

Moreover,

$$X_{P_i} = \left\{ x \,\middle|\, (x \in \Sigma^*) \wedge (i\gamma_x = f) \right\} \quad . \tag{2.8}$$

Therefore, by the definition of union of sets, we have that:

$$\bigcup_{P_i \in E_{\tilde{P}}} X_{P_i} = \left\{ x \,\middle|\, (x \in \Sigma^*) \wedge ((\exists i)(\exists f)((i \in I) \wedge (i\gamma_x = f) \wedge (f \in F))) \right\} \quad . \tag{2.9}$$

Hence, we conclude that:

$$X_{\tilde{P}} = \bigcup_{P_i \in E_{\tilde{P}}} X_{P_i} \quad . \tag{2.10}$$

□

Although this theorem states a close relation between the solution of an extended problem and the solution of a set of problems, there is no indication about the methods of how to "reduce," in a general case, the solution of an extended problem $\tilde{P}$ to the solutions of the set of problems $E_{\tilde{P}}$.

In fact, this "reduction" is closely related to the search strategy adopted in the problem-solving process.

Conversely, we want to focus our interest in the following pages only on problems and their properties.

Definition 7.

The length 1 of a string $x \in \Sigma^*$ is defined as the measure function of Σ^* into N (set of natural numbers) such that:

$$1(\sigma) = 1 \qquad (\forall\sigma)(\sigma \in \Sigma) \tag{2.11}$$

$$1(xy) = 1(x) + 1(y) \quad (\forall x)(\forall y)((x \in \Sigma^*) \wedge (y \in \Sigma^*)) \quad . \qquad \square$$

We have, then, this well-known property of the null string.

Theorem 2.

We have that:

$$1(\varepsilon) = 0 \quad . \tag{2.12}$$

Proof. $1(xy) = 1(x) + 1(y)\,(\forall x)(\forall y)((x \in \Sigma^*) \wedge (y \in \Sigma^*))$. (2.13)
But when $x = \varepsilon$, we have that:

$$1(\varepsilon y) = 1(\varepsilon) + 1(y) \quad . \tag{2.14}$$

Since,

(2.15)

$$\varepsilon y = y \quad ,$$

we obtain that:

$$1(y) = 1(\varepsilon) + 1(y) \quad . \tag{2.16}$$

This equation between natural variables has the only solution:

$$1(\varepsilon) = 0 \quad . \tag{2.17}$$

$\square$

Definition 8.

A simple cost c is a measure function of Σ^* into R (set of real numbers) such that:

$$c(xy) = c(x) + c(y)\,(\forall x)(\forall y)((x \in \Sigma^*) \wedge (y \in \Sigma^*)) \quad . \tag{2.18}$$

$\square$

Theorem 3.

We have that:

$$c(\varepsilon) = 0 \quad . \tag{2.19}$$

Proof. The proof is similar to the proof of Theorem 2. □

Theorem 4

A simple cost c is completely determined by its restriction to Σ.

Proof. Because of Definition 4 and (2.1), we have that:

$$x = \sigma_{i_1}\sigma_{i_2} \ldots \sigma_{i_k} \quad (\forall x)(x \in \Sigma^*) \quad . \tag{2.20}$$

Therefore, because of Definition 8, we obtain that:

$$c(x) = c(\sigma_{i_1}) + c(\sigma_{i_2}) + \ldots + c(\sigma_{i_k})(\forall x)(x \in \Sigma^*) \quad . \tag{2.21}$$

The theorem has thus been proved. □

Definition 9

A composite cost k is a measure function of $(S \times \Sigma)^*$ into $\underline{R}$ such that: □

$$k(xy) = k(x) + k(y)(\forall x)(\forall y)((x \in (Sx\Sigma)^*) \wedge (y \in (Sx\Sigma)^*)) \quad . \tag{2.22}$$

Theorem 5.

We have that:

$$k(\varepsilon) = 0 \quad . \tag{2.23}$$

Proof. The proof is similar to the proof of Theorem 2. □

Theorem 6.

A composite cost k is completely determined by its restriction to S x Σ.

Proof. The proof is similar to the proof of Theorem 4. □

We conclude this introductory section by outlining the close relation existing between our definition of problem and the classical definitions of automaton and graph.

Definition 10.

A (deterministic) automaton A is a quintuple $A = (S,\Sigma,M,s_o,F)$ where,

$S = \{s_o,\ldots,s_{n-1}\}$ is a finite set of elements called the states of A;

$\Sigma = \{\sigma_o,\ldots,\sigma_{m-1}\}$ is a finite set of elements called the inputs of A;

$M = \{M_{\sigma_o},\ldots,M_{\sigma_{m-1}}\}$ is a finite set of mappings of S into S;

$s_o \subseteq S$ is called the initial state;

$F \subseteq S$ is called the set of final states. □

Definition 11.

A (directed, labeled) graph is a triple G = (V,A,R) where,

$V = \{v_o,\ldots,v_{n-a}\}$ is a finite set of elements called the vertices of G;

$A = \{a_o,\ldots,a_{m-1}\}$ is a finite set of elements called the labels of G;

$R = \{R_{a_o},\ldots,R_{a_{n-1}}\}$ is a finite set of mappings from V into V. □

The following two definitions illustrate the relations existing between the theories of problems, automata, and graphs.

Definition 12.

The automaton associated to the problem $P = (S,\Sigma,\Gamma,i,f)$ is the automaton $A = (S,\Sigma,\Gamma,i,\{f\})$. □

Definition 13

The graph associated to the problem $P = (S,\Sigma,\Gamma,i,f)$ is the graph $G = (S,\Sigma,\Gamma)$. □

In the next sections, we shall introduce new concepts based on the notions which have been introduced here. In this way, the results of the developing theory of problems, useful for obtaining concrete applications, will be exposed in a gradual and natural way.

3. The Resolvent Set H

In this section, we investigate in detail the state-space S with the purpose of supplying it with an appropriate framework, utilized for developing considerations on the problem-solving methodology.

Definition 14

A state $s_j \in S$ is reachable from the state $s_i \in S$ if:

$$(\exists x)((x \in \Sigma^*) \wedge (s_i \gamma_x = s_j)) \quad . \tag{3.1}$$

□

Definition 15

A state $s_j \in S$ is k-reachable from the state $s_i \in S$ if:

$$(\exists x)((x \in \Sigma^*) \wedge (s_i \gamma_x = s_j) \wedge (l(x) = k)) \quad . \tag{3.2}$$

□

Definition 16

The reachable set from the state $s_i \in S$ is the set $R_{s_i} \subseteq S$ such that:

$$R_{s_i} = \left\{ s_j \,\middle|\, (s_j \in S) \wedge (\exists x)((x \in \Sigma^*) \wedge (s_i \gamma_x = s_j)) \right\} \quad . \tag{3.3}$$

□

Definition 17

The k-reachable set from the state $s_i \in S$ is the set $R^k_{s_i} \subseteq S$ such that:

$$R^k_{s_i} = \left\{ s_j \,\middle|\, (s_j \in S) \wedge (\exists x)((x \in \Sigma^*) \wedge (s_i \gamma_x = s_j) \wedge (l(x) = k)) \right\} \quad . \tag{3.4}$$

□

Theorem 7.

We have that:

$$R_{s_i} = \bigcup_{k=0}^{\infty} R^k_{s_i} \quad . \tag{3.5}$$

□

Proof. The proof is directly obtained from Definitions 16 and 17. □

Definition 18

The reachable set of the problem $P = (S,\Sigma,\Gamma,i,f)$ is the set $R \subseteq S$ such that $R = R_i$. □

Definition 19

The k-reachable set of the problem $P = (S,\Sigma,\Gamma,i,f)$ is the set $R^k \subseteq S$ such that $R^k = R^k_i$. □

Definition 20

A state $s_j \in S$ is generating of the state $s_i \in S$ if:

$$(\exists x)((x \in \Sigma^*) \wedge (s_j \gamma_x = s_i)) \quad . \tag{3.6}$$

□

Definition 21

A state $s_j \in S$ is k-generating of the state $s_i \in S$ if:

$$(\exists x)((x \in \Sigma^*) \wedge (s_j \gamma_x = s_i) \wedge (l(x) = k)) \quad . \tag{3.7}$$

□

Definition 22

The generating set of the state $s_i \in S$ is the set $0_{s_i} \subseteq S$ such that:

$$0_{s_i} = \left\{ s_i \,\middle|\, (s_j \in S) \wedge (\exists x)((x \in \Sigma^*) \wedge (s_j \gamma_x = s_i)) \right\} \quad . \tag{3.8}$$

□

Definition 23

The k-generating set of the state $s_i \in S$ is the set $0^k_{s_i} \subseteq S$ such that:

$$0^k_{s_i} = \left\{ s_j \,\middle|\, (s_j \in S) \wedge (\exists x)((x \in \Sigma^*) \wedge (s_j \gamma_x = s_i) \wedge (l(x) = k)) \right\} \quad . \tag{3.9}$$

□

Theorem 8.

We have that:

$$0_{s_i} = \bigcup_{k=0}^{\infty} 0^k_{s_i} \quad . \tag{3.10}$$

Proof. The proof is directly obtained from Definitions 22 and 23.

□

Definition 24

The generating set of the problem $P = (S,\Sigma,\Gamma,i,f)$ is the set $0 \subseteq S$ such that $0 = 0_f$.

□

Definition 25

The k-generating set of the problem $P = (S,\Sigma,\Gamma,i,f)$ is the set $0^k \subseteq S$ such that $0^k = 0^k_f$.

□

We now present two necessary and sufficient conditions for the solvability of a problem.

Theorem 9.

A problem $P = (S,\Sigma,\Gamma,i,f)$ is solvable if $f \in R$.

Proof.

a) Only if part.

P is solvable, hence it has a solution x, i.e., because of Definition 4:

$$(\exists x)((x \in \Sigma^*) \wedge (i\gamma_x = f)) \quad . \tag{3.11}$$

Therefore $f \in R_i$ and $f \in R$.

b) If part.

$f \in R$, hence, because of Definition 18, $f \in R_i$. Hence,

$$(\exists x)((x \in \Sigma^*) \wedge (i\gamma_x = f)) \quad . \tag{3.12}$$

Therefore, because of Definition 4, x is a solution of P, and P is solvable. □

Theorem 10.

A problem $P = (S,\Sigma,\Gamma,i,f)$ is solvable if $i \in 0$.

Proof. The proof is similar to the proof of Theorem 9. □

Let us now define the resolvent set H and outline its importance and its algebraic properties.

Definition 26

The resolvent set of the problem $P = (S,\Sigma,\Gamma,i,f)$ is the set $H \subseteq S$ such that:

$$H = R \cap 0 \quad . \tag{3.13}$$

□

Definition 27

A k-step solution of the problem $P = (S,\Sigma,\Gamma,i,f)$ is a solution $x \in \Sigma^*$ of P such that $l(x) = k$. □

Definition 28

The (k-step) solution sequence generated by the (k-step) solution $x = \sigma_{i_1} \ldots \sigma_{i_k}$ of the problem $P = (S,\Sigma,\Gamma,i,f)$ is the sequence of states:

$$G_x^{(k)} = (i,s_1,s_2,\ldots,s_{k-1},f) \quad , \tag{3.14}$$

such that:

$$
\begin{aligned}
s_1 &= i\ \gamma\sigma_{i_1} \\
s_2 &= s_1\ \gamma\sigma_{i_2} \\
&\cdot \\
&\cdot \\
&\cdot \\
s_{k-1} &= s_{k-2}\ \gamma\sigma_{i_{k-1}} \\
f &= s_{k-1}\ \gamma\sigma_{i_k} \quad .
\end{aligned}
\tag{3.15}
$$

□

Theorem 11.

Each state $g_x^{(k)}$ of every k-step solution sequence $G_x^{(k)}$ of a problem $P = (S,\Sigma,\Gamma,i,f)$ belongs to H.

Proof. Because of Definitions 19, 25, and 28 we have that:

$$g_x^{(k)} \in \bigcup_{n=0}^{k} R^n \subseteq R \quad , \tag{3.16}$$

and

$$g_x^{(k)} \in \bigcup_{n=0}^{k} 0^n \subseteq 0 \quad . \tag{3.17}$$

Hence,

$$g_x^{(k)} \in R \cap 0 \text{ and, therefore, } g_x^{(k)} \in H \quad .$$

□

Theorem 12.

Every state $h \in H$ of the resolvent set H of a problem $P = (S,\Sigma,\Gamma,i,f)$ belongs at least to one k-step solution sequence of P.

Proof. For each element $h \in H$, we have that:

$$(h \in R) \wedge (h \in 0) \quad . \tag{3.18}$$

Hence, we draw that:

$$(\exists x_1)((x_1 \in \Sigma^*) \wedge (i\gamma_{x_1} = h)) \quad , \tag{3.19}$$

and,

$$(\exists x_2)((x_2 \in \Sigma^*)(h\gamma_{x_2} = f)) \quad . \tag{3.20}$$

Let us consider:

$$x = x_1 x_2 \quad . \tag{3.21}$$

Then we obtain that:

$$(\exists x)((x \in \Sigma^*) \wedge (i\gamma = f)) \quad . \tag{3.22}$$

Therefore, x is a solution of P. Let us suppose that:

$$l(x_1) = k_1 \quad , \quad l(x_2) = k_2 \quad , \quad l(x) = k_1 + k_2 = k \quad .$$

The k-step solution sequence $G_x^{(k)}$ is such that:

$$s_{k_1} = i\gamma_{x_i} = h \quad , \quad \text{hence } h \in G_x^{(k)} \quad .$$

□

Theorem 13.

Given a problem $P = (S, \Sigma, \Gamma, i, f)$ we have that:

$$H = \left\{ h \,\middle|\, h \text{ is a state of a solution sequence } G_x \right\} \quad . \tag{3.23}$$

Proof. Because of Theorems 11 and 12 we have that $(\forall x)$ $((h \text{ is a state of a solution sequence } G_x) \rightarrow (h \in H))$ because of Theorem 11. (3.24)

Moreover, $(\forall h)((h \in H) \rightarrow (h$ is a state of a solution sequence $G_x))$ because of Theorem 12. Hence, we obtain that: (3.25)

$$H = \left\{ h \,\middle|\, h \text{ is a state of a solution sequence } G_x \right\} \quad . \tag{3.26}$$

□

This last theorem emphasizes the importance of the resolvent set H which appears as the natural base for further investigation of the properties of the state-space representation. We point out that a bidirectional algorithm incrementally building up the resolvent set H can easily be defined on the ground of Definition 26.

We can now further investigate some algebraic properties of H.

Definition 29

An arc of a problem $P = (S,\Sigma,\Gamma,i,f)$ is a couple of states $u = (s_o,s_1)$ such that:

$$(\exists\sigma)((\sigma \in \Sigma) \wedge (s_o \gamma\sigma = s_1)) \quad . \tag{3.27}$$

The state s_o is called the initial state, and the state s_1 is called the final state of u. □

Definition 30

A loop is an arc $u = (s_o,s_1)$ such that $S_o = S_1$. □

Definition 31

A path of a problem $P = (S,\Sigma,\Gamma,i,f)$ is a finite sequence of states $\mu = (s_o,\ldots,s_k)$ such that:

$$(\exists x)((x \in \Sigma^+) \wedge (x = \sigma_{i_1} \ldots \sigma_{i_k}) \wedge (s_1 = s_o \gamma\sigma_{i_1})$$

$$\wedge \ldots \wedge (s_k = s_{k-1} \gamma\sigma_{i_k})) \quad , \tag{3.28}$$

where,

$$\Sigma^+ = \Sigma^* - \{\varepsilon\} \quad . \tag{3.29}$$

Such string x is called a generating string of μ. □

Definition 32

The length of a path $\mu = (s_o,\ldots,s_k)$ is the length of a generating string of it. Please note that while any path can have more than one generating string of itself, the lengths of all these generating strings are all equal among themselves. □

Definition 33

A circuit is a path $\mu = (s_o,\ldots,s_k)$ such that $s_o = s_k$. □

Definition 34

A ring is a path $\mu = (s_o,s_1,\ldots,s_k)$ such that:

$$s_o = s_1 = \ldots = s_k \quad . \tag{3.30}$$

□

Definition 35

Given a problem $P = (S,\Sigma,\Gamma,i,f)$, we define the binary relation $\leq$ on H, in this way:

$$h_i \leq h_j \quad , \tag{3.31}$$

whenever,

$$h_i \in R_{h_j} \quad (\forall h_i)(\forall h_j)((h_i \in H) \wedge (h_j \in H)) \quad . \tag{3.32}$$

□

Theorem 14.

The binary relation $\leq$ on H is a partial preorder.

Proof. We have that:

$$h \in = h \quad (\forall h)(h \in H) \quad , \tag{3.33}$$

and

$$h \in R_h \quad (\forall h)(h \in H) \quad . \tag{3.34}$$

Therefore,

$$h \leq h \quad (\forall h)(h \in H) \quad . \tag{3.35a}$$

Hence, the reflexive property holds. Moreover, if

$$(h_i \leq h_j) \wedge (h_j \leq h_k) \quad , \tag{3.35b}$$

we have that,

$$(h_i \in R_{h_j})(h_j \in R_{h_k}) \quad , \tag{3.36}$$

and,

$$(\exists x_1)((x_1 \in \Sigma^*) \wedge (h_j \gamma \sigma_{x_1} = h_i)) \quad , \tag{3.37}$$

and,

$$(\exists x_2)((x_2 \in \Sigma^*) \wedge (h_k \gamma \sigma_{x_1} = h_j)) \quad . \tag{3.38}$$

Therefore, if $x = x_2 x_1$, then $h_k \sigma_x = h_i$.

Therefore, $h_i \in R_{h_k}$, and, hence, $h_i \leq h_k$.

Therefore, we have proved that the transitivity property holds. □

Theorem 15.

Given a problem $P = (S, \Sigma, \Gamma, i, f)$ if a state $h \in H$ is an element of a circuit of P, then all the elements of the circuit belong to H.

Proof. Let be $h \in H$, and $\mu = (s_1, \ldots, s_n, h, s_{n+1}, \ldots, s_k)$ with $s_1 = s_k$ (i.e., μ is a circuit of P).

Because of Definition 26, we have that $h \in R$, and, hence, $\{s_{n+1}, \ldots, s_k\} \subseteq R$; moreover, we have that $h \in 0$, and, hence, $\{s_1, \ldots, s_n\} \subseteq 0$. But since $s_1 = s_k$, we have that $\{s_{n+1}, \ldots, s_k\} \subseteq 0$,

and we have that $\{s_1,\ldots,s_n\} \subseteq R$. Therefore, we conclude that for each element s_μ of μ, it holds that,

$$(s_\mu \in R) \wedge (s_\mu \in 0) \quad . \tag{3.39}$$

Therefore, each element s_μ of μ belongs to H. □

Theorem 16.

The binary relation $\leq$ on H is a partial order if H doesn't contain circuits (it may contain loops and rings).

Proof. In Theorem 14, we have already proved that reflexivity and transitivity properties hold for $\leq$. We must now prove that antisymmentry property holds. We will prove such property if H doesn't contain circuits.

a) Only if part.

We will prove that if antisymmetry property holds, then H doesn't contain circuits. We will prove this case by absurdum. In fact, let us suppose that antisymmetry property holds, i.e.:

$$(((h_1 \leq h_2) \wedge (h_2 \leq h_1)) \rightarrow (h_1 = h_2)) \quad , \tag{3.40}$$

$$(\forall h_1)(\forall h_2)((h_1 \in H) \wedge (h_2 \in H)) \quad ,$$

and, let us suppose, by absurdum that it exists, a circuit of H, $\mu = (s_1, s_2, \ldots, s_{k-1}, s_1)$. We will have, under this assumption and because of Definition 35, that both $(s_1 \leq s_2)$ and $(s_2 \leq s_1)$. But, also, $s_1 \neq s_2$; so the antisymmetry property has been contradicted.

b) If part.

We will prove that if H doesn't contain circuits, the antisymmetry property holds. We will prove this case by absurdum. Let us suppose that H doesn't contain circuits. And, let us suppose, by absurdum, that the antisymmetry property doesn't hold, i.e.:

$$(h_1 \leq h_2) \wedge (h_2 \leq h_1) \wedge (h_1 \neq h_2) \quad . \tag{3.41}$$

Because of Definition 35, we have that:

$$(h_1 \in R_{h_2}) \wedge (h_2 \in R_{h_1}) \wedge (h_1 \neq h_2) \quad . \tag{3.42}$$

□

Hence, it exists, by construction, as a circuit $\mu = (h_2,\ldots, h_1,\ldots,h_2)$ which contradicts the hypothesis that H doesn't contain circuits.

Theorem 17.

Given a problem $P = (S,\Sigma,\Gamma,i,f)$, if H doesn't contain circuits, then i is the maximum for the partially ordered set $(H,\leq)$.

Proof. We have that:

$$h \in R_i \quad (\forall h)(h \in H) \quad , \tag{3.43}$$

and therefore, because of Definition 35:

$$h \leq i \quad (\forall h)(h \in H) \quad . \tag{3.44}$$

□

Theorem 18.

Given a problem $P = (S,\Sigma,\Gamma,i,f)$, if H doesn't contain circuits, then f is the minimum for the partially ordered set $(H,\leq)$.

Proof. The proof is similar to the proof of Theorem 17. □

We outline that even if H doesn't contain circuits the couple $(H,\leq)$ is not, in general, a lattice. We can illustrate such fact by means of an example. In Figure 4 we show the diagram of a particular resolvent set $\bar{H}$ and of its partial order $\leq$, which is not a lattice. For instance, it is clear that it doesn't exist at least upper bound for the two states h_i and h_j; also it doesn't exist at greatest lower bound for the two states k_i and k_j.

Definition 36

In correspondence of every (k-step) solution sequence $G_x^{(k)} = (i,s_1,\ldots,s_{k-1},f)$ which doesn't contain circuits, we associate a (k-step) solution set $\mathcal{G}_x^{(k)} = \{i,s_1,\ldots,s_{k-1},f\}$. □

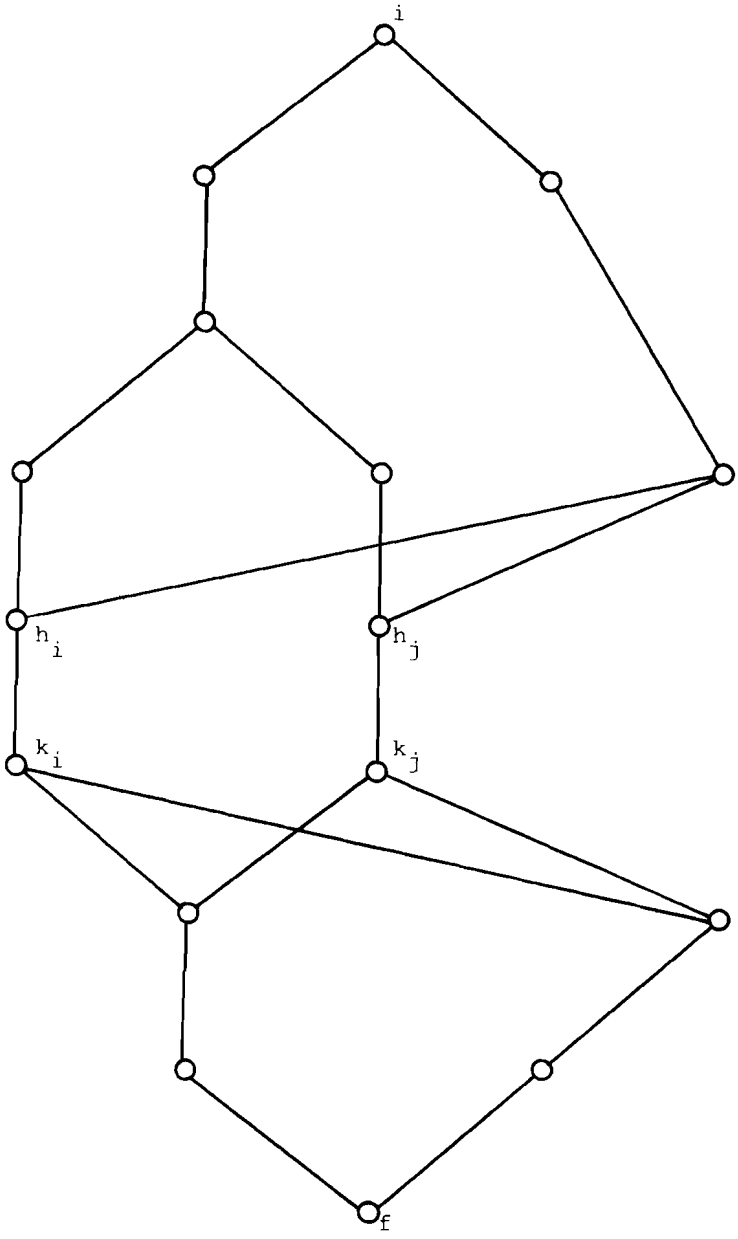


Figure 4. Diagram for a particular resolvent set $\overline{H}$.

Theorem 19.

The binary relation $\leq$ defined on H is a total order on every (k-step) solution set $\mathcal{G}_x^{(k)} = \left\{i, s_1, \ldots, s_{k-1}, f\right\}$.

Proof: We have that, $\mathcal{G}_x^{(k)}$ is a subset of H, i.e.:

$$\mathcal{G}_x^{(k)} \subseteq H \quad , \tag{3.45}$$

because of Theorem 11. Moreover, $\leq$ is a total order on $\mathcal{G}_x^{(k)}$ because of Definitions 28 and 35. □

Theorem 20.

The couple $(\mathcal{G}_x^{(k)}, \leq)$ is a lattice.

Proof. The proof is obvious because of the properties of set union and set intersection. □

We outline that we can easily construct an algorithm which allows one to "open" all the circuits which could possibly exist in the resolvent set H if they happen to exist, so that a set H* without circuits could be deduced.

Definition 37

Given a problem $P = (S, \Sigma, \Gamma, i, f)$, we define the distance d, of two elements $h \in H$ and $k \in H$ of the resolvent set H, as a mapping from $H \times H$ into N (set of natural numbers) such that:

d (h,k) is the minimum of the lengths of the paths from h to K;

d (h,k) = d (k,h) = 0, if k = h. □

It is obvious that the above defined distance d doesn't exist for every couple (h,k) such that $h \in H$ and $h \in K$.

Theorem 21.

The distance d (h,k) of two states $h \in H$ and $k \in H$ exists if $k \leq h$.

Proof. The proof can be easily obtained because of Definitions 31, 35 and 37. □

Definition 38

Given a problem $P = (S,\Sigma,\Gamma,i,f)$, the height g of a state $h \in H$ is defined in the following way:

$$g(h) = d(h,f) \quad (\forall h)(h \in H) \quad . \tag{3.46}$$

□

Definition 39

Given a problem $P = (S,\Sigma,\Gamma,i,f)$, the depth p of a state $h \in H$ is defined in the following way:

$$p(h) = d(i,h) \quad (\forall h)(h \in H) \quad . \tag{3.47}$$

□

Theorem 22.

For every state $h \in H$, there exists a unique depth and a unique height of h.

Proof. The proof can be easily obtained because of Definition 37 and because of Theorem 21. □

The last two definitions introduce in a natural way the concept of "level" in the resolvent set H. The great importance of H as the set which contains all and only the states which belong to solution sequences was already shown in Theorems 11, 12, and 13. We now briefly outline the two most important reasons for defining levels in H.

The first reason is that the existence of levels in H allows an easier and more natural definition of the "intersection" point and an easier "welding" in the bidirectional search algorithm. Intersection and welding are, in fact, the most critical topics in the design of any bidirectional algorithm.

The second reason is that, if we look at a search strategy in the statespace as at a kind of wave expanding from i or from f into the resolvent set H, then the above-defined levels allow us to set a quantitative measure of the depth and of the breadth of the wave front at every instant during the search process.

4. Optimality

In this section, we investigate the problems which arise when costs are taken into account. It is possible in this way to investigate the quality of the solutions which are obtained. In particular, the property of existence of "good" solutions is illustrated. We first briefly investigate the cardinality of

the solution set X_p of a problem P. Clearly X_p is not finite in general although H is always finite.

Theorem 23.

The solution set $X_p \subseteq \Sigma^*$ of a problem $P = (S,\Sigma,\Gamma,i,f)$ is finite if H doesn't contain circuits.

Proof. The proof is a direct consequence of the finiteness of the resolvent set H. □

Theorem 24.

Given a problem $P = (S,\Sigma,\Gamma,i,f)$, the solution set X_p is a countable set.

Proof. We have that $X_p \in \Sigma^*$. Since Σ^* is a countable set, then X_p is a countable set as well. □

Definition 40

A solution $\bar{x} \in \Sigma^*$ of a problem $P = (S,\Sigma,\Gamma,i,f)$ is minimal if:

$$l(\bar{x}) = \min_{x \in X_p} \left\{ l(x) \right\} . \qquad (4.1)$$

□

Definition 41

A solution $\bar{x} \in \Sigma^*$ of a problem $P = (S,\Sigma,\Gamma,i,f)$ is simply (compositely) optimal for a simple cost c (composite cost k) if:

$$c(\bar{x}) = \min_{x \in X_p} \left\{ c(x) \right\} \quad (k(\bar{x}) = \min_{x \in X_p} \left\{ k(x) \right\} . \qquad (4.2)$$

□

We outline that, in general, neither the existence nor the uniqueness of a simply (compositely) optimal or minimal solution of a problem can be proved.

Theorem 25.

Given a problem $p = (S,\Sigma,\Gamma,i,f)$, if $X_p \neq \emptyset$ and H doesn't contain circuits, at least one minimal and one simply (compositely) optimal solution of P exists.

Proof. The proof is a direct consequence of Theorem 23 which states the finiteness condition of X_p. □

Theorem 26.

Given a problem $P = (S,\Sigma,\Gamma,i,f)$ if $X_p \neq \emptyset$, at least one minimal solution of P always exists.

Proof. The proof is a direct consequence of Definitions 7 and 40. □

Theorem 27.

Given a problem $P = (S,\Sigma,\Gamma,i,f)$, if $X_p \neq \emptyset$ and the simple (composite) cost c (κ) has values only in R^+ (nonnegative real numbers), at least one simply (compositely) optimal solution of P always exists.

Proof. The proof is similar to the proof of Theorem 26. □

Definition 42.

Given a problem $P = (S,\Sigma,\Gamma,i,f)$, the minimal resolvent set H' of P is the set of all states of the resolvent set H which are in a (k-step) solution sequence $G\binom{(k)}{\bar{x}}$, where $\bar{x}$ is a minimal solution of P. □

Theorem 28.

The minimal resolvent set H' doesn't contain circuits.

Proof.

The proof is a direct consequence of Definitions 40 and 42, and of the property of minimum. In fact, the existence of a circuit in a solution x would yield another solution x' obtained from x by excluding the circuit. Hence, the length of x' would be less than the length of x, which is against the assumption that x is a minimal solution of P. □

Theorem 29.

The binary relation $\leq'$ defined as the restriction of $\leq$ on H' is a partial order on H'.

Proof. The proof of the theorem, because of Theorem 28, is a direct consequence of Definition 35 and of Theorem 16. We outline that the partially ordered set $(H', \leq')$ is not in general a lattice. □

We can illustrate such fact by means of an example. In Figure 5, we show the diagram of a particular minimal resolvent set H' and of its partial order $\leq'$, which is not a lattice. For instance, it is clear that it doesn't exist at least upper bound for the two states h'_i and h'_j; also it doesn't exist at greatest lower bound for the two states k'_i and k'_j.

Definition 43.

Given a problem $P = (S,\Sigma,\Gamma,i,f)$, the simply (compositely) optimal resolvent set H"(H"') is the set of all states of the resolvent set H which are in a (k-step) solution sequence $G_x^{(k)}$, when x is a simply (compositely) optimal solution of P. We outline that H" (H"'), in general, can contain circuits. □

Theorem 30.

The simply (compositely) optimal resolvent set H" (H"') doesn't contain circuits if the simple (composite) cost c (k) has values only in R^+ (set of the nonnegative real numbers).

Proof. The proof is similar to the proof of Theorem 28 (i.e., we can repeat, in the case of simple (composite) cost c (k) which has values only in R^+ (set of nonnegative real numbers), the same considerations related to the notion of length, made in the proof of Theorem 28). □

We have investigated in this section the conditions for the existence of "good" solutions of problems, which are, of course, the most interesting ones from a practical point of view. Algorithms can be defined which allow one to test, in an efficient way, the "goodness" of a solution or to search directly only the "good" solutions, as well.

5. Morphisms

In this section, we will outline some preliminary results which are related to the problem of comparing different representations of problems.

More precisely, in this section we investigate the conditions which allow us to consider two problems, although obtained by semantic domains of different nature, as substantially equivalent with respect to their algebraic (or syntactic) structure, and with respect to the effort which is required to solve them (when the heuristic aspect is ignored).

In the following sections, the exigence of stating in a formal way some criteria for comparing different problem representations will be examined with more detail, in the direction of achieving a unitary presentation of different approaches to problem solving.

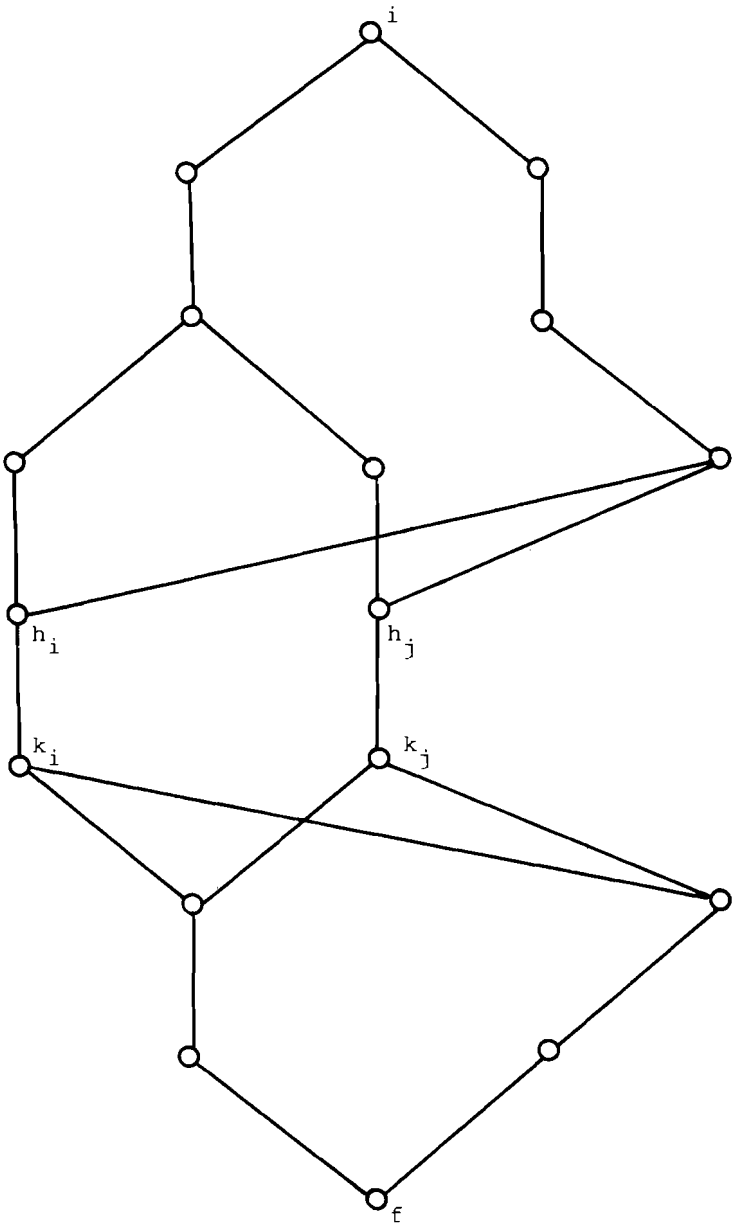


Figure 5. Diagram for a particular minimal resolvent set $\bar{\Pi}'$.

In this way, the different alternatives which are offered to a problem solver's activity are examined, compared, and exposed, as, substantially, the operation made of a representation language interpreter.

Definition 44

The problem $P" = (S",\Sigma",\Gamma",i",f")$ is a homomorphic image of the problem $P' = (S',\Sigma',\Gamma',i',f')$ if there exists a mapping ϕ of S' onto $S"$ and a mapping ξ of Σ' onto $\Sigma"$ such that:

$$\gamma\sigma'\phi = \phi\gamma\sigma'\xi \quad (\forall\sigma')(\sigma' \in \Sigma') \quad , \tag{5.1}$$

and,

$$i'\phi = i" \quad , \quad f'\phi = f" \quad . \tag{5.2}$$

The couple (ϕ,ξ) is said as a homomorphism of P' onto $P"$. □

Theorem 31.

Given a homomorphism (ϕ,ξ) of a problem $P' = (S',\Sigma',\Gamma',i',f')$ onto a problem $P" = (S",\Sigma",\Gamma",i",f")$ if ξ is one-to-one, then $\Sigma"$ can be renamed in such a way as to become equal to Σ'; in this way ξ is reduced to the identity function.

Proof. The proof is a direct consequence of Definition 44. □

Theorem 32.

Given a homomorphism (ϕ,ξ) of a problem $P' = (S',\Sigma',\Gamma',i',f')$ onto a problem $P" = (S",\Sigma",\Gamma",i",f")$ if ϕ is one-to-one, then $S"$ can be renamed in such a way as to become equal to S'; in this way ϕ is reduced to the identity function.

Proof. The proof is similar to the proof of Theorem 31. □

Definition 45

The problem $P" = (S",\Sigma",\Gamma",i",f")$ is an isomorphic image of the problem $P' = (S',\Sigma',\Gamma',i',f')$ if there exists an homomorphism (ϕ,ξ) of P' onto $P"$ such that both ϕ and ξ are one-to-one mappings. The couple (ϕ,ξ) is said as an isomorphism of P' onto $P"$. □

Theorem 33.

If two problems P' and $P"$ are isomorphic, the first one can be obtained from the second one by merely renaming the states

and the inputs.

Proof. The proof is a direct consequence of Theorems 31 and 32 and Definition 45. □

The results illustrated in this section are simply preliminary, and merely constitute a sketched outline for further investigations. However, they appear to be very general for their implications, and can be specified, in some particular cases of semantic domains, in order to enhance their significance.

6. Relations Between Problems

In this section, we introduce general notions and criteria designed to confront and compare different problems represented within various approaches to problem solving (namely state-space approach and problem-reduction approach). Different types of relations between problems are illustrated, and properties based on these notions are presented. The formal framework introduced here constitutes the substantial background on which, in the following section, we have founded a unitary description of the theory of problem solving, which constitutes the goal of the whole paper, and which, at the same time, provides the most useful results which have been achieved.

Definition 46

A source problem is a quadruple $U = (S,\Sigma,\Gamma,i)$ where:

(S,Σ,Γ) is a problem schema;

$i \in S$ is a particular element of S called the initial state.

□

Definition 47

An object problem is a quadruple $B = (S,\Sigma,\Gamma,f)$ where:

(S,Σ,Γ) is a problem schema;

$f \in S$ is a particular element of S called the final state.

□

Definition 48

Two problems $P = (S,\Sigma,\Gamma,i,f)$ and $P' = (S',\Sigma',\Gamma',i',f')$ are similar iff they share the same problem schema, i.e., iff:

$$S = S' \quad , \quad \Sigma = \Sigma' \quad , \quad \Gamma = \Gamma' \quad . \tag{6.1}$$

In this case, we shall write:

$$P \simeq P' \quad . \tag{6.2}$$

□

Definition 49

Two problems $P = (S,\Sigma,\Gamma,i,f)$ and $P' = (S',\Sigma',\Gamma',i',f')$ are i-similar iff they share the same source problem, i.e., iff:

$$S = S' \quad , \quad \Sigma = \Sigma' \quad , \quad \Gamma = \Gamma' \quad , \quad i = i' \quad . \tag{6.3}$$

In this case, we shall write:

$$P \overset{i}{\simeq} P' \quad . \tag{6.4}$$

□

Definition 50

Two problems $P = (S,\Sigma,\Gamma,i,f)$ and $P' = (S',\Sigma',\Gamma',i',f')$ are f-similar iff they share the same object problem, i.e., iff:

$$S = S' \quad , \quad \Sigma = \Sigma' \quad , \quad \Gamma = \Gamma' \quad , \quad f = f' \quad . \tag{6.5}$$

In this case, we shall write:

$$P \overset{f}{\simeq} P' \quad . \tag{6.6}$$

□

Definition 51

Two problems $P = (S,\Sigma,\Gamma,i,f)$ and $P' = (S',\Sigma',\Gamma',i'f')$ are equal iff:

$$S = S',\ \Sigma' = \Sigma',\ \Gamma = \Gamma',\ i = i',\ f = f' \quad . \tag{6.7}$$

In this case, we shall write:

$$P = P' \quad . \tag{6.8}$$

□

Definition 52

The universal problem set $\mathcal{P}$ is the set of all the problems, i.e.:

$$\mathcal{P} = \left\{ P \,\middle|\, P = (S,\Sigma,\Gamma,i,f) \right\} \quad . \tag{6.9}$$

□

Theorem 34.

The binary relations $\simeq$, $\overset{i}{\simeq}$, $\overset{f}{\simeq}$, $=$, between problems, are equivalence relations on the universal problem set $\mathcal{P}$.

Proof. The proof is obtained directly because each one of the four binary relations on $\mathcal{P}$ $\simeq$, $\overset{i}{\simeq}$, $\overset{f}{\simeq}$, and $=$ is based on the notion of equality between sets which is a well-known equivalence relation.

□

Definition 53

A state-subproblem P' of a problem $P = (S,\Sigma,\Gamma,i,f)$ is a problem $P' = (S',\Sigma',\Gamma',i',f')$ such that:

$$\begin{aligned} &S' \subseteq S; \\ &\Sigma' = \Sigma; \\ &\Gamma' = \Gamma \text{ restricted to } S'; \\ &i' \in S', \quad f' \in S'. \end{aligned} \tag{6.10}$$

We shall write:

$$P' \overset{S}{\sqsubseteq} P \quad . \tag{6.11}$$

If,

$$S' \subset S \quad , \tag{6.12}$$

then P' is said as a proper state-subproblem of P, and we shall write:

$$P' \overset{S}{\sqsubset} P \quad . \tag{6.13}$$

□

Definition 54

An input-subproblem P" of a problem $P = (S,\Sigma,\Gamma,i,f)$ is a problem $P'' = (S'',\Sigma'',\Gamma'',i'',f'')$ such that:

$$\begin{aligned} &S'' = S; \\ &\Sigma'' \subseteq \Sigma; \\ &\Gamma'' \subseteq \Gamma; \\ &i'' = i, \quad f'' = f \quad . \end{aligned} \tag{6.14}$$

We shall write:

$$P'' \overset{i}{\sqsubseteq} P \quad . \tag{6.15}$$

If,

$$(\Sigma'' \subset \Sigma) \wedge (\Gamma'' \subset \Gamma) \quad , \tag{6.16}$$

then P" is said as a proper input-subproblem of P, and we shall write:

$$P'' \overset{i}{\sqsubset} P \quad . \tag{6.17}$$

□

Definition 55

A (input-state) subproblem P"' of a problem $P = (S,\Sigma,\Gamma,i,f)$ is a problem $P''' = (S''',\Sigma''',\Gamma''',i''',f''')$ such that:

$$S''' \subseteq S; \tag{6.18}$$
$$\Sigma''' \subseteq \Sigma;$$
$$\Gamma''' \subseteq \Gamma;$$
$$i''' \in S''', \quad f''' \in S''' \quad .$$

We shall write:

$$P''' \sqsubseteq P \quad . \tag{6.19}$$

If,

$$(S''' \in S) \wedge ((\Sigma''' \in \Sigma) \wedge (\Gamma''' \in \Gamma)) \quad , \tag{6.20}$$

then P"' is said as a proper (input-state) subproblem of P, and we shall write:

$$P''' \sqsubset P \quad . \tag{6.21}$$

□

Theorem 35.

The binary relations $\overset{s}{\sqsubseteq}$, $\overset{i}{\sqsubseteq}$, $\sqsubset$ among problems are partial ordering relations of the universal problem set $\mathcal{P}$.

Proof. The proof is obtained directly because each one of the three binary relations on $\mathcal{P}$ $\overset{s}{\sqsubseteq}$, $\overset{i}{\sqsubseteq}$, and $\sqsubseteq$ is based on the notion of inclusion between sets which is a well-known partial ordering relation. □

Definition 56

The null problem P is a problem $P_\nu = (S_\nu, \Sigma_\nu, \Gamma_\nu, i_\nu, f_\nu)$ where:

$$S_\nu = \Sigma_\nu = \Gamma_\nu = \emptyset \quad , \tag{6.22}$$

and, therefore, i_ν and f_ν don't exist. □

Definition 57

An empty problem P_ε is a problem $P_\varepsilon = (S_\varepsilon, \Sigma_\varepsilon, \Gamma_\varepsilon, i_\varepsilon, f_\varepsilon)$ where:

$$\Sigma_\varepsilon = \Gamma_\varepsilon = \emptyset \quad . \tag{6.23}$$

□

Definition 58

A complete problem P_I is a problem $P_I = (S_I, \Sigma_I, \Gamma_I, i_I, f_I)$ such that:

$$s_i \in R^1_{s_j} \quad (\forall s_i)(\forall s_j)((s_i \in S_I) \wedge (s_j \in S_I)) \quad . \tag{6.24}$$

□

Definition 59

A compact problem P_H is a problem $P_H = (S_H, \Sigma_H, \Gamma_H, i_H, f_H)$ such that:

$$S_H = H \quad . \tag{6.25}$$

□

Definition 60

A trivial problem P_τ is a problem $P_\tau = (S_\tau, \Sigma_\tau, \Gamma_\tau, i_\tau, f_\tau)$ where $i_\tau = f_\tau$. □

Theorem 36.

Given a trivial problem $P_\tau = (S_\tau, \Sigma_\tau, \Gamma_\tau, i_\tau, f_\tau)$, we have that:

$$\varepsilon \in X_P \quad . \tag{6.26}$$

□

Proof. The proof is a direct consequence of Definitions 4 and 6.

We present now a very general definition which is comprehensive of all the possible relations between problems which can be based on the comparison of their solution sets.

Definition 61

An implicant of a problem P is a couple $L = (\pi, \psi)$ where:

$\pi = (P_1, P_2, \ldots, P_k)$ is a finite sequence of problems;

ψ is a mapping of $X_{P_1} \times X_{P_2} \times \ldots \times X_{P_k}$ into X_P.

We shall write:

$$\pi \overset{\psi}{\rightarrow} P \quad . \tag{6.27}$$

If ψ is a mapping onto X_P, then L is said as a full implicant of P. In this case, we shall write:

$$\pi \overset{\psi}{\rightarrow} P \quad . \tag{6.28}$$

□

Let us now present some particularly important and usual cases of this general definition.

Let us suppose that $\pi = (P_1)$ and that ψ is a one-to-one mapping I of X_{P_1} onto X_P defined in the following way:

$$I\ (x_i^1) = x_i^1 \quad (\forall x_i^1)\,(x_i^1 \in X_{P_1}) \quad . \tag{6.29}$$

In this case, it is useful to introduce the following definition.

Definition 62

A problem P_1 is equivalent to a problem P iff $X_{P_1} = X_P$. We shall write:

$$P_1 \overset{I}{\leftrightarrow} P \quad . \tag{6.30}$$

□

Let us suppose that $\pi = (P_1)$ and that ψ is a one-to-one mapping I of X_P into X_P defined as in relation (6.29). In this case, it is useful to introduce the following definition.

Definition 63

A problem P_1 is dominant of a problem P iff $X_{P_1} = X_P$. We shall write:

$$P_1 \overset{I}{\leftrightarrow} P \quad . \tag{6.31}$$

□

Let us suppose that $\pi = (P_1, P_2, \ldots, P_k)$ and that ψ is a mapping C of $X_{P_1} \times X_{P_2} \times \ldots \times X_{P_k}$ onto X_P defined in the following way:

$$C(x^1_{i_1}, x^2_{i_2}, \ldots, x^k_{i_k}) = x^1_{i_1}, x^2_{i_2} \ldots x^k_{i_k} \tag{6.32}$$

$$(\forall x^1_{i_1})(\forall x^2_{i_2}) \ldots (\forall x^k_{i_k})((x^1_{i_1} \in X_{P_1})(x^2_{i_2} \in X_{P_2}) \ldots (x^k_{i_k} \in X_{P_k})) \quad .$$

In this case, it is useful to introduce the following definition.

Definition 64

A finite sequence of problems $\pi = (P_1, P_2, \ldots, P_k)$ is a <u>full covering</u> of a problem P iff:

$$X_{P_1} X_{P_2} \ldots X_{P_k} = X_P \quad , \tag{6.33}$$

where, in the first member of relation (6.33), we have used the operation of composition of sets of strings, defined in the following way:

$$X_{P_i}\ X_{P_j} = \left\{ x \,\middle|\, (x = x^i_{h_i}\ x^j_{h_j}) \wedge (x^i_{h_i} \in X_{P_i}) \wedge (x^j_{h_j} \in X_{P_j}) \right\} \quad . \tag{6.34}$$

We shall write:

$$(P_1, P_2, \ldots, P_k) \overset{C}{\rightarrow} P \quad . \tag{6.35}$$

□

Let us suppose that $\pi = (P_1, \ldots, P_k)$ and that ψ is a mapping C of $X_{P_1} \times X_{P_2} \times \ldots \times X_{P_k}$ into X_P defined as in relation (6.32).
In this case, it is useful to introduce the following definition.

Definition 65

A finite sequence of problems $\pi = (P_1, P_2, \ldots, P_k)$ is a partial covering of a problem P iff:

$$X_{P_1}\ X_{P_2} \ldots X_{P_k} = X_P \quad . \tag{6.36}$$

We shall write:

$$(P_1, P_2, \ldots, P_k) \overset{C}{\rightarrow} P \quad . \tag{6.37}$$

□

Definition 66

A solved problem is a couple T = (P,Y) where:

$P = (S, \Sigma, \Gamma, i, f)$ is a problem;

$Y \subset X_P$.

If $Y = X_P$, T is said as a completely solved problem.

□

The notion, here introduced, of a solved problem T is considered as a helpful tool when problems, which are related with T, have to be solved by utilizing the information contained in T. This implication with a type of learning activity, will be exposed with greater detail in the following section.

We now investigate some interesting properties of particular subsets of the universal problem set.

Definition 67

The subsidiary problem set of a problem P is the set $\mathcal{P}_P^S \subset \mathcal{P}$ such that:

$$\mathcal{P}_P^S = \left\{ P_i \,\middle|\, (P_i \in \mathcal{P}) \wedge (P_i \stackrel{S}{\sqsubseteq} P) \right\} \quad . \tag{6.38}$$

□

Definition 68

The auxiliary problem set of a problem P is the set $\mathcal{P}_P^i \subset \mathcal{P}$ such that:

$$\mathcal{P}_P^i = \left\{ P_i \,\middle|\, (P_i \in \mathcal{P}) \wedge (P_i \stackrel{i}{\sqsubseteq} P) \right\} \quad . \tag{6.39}$$

□

Definition 69

The coproblem set of a problem P is the set $\mathcal{P}_p \subset \mathcal{P}$ such that:

$$\mathcal{P}_p = \{ P_i \,|\, (P_i \in \mathcal{P}) \wedge (P_i \sqsubseteq P) \} \quad . \tag{6.40}$$

□

It is obvious that, because of the definitions of $\stackrel{S}{\sqsubseteq}$, $\stackrel{i}{\sqsubseteq}$, and $\sqsubseteq$, we have:

$$\mathcal{P}_P^S \subseteq \mathcal{P}_p \quad (\forall P)(P \in \mathcal{P}) \tag{6.41}$$

$$\mathcal{P}_P^i \subseteq \mathcal{P}_p \quad (\forall P)(P \in \mathcal{P}) \quad .$$

□

Theorem 37.

Given a problem P, the couple $(\mathcal{P}_P^i, \stackrel{iP}{\sqsubseteq})$, where $\stackrel{iP}{\sqsubseteq}$ is the restriction of the relation $\stackrel{i}{\sqsubseteq}$ over $\mathcal{P}$ to $\mathcal{P}_P^i$, is a partial order.

Proof. The proof is similar to the proof of Theorem 35. □

Theorem 38.

Given a problem P, the couple $(\mathcal{P}_P, \stackrel{P}{\sqsubseteq})$, where $\stackrel{P}{\sqsubseteq}$ is the restriction of the relation $\sqsubseteq$ over $\mathcal{P}$ to $\mathcal{P}_p$, is a partial order.

Proof. The proof is similar to the proof of Theorem 35. □

Theorem 39.

Given a problem P, the sets $\wp^s_P$, $\wp^i_P$, and $\wp_P$ are finite sets.

Proof. The proof is directly obtainable from the finiteness of S and from Definitions 53, 54, and 55. □

Theorem 40.

Given a problem P, P is the maximum for each one of the three partial orders $(\wp^s_P, \overset{sP}{\sqsubseteq})$, $(\wp^i_P, \overset{iP}{\sqsubseteq})$ and $(\wp_P, \overset{P}{\sqsubseteq})$.

Proof. The proof is directly obtainable because of Definitions 53, 54 and 55. □

Theorem 41.

Given a problem P, the null problem P_ν is the minimum for the partial orders $(\wp^s_P, \overset{sP}{\sqsubseteq})$ and $(\wp_P, \overset{P}{\sqsubseteq})$. □

Proof. The proof is directly obtainable because of Definitions 53, 55 and 56. □

Theorem 42.

Given a problem P, the empty problem P_ε is the minimum for the partial order $(\wp^i_P, \overset{iP}{\sqsubseteq})$.

Proof. The proof is directly obtainable because of Definitions 54 and 57. □

Theorem 43.

Given a problem P, the partial orders $(\wp^s, \overset{sP}{\sqsubseteq})$, $(\wp^i, \overset{iP}{\sqsubseteq})$ and $(\wp_P, \overset{P}{\sqsubseteq})$ are lattices.

Proof. The proof is obtained directly because each one of the three Sets $\wp^s_P$, $\wp^i_P$, and $\wp_P$ is based on the notion of power set on a given set, each one of the three binary relations $\overset{s}{\sqsubseteq}$, $\overset{i}{\sqsubseteq}$, $\sqsubseteq$ is based on the notion of inclusion between sets, and the power set on a given set is a well-known lattice under the relation of inclusion. □

Definition 70.

Given a problem P, we define the s-distance d^s of two problems $P_1 \in \wp_P^s$ and $P \in \wp_P^s$ in the following way: $d^s(P_1,P_2) = \min \{$lengths of the chains connecting P_1 to P_2 in the diagram of the lattice $(\wp_P^s, \overset{sP}{\sqsubseteq})\}$. □

Please note that the various definitions of distances, introduced in this paper and based on partial orders, have to be intended not in a topological way. We recall that the length of a chain in a diagram of a lattice is the number of arcs in it.

Definition 71

Given a problem P, we define the i-distance d^i of two problems $P_1 \in \wp_P^i$ and $P_2 \in \wp_P^i$ in the following way: $d^i(P_1,P_2) = \min \{$lengths of chains connecting P_1 to P_2 in the diagram of the lattice $(\wp_P^i, \overset{iP}{\sqsubseteq})\}$. □

Definition 72

Given a problem P, we define the p-distance d^P of two problems $P_1 \in \wp_P$ and $P_2 \in \wp_P$ in the following way: $d^P(P_1,P_2) = \min \{$lengths of chains connecting P_1 to P_2 in the diagram of the lattice $(\wp_P, \overset{P}{\sqsubseteq})\}$. □

It is obvious that given a problem P and two problems $P_1 \in \wp_P^s$ and $P_2 \in \wp_P^s$, the s-distance $d^s(P_1,P_2)$ doesn't necessarily exist.

Conditions for the existence of $d^s(P_1,P_2)$ can easily be stated. Similar remarks hold for d^i and d^P, as well.

Definition 73

Given a problem P, we define the s-depth p^s of a problem $P_1 \in \wp_P^s$, the i-depth p^i of a problem $P_2 \in \wp_P^i$, and the p-depth p^p of a problem $P_3 \in \wp_P$ in the following way:

$$p^s(P_1) = d^s(P,P_1) \qquad (6.42)$$

$$p^i(P_2) = d^i(P,P_2)$$

$$p^p(P_3) = d^p(P,P_3) \quad .$$

□

Definition 74

Given a problem P, we define the s-height g^s of a problem $P_1 \in \wp_P^s$, the i-height g^i of a problem $P_2 \in \wp_P^i$, and the p-height g^p of a problem $P_3 \in \wp_P$ in the following way:

$$g^s(P_1) = d^s(P_1, P_\nu)$$
$$g^i(P_2) = d^i(P_2, P_\epsilon) \qquad (6.43)$$
$$g^p(P_3) = d^p(P_3, P_\nu)$$

□

Theorem 44.

Given a problem P, for every $P_1 \in \wp_P^s$ there exists one and only one $p^s(P_1)$ and $g^s(P_1)$; for every $P_2 \in \wp_P^i$ there exists one and only one $p^i(P_2)$ and $g^i(P_2)$; for every $P_3 \in \wp_P$ there exists one and only one $p^p(P_3)$ and $g^p(P_3)$.

Proof. The proof is obtained directly because of Definitions 70, 71, 72, 73, 74 and because of Theorems 41, 42, and 43. □

The last definitions and properties presented are very interesting because they introduce a natural base for further investigation of the following topics:

a) understanding and measurement of the complexity of a reduction strategy;

b) computation of a heuristic measure function to guide an expansion strategy;

c) comparison of the complexities of different representations of a problem in order to obtain "good" solutions by means of an appropriate search strategy.

In conclusion, the computation effort required for solving any kind of "auxiliary" problem, useful for a more efficient solution of a given problem, has to be considered in order to evaluate the overall computation effort involved in the whole solution process.

7. The Unitary Approach

In the preceding sections, we have presented a formal approach to problem solving which is comprehensive of both the state-space and problem-reduction approaches.

This observation is based on the fact that in section 2 we have presented a formalization of the state-space approach to the notion of problem.

However, because of the algebraic structure which we have developed in sections 2, 3, and 4, we have obtaines as well, in sections 5 and 6, a formal definition of relationships between problems and an algebraic structure of "problem spaces."

In this way, the necessary framework, in which problem-reduction approach can be usefully embedded, has thus been presented as well. The adopted algebraic framework has allowed us to obtain a synthetic, coherent, and unitary description of the matter. However, algebra is an unsuitable representation language and an inappropriate tool for describing the basic activities of an automatic problem solver. Our point of view is that algebra represents a useful framework for a "syntactic" description of problem-solving approaches, whereas logic constitutes an appropriate tool for a "semantic" description which is itself a concrete base for the design of representation languages and representation language interpreters.

In fact, a semantic reformulation of the matter presented in the preceding sections is now being done at the Milan Polytechnic Artificial Intelligence Project.

In this section, we are going to point out the unitary aspect of our theory by presenting the main implications of the above-outlined, formal approach on problem solvers' and representation languages' design criteria. Of course, because of the above-mentioned reasons, this will be done in a partly informal way.

Our basic point of view is that a man can draw from the intuitive problem and from its environment two different sets of effective information: control information and problem information. These two sets will constitute, expressed in an appropriate representation language, the control base (B_C) and the problem base (B_P).

The automatic problem solver acts on these two bases of information as an interpreter and can perform the three basic activities of selection (S), search (R), and learning (L).

Its activities are controlled and organized by a monitor system (M). We now examine in detail the above-outlined concepts, which are graphically illustrated in Figure 6.

The problem base contains all informations on the problem to be solved (P) and its environment, which the man thinks sufficient for the solution of P. In fact, the problem base is built up as a set which contains: P (possibly many different representations), implicants of P, auxiliary problems related to P, solved problems, simple and/or composite costs for P or for the other problems of the problem base. The problem base is first submitted to an ordering process which gives to the information contained in it a hierarchic ordering (e.g., a discrimination net

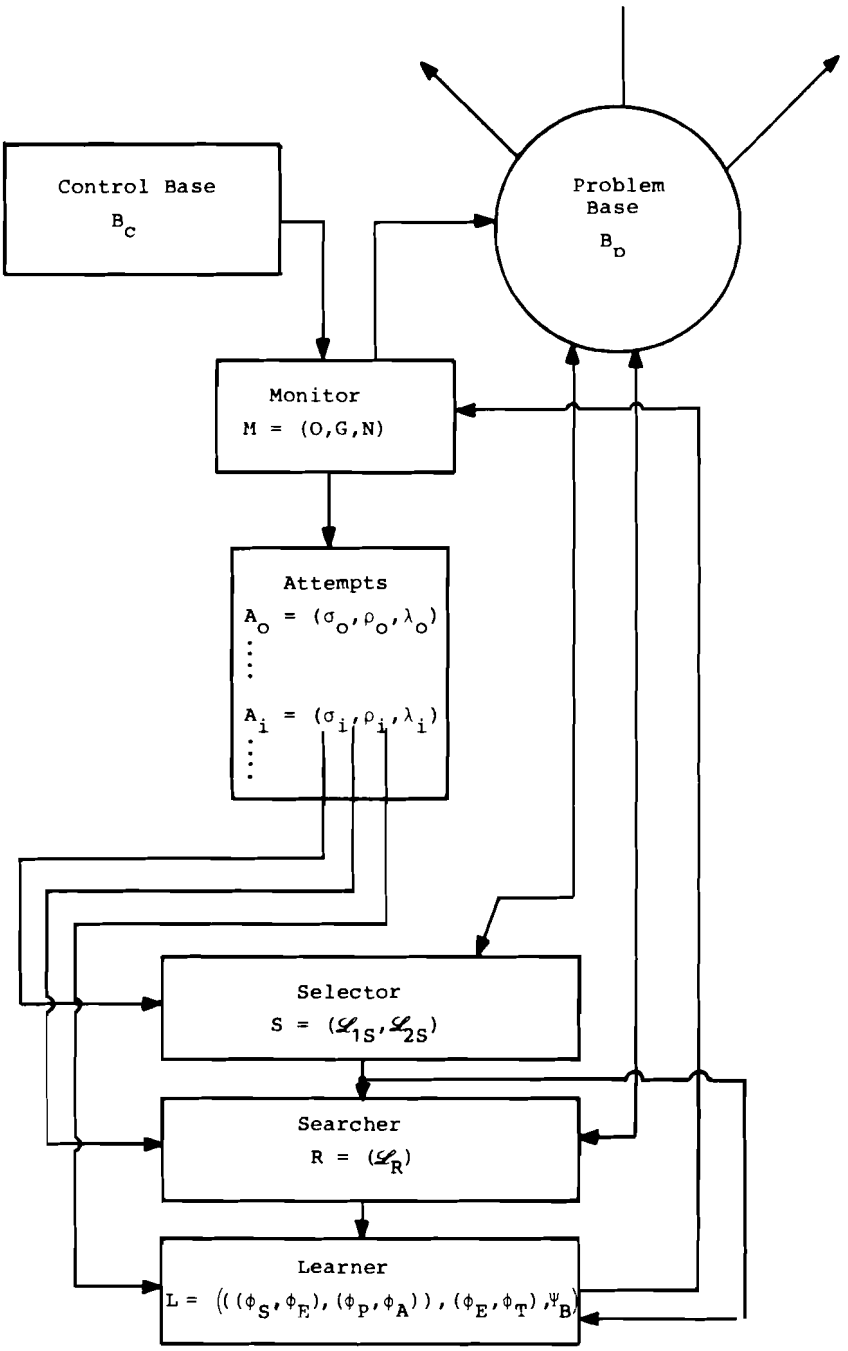


Figure 6. Schema of an automatic problem solver.

or a tree) with the purpose of allowing an easier and more efficient exploration of it whenever it is required by the monitor system. The basic characteristic of the problem base is that it constitutes a dynamic set. In fact, whenever the automatic problem solver achieves some useful results during its search activity, this is inserted in the problem base, at its right place, and can be used afterward as a datum of the problem.

All problems which are contained in the problem base may be either active problems or passive problems. The active problems (P_a) are those problems which can be considered as reduction operators when a reduction strategy is used in order to construct the solution of a problem. They may be implicants, coverings, subproblems, auxiliary problems, etc. The passive problems (P_p) are the solved problems which consitute the "terminal nodes" of a reduction strategy.

We want now to emphasize that the man doesn't exactly know if the informations of the problem base are enough, short, or redundant for the solution of the problem; or which of them are to be used; or in which way the cooperation process among them must be organized in order to solve the problem.

We now shall describe the basic activities of the monitor systems (M). The monitor system can be considered as a triple:

M = (O,G,N), where O,G,N are functions which specify the activities of the monitor M, namely:

O = an ordering function which acts on the problem base B_p and gives to it an appropriate hierarchic ordering;

G = a function which controls the generation of the attempts;

N = a function which manages the generated attempts by means of "interrupt," "activate," and "call garbage collector" signals.

The global activity of the monitor system is fixed by the user of the automatic problem solver by means of the control information.

The ordering activity of M has already been shown; we will only outline that it also controls the insertion in the problem base of the new information arising during the solution of the problem.

The basic activities of the monitor system are the generation (G) and management (N) of attempts. An attempt A_i is triple $A_i = (\sigma_i, \rho_i, \lambda_i)$ where:

σ_i constitutes an input signal for the selection activity;

ρ_i constitutes an input signal for the search activity;

λ_i constitutes an input signal for the learning activity.

The attempts are generated by G in a temporal sequence. The signals σ_i, ρ_i, and λ_i of an attempt A_i are generated as a consequence of the initial control information and of the preceding learning activity. In particular, it is clear that $A_o = (\sigma_o, \rho_o, \lambda_o)$ must be entirely specified in the control base.

The generated attempts, $A_o, A_1, A_2, \ldots$, are organized by N in an appropriate hierarchic structure which must allow an easier management. The attempts must be managed in such a way that they can be considered as coroutines: old attempts can be activated and new attempts can be interrupted whenever it is useful. The attempt management N is done by means of the control information. In particular, the "activate" and "interrupt" signals are generated on the basis of the computational effort done up to a certain point estimated by means of functions contained in the control base. The attempt management N must also provide a garbage collector which destroys the old unuseful attempts whenever it is necessary.

We can now examine in detail the content of the control base (B_C). The control base contains all informations that a man can draw from the intuitive problem and from his knowledge of the operating way of the artificial automatic problem solver in order to initialize the metasystem and to control its dynamic development.

In other words, the problem solver--considered as a metasystem--is an artificial entity, existing outside the user, which has been provided by the artificial intelligence scientist as a general-purpose tool.

However, the ingenuity and creativity of the user can be exploited in order to "tune" such a general-purpose tool in the direction of the semantic domain from which the problem arises.

Hence, such initialization (or specification) of the general-purpose metasystem constitutes a way of defining a special-purpose system. In particular B_C must contain:

a) the three functions O,G,N, which determine the activity of the monitor system M;

b) the first attempt $A_o = (\sigma_o, \rho_o, \lambda_o)$;

c) the sets of functions $\Phi_S, \Phi_\Sigma, \Phi_P, \Phi_A, \Phi_E, \Phi_T$;

d) the functions $\phi_S, \phi_\Sigma, \phi_P, \phi_A, \phi_E, \phi_T$, and ψ_β.

Now, we can precisely define the basic activities of the selector, of the searcher, and of the learner.

The selector, S, can be considered as a couple:

$S = (f_{s1}, f_{s2})$, where f_{s1} and f_{s2} are functions which specify the activities of the selector S, namely:

a) f_{s1} is a function which acts on the problem base and which selects: either a set of states to be expanded, if an expansion attempt is active at that time, or a set of passive problems which must take part in a reduction operation, if a reduction attempt is active at that time; thus, either $f_{s1} \in \Phi_S$ or $f_{s1} \in \Phi_P$.

b) f_{s2} is a function which acts on the problem base and which selects: either a set of inputs for executing an expansion operation, if an expansion attempt is active at that time, or a set of active problems for performing the reduction, if a reduction attempt is active at that time; thus, either $f_{s2} \in \Phi_\Sigma$ or $f_{s2} \in \Phi_A$.

A better understanding of the operation mode of S is obtained by considering in detail its input σ_i:

σ_i is a triple;

$\sigma_i = (\alpha, \beta, \gamma_\sigma)$ where,

a) α is an activation function for the block S;

b) β is a function which specifies if the type of activity of S must be tuned in the direction of expansion (E) or reduction (T);

$\gamma_\sigma = ((f_S, f_\Sigma), (f_p, f_A))$, where:

a) (f_S, f_Σ) constitutes the functions (f_{s1}, f_{s2}) of S if an expansion attempt is active, i.e., $(\beta = E)$;

b) (f_p, f_A) constitutes the functions (f_{s1}, f_{s2}) of S if a reduction attempt is active, i.e., $(\beta = T)$.

Of course, a bidirectional exchange of informations between S and the problem base is provided. The output of S is an input of R. The searcher R can be considered as a "one-tuple": $R = (f_R)$, where f_R is a function which specifies the activity of the searcher R. f_R is a function which acts on that part of the problem base selected by S and performs the expansion or reduction operation; thus, either $f_R \in \Phi_E$ or $f_R \in \Phi_T$. Let us consider the input of R, ρ_i.

ρ_i is a triple;

$\rho_i = (\alpha, \beta, \gamma_\rho)$ where,

a) α and β have already been defined;

b) $\gamma\rho = (f_E, f_T)$ where,

f_E constitutes the function f_R of R, if an expansion attempt is active, i.e., (β = E);

f_T constitutes the function f_R of R, if a reduction attempt is active, i.e., (β = T).

Of course bidirectional exchange of information between R and the problem base is provided. The learner L can be considered as a triple:

$$L = (((\phi_S, \phi_\Sigma), (\phi_P, \phi_A)), (\phi_E, \phi_T), \psi_\beta),$$

where,

a) each ϕ_j is a function which selects a function f_j of a given set $\phi_j = \{f_j\}$, (where the f_j's have been previously defined);

b) ψ_β is a function which selects a value for β from the set $\Theta = \{E, t\}$.

The selected functions f_j and the selected value for β will be used by the monitor in order to set up the next attempt. The functions ϕ_j and the function ψ_β are provided by the control base and act on Φ_j, and Θ by taking into account the informations

obtained from S and R which constitute the inputs of L, again see Figure 6. The above exposed concepts, although not completely formalized, give a clear and precise insight into the basic structure of an automatic problem solver. It is also evident that the design of an automatic problem solver is the design of a representation language interpreter, as it has been shown in section 1. We want to point out, as well, that none of the now existing goal-oriented languages has all the properties illustrated in the above schema (in particular, no interpreter yet exists with an explicitely designed learning part).

It is reasonable to assert that the overall theoretical structure, which has been briefly outlined in this section, is, indeed, a source of many new research directions intended to provide a more comprehensive and detailed description and understanding of the whole subject. On the other hand, we are aware of having here presented a unitary and broadly comprehensive framework which we deem as capable of embracing the important notions of problem solving and representation languages.

In conclusion, we believe that the time has arrived in which it is necessary to orient future research trends and to base the design of future representation languages and interpreters on theoretical grounds which are very much needed.

In other words, we feel that the difficulties with languages for representing problems, at present, are similar to the difficulties which existed in the sixties--in the languages used for describing algorithms.

The advent of the theory of formal languages and of syntax-directed translators can now be followed in an analog style by the development of the new theory of problems and of theoretically structured problem solvers and interpreters.

In conclusion, this section represents, in our view, a valid contribution toward these research goals.

8. Conclusions

In this section, we shall give a brief insight into the semantic description of our unitary approach which is now being investigated and which is not yet completely developed. Other conclusive remarks and promising research directions are presented at the end of this section as well. The semantic description is intended to be a more appropriate presentation of the unitary approach for achieving the following goals:

a) a formal definition of the basic activities of automatic problem solvers;

b) construction of a useful base for the detailed design of representation languages interpreters;

c) outline of specific design criteria for representation languages.

Following, we shall briefly sketch the most important definitions which constitute an approach, called semantic, which is alternative to the approach, called syntactic, which has been presented in section 2.

The reason for presenting these results here is related to the need for giving a more careful and precise description of the activity of selection, research, and learning, which belong to the kernel of the metasystem (automatic problem solving).

Definition 75

The attribute set is a finite set A of elements A_i called attributes, i.e.:

$$A = \left\{A_1, A_2, \ldots, A_n\right\} \quad . \tag{8.1}$$

Definition 76

The value set bound to an attribute A_i is a finite set V_i of elements v_j^i called the values for the attribute A_i, i.e.:

$$V_i = \left\{v_1^i, v_2^i, \ldots, v_{m_i}^i\right\} \quad . \tag{8.2}$$

Definition 77

An attribute-value couple (AVC) is a couple $c_i = (A_i, v_j^i)$ where $A_i \in A$ and $v_j^i \in V_i$.

Definition 78

The attribute-value couple set (AVC set) for an attribute $A_i \in A$ is the set C_i defined in the following way:

$$C_i = \left\{c_i \,\middle|\, (c_i = (A_i, v_j^i))\,(v_j^i \in V_i)\right\} \quad . \tag{8.3}$$

Definition 79

An S-state is an n-tuple $\bar{s}$ of AVC's such that:

$$\bar{s} = (c_1, c_2, \ldots, c_n) \quad , \tag{8.4}$$

where,

$$c_1 \in C_1, \ldots, c_n \in C_n \quad . \tag{8.5}$$

Definition 80

The S-state-space is the set $\bar{S}$ of S-states such that:

$$\bar{S} = C_1 \times C_2 \times \ldots \times C_n \quad . \tag{8.6}$$

Definition 81

A legal condition (LC) on an S-state-space $\bar{S}$ is a binary relation L_i on $\bar{S}$, i.e.:

$$L_i \subseteq \bar{S} \times \bar{S} \quad ,$$

and,

$$L_i = \left\{ (s', s") \,\middle|\, (s' \in \bar{S}) \wedge (s" \in \bar{S}) \wedge P_i(s', s") \right\} \quad , \tag{8.7}$$

where P_i is a predicate, i.e.:

$$P_i(s', s") : \bar{S} \times \bar{S} \rightarrow \{T, F\} \quad . \tag{8.8}$$

(T and F stand for true and false).

Please note that the nature of "property" $P_i(s', s")$, is directly related to the core of the semantic aspect of the problem. Therefore, it is natural to utilize, for the formal description of this property, any appropriate semantically oriented language as, for instance, the first-order predicate logic.

In this way, an interesting connection is being presented here which shows an interaction between an algebraic framework and a logic one.

This is a further example of the unitary effort on representation languages which constitutes the main spirit of our research activity.

Definition 82

A legal condition set on an S-state-space $\bar{S}$ (LC set) is a finite set L defined in the following way:

$$\left\{L = L_o, L_1, \ldots, L_t\right\} \quad , \tag{8.9}$$

where,

$L_o = \bar{S} \times \bar{S}$;

L_i, for i = 1,...,t, is an LC on $\bar{S}$.

Definition 83

The constraint N related to an LC set L on an S-state-space $\bar{S}$ is the binary relation on $\bar{S}$ defined in the following way:

$$N = \bigcap_{i=o}^{t} L_i \quad . \tag{8.10}$$

Definition 84

An S-problem schema $\bar{M}$ is a couple $\bar{M} = (\bar{S},N)$ where:

$\bar{S}$ is an S-state-space;

N is a constraint on $\bar{S}$.

Definition 85

An S-problem $\bar{P}$ is a quadruple $\bar{P} = (\bar{S},N,\bar{i},\bar{f})$ where:

$(\bar{S},N)$ is an S-problem schema;

$\bar{i} \in \bar{S}$ is an S-state called the initial S-state;

$\bar{f} \in \bar{S}$ is an S-state called the final S-state.

The few definitions presented above are sufficient to outline the main characteristics of the mentioned semantic presentation of the unitary approach with respect only to the state-space model.

We conclude this insight into this promising research matter by showing how the basic activity of selection can be adequately defined by means of this new formal framework in the particular case of the state-space approach presented.

Definition 86

A global representation of a problem P is a quintuple $\bar{P}_G = (A,V,L,\bar{i},\bar{f})$ such that:

$A = \{A_1,A_2,\ldots,A_n\}$ is an attribute set:

$V = \{V_1,V_2,\ldots,V_n\}$ is a set of value sets for the attributes of A;

L is a legal condition set defined on the S-state-space $\bar{S}$ obtainable from A and V;

$\bar{i}$ is the initial S-state;

$\bar{f}$ is the final S-state.

The selector is intended as a system acting on a global representation of a problem $\bar{P}_G$ and yielding a new representation called selected representation $\bar{P}_T$ which contains only the "elements" selected for building up the problem to be expanded (we recall that this description is related to the space-state approach only).

Definition 87

A selected representation of a global representation of a problem $\bar{P}$, $\bar{P}_G = (A,V,L,\bar{i},\bar{f})$, is a quintuple $P_T = (A_T,V_T,L_T, \bar{i}_T, \bar{f}_T)$ such that:

$$A_T = A \ ;$$
$$V_T \subseteq V \ ;$$
$$L_T \subseteq L \ ;$$
$$\bar{i}_T = \bar{i} \ ;$$
$$\bar{f}_T = \bar{f} \ .$$

Definition 88

A selector is a couple $T = (\rho,\tau)$, where ρ,τ are two functions such that:

$$\rho : V \to V$$

$$\tau : L \to L \quad .$$

Theorem 45.

A selector $T = (\rho,\tau)$ is a function such that:

$$T : \left\{\bar{P}_G\right\} \to \left\{\bar{P}_T\right\} \quad ,$$

where $\left\{\bar{P}_G\right\}$ is the set of all global representations of problems and $\left\{\bar{P}_T\right\}$ is the set of all selected representations of problems.

Proof. The proof is directly obtained because of Definitions 86, 87, and 88.

These concepts are enough to point out that the most important and promising research direction which arose during the investigation of the matter presented in this paper is the development of a complete and adequate semantic presentation of our unitary approach. Other research directions have been presented in the preceding sections. We recall the most promising ones:

a) understanding and measuring of complexity on the basis of the rich algebraic structure of the resolvent set H and of the subsets of $\not{p}$: $\not{p}^s_p$, $\not{p}^i_p$, and $\not{p}_p$;

b) invention of "intersection" and "welding" methods for bidirectional algorithms based on the "levels" defined in the resolvent set H;

c) confrontation of different representations of a problem by means of the concept of morphism;

d) design of more detailed problem solvers and representation languages' interpreters

e) deeper understanding of learning in problem solving.

We believe that we can conclude this paper by asserting that the presented matter, although not completely formalized and also not always deeply detailed on some important topics, can be considered as a set of "first cut" results. Precise investigation directions have been found out, some topics have been investigated in more detail, and a few search methods have been recognized as now promising ones.

Acknowledgment

The authors are indebted to the cooperation of the researchers of the Milan Polytechnic Artificial Intelligence Project (MP-AI Project) and, in particular, to Dr. A. Sangiovanni Vincentelli.

9. References

[1] Banerji, R.B. Theory of Problem Soving, an Approach to Artificial Intelligence. New York: Elsevier, 1969.

[2] Berge, C. Theorie des Graphes et Ses Applications. Paris: Dunod, 1958.

[3] Bobrow, D.G., and Raphael, B. "New Programming Languages for AI Research." Presented at the Third International Joint Conference on Artificial Intelligence, Stanford University, Stanford, California, 1973.

[4] Coray, G. "Additive Features in Positional Games." ORME-IP-IRIA-NATO, Nato Advanced Study Institute on Computer Oriented Learning Processes, Procèdures Informatique d'Apprentissage, 1974.

[5] Ginzburg, A. Algebraic Theory of Automata. New York: Academic Press, 1968.

[6] Hartmanis, J., and Stearns, R.E. Algebraic Structure Theory of Sequential Machines. Englewood Cliffs, New Jersey: Prentice-Hall, 1966.

[7] MacLane, S., and Birkhoff, G. Algebra. New York: MacMillan, 1967.

[8] Nilsson, N.J. Problem-Solving Methods in Artificial Intelligence. New York: McGraw Hill, 1971.

[9] Pohl, I. "Bi-Directional and Heuristic Search in Path Problems," SLAC Report 104. Stanford Linear Accelerator Center. Stanford University. Stanford, California, 1969.

[10] Vincentelli, A., Sangiovanni, and Somalvice, M. "Formulazione Toerica des Metodo dello Spazio degli Stati per la Risoluzione Automatica dei Problemi." Relazione Interna 72-74, MEMO MP-AIM-6, Politecnico di Milano, Istituto di Elettrotecnica ed Elettronica, Laboratorio di Calcolatori, Milan, 1972.

[11] Vincentelli, A., Sangiovanni, and Somalvico, M. "State-Space Approach in Problem-Solving Optimization." Relaxione Interna 73-15, MEMO MP-AIM-12. Politecnico di Milano, Istituto di Elettrotecnica ed Elettronica, Laboratorio di Calcolatori, 1973a.

[12] Vincentelli, A., Sangiovanni, and Somalvico, M. "Theoretical Aspects of State-Space Approach to Problem-Solving." Relazione Interna 73-16, MEMO MP-AIM-16. Politecnico di Milano, Istituto di Elettrotecnica ed Elettronica, Laboratorio di Calcolatori, 1973b.

Logic and Interpreters

Enrico Pagello

We want to suggest the idea of relating the process of interpreting the input statement of a program to a condition over a complemented distributive lattice, which is the model of a propositional logic language--choosen as programing language--so that an algebraic semantic definition of the programing activity and interpretation process may be given.

Our point of view is to follow the idea [1] of using a theorem proving system as an interpreter of a mathematical logic language, so that we can consider propositional logic languages, or predicate logic languages as programing languages with an interpreter. We have choosen the simplest branch of algebraic approach to mathematical logic [4], the Boolean algebras, for developing a definition of a simple programing language, based on propositional calculus, through an algebraic semantic domain, following the classical results of relations between lattice theory and propositional logic [3].

To build semantical models of programing language, we can consider the relations between programs, denoted by topological relations in the function space in which the set of programs [5] is mapped, and define the computations as the sequence of states generated by the interpreter.

Therefore, we shall specify a syntactic domain for our programing language, i.e., we shall develop a general theory, and we shall give a particular axiomatic system, i.e., the semantic model, based on lattice theory. Also, we shall define the interpreter model on this semantic domain.

Therefore, if we consider the problem of interpreting a valid sentence of programing language as the problem of accepting the input sentence by generating a proof procedure of validity of the theorem--where the statements of programs are the sequences of input theorems, the computations are the derivations of the proof procedure, and the interpreter is the theorem prover--then a semantic model of this process will constitute an interpretation of interpreter acting.

We shall consider the theory of axiomatic systems as a model for our logic formula taken as programing statements because they have the property for giving a complete characterization of all those relations explicitly definable in them [2].

References

[1] Kowalski, R. "Predicate Logic as Programming Language." Proceedings of I.F.I.P. 1974, Stockholm. In press.

[2] Kreisel, G., and Krivine, J.L. Elements of Mathematical Logic, Model Theory." Amsterdam: North Holland, 1971.

[3] Ruthenford, D.E. Introduction to Lattice Theory. Edinburgh: Oliver & Boyd Ltd., 1965.

[4] Stone, M.H. "The Theory of Representation for Boolean Algebras." Trans. Amer. Math. Soc. 40, 1936.

[5] Wegner, P. "A Framework for Semantic Modelling." Tech. Report No. 73-65, Center for Comp. & Inf. Sci., Brown University, Providence, Rhode Island, 1973.

Artificial Learning Systems and QAS

A.M. Andrew

1. Learning to Answer Questions

A person performing a question-answering task will certainly learn from experience as he operates. Not only will he have a strong incentive to find out more about the subject matter on which he is liable to be questioned, but also he will learn, for example, how much detail his customers expect to receive in their answers, and under what circumstances they will welcome a request for clarification of the question before an answer is given.

Computer based question-answering systems are usually able to add to their store of information pertaining to the subject matter with which they deal. In other words, they can extend their data bases. Some of the very recent developments in artificial intelligence stimulated by the development at MIT of the programing language PLANNER have provided very powerful ways of organising and modifying data bases.

In an automatic question-answering system with a very large data base, such as is visualised as an IIASA project, there would be advantages in letting the organisation of the data base be influenced by the questions asked and by feedback indicating user satisfaction with the answers given. One fairly simple way in which the organisation could be improved as operational experience was gained would be by arranging that parts of the data base which are frequently wanted are so placed in store that they can be rapidly accessed. Apart from this, a great many heuristic rules could be developed which would speed up the process of locating data base entries relevant to a particular enquiry; for example, where the required entry is the intersection of a number of subject areas indicated by descriptors, the complexity of the search could be strongly influenced by the order in which the descriptors were applied to narrow the field, and heuristic rules for such ordering would depend on operational experience.

Operational experience could also be used to alter the data base quite fundamentally. As a subject area develops, the relationships among its component parts may change. For best results, the structure of the data base should change accordingly, even though most of the information it contains may have been entered when the relationships between subject divisions were in their out-of-date form. An example of such a change in relationships (and a data base insufficiently flexible to adjust to it) can be seen in the classification of British patents, where computing devices of all kinds are classified with mechanisms and linkages

rather than under the heading of electronics. This change in the appropriate relationship has arisen over a period of some years and is a particularly obvious one. However, there are subtle and comparatively short-term changes in the associations of ideas in any given field, and these could profitably be reflected in the structure of the data base. Up-to-date information to determine the changes could only come from analysis of user behaviour.

It is, of course, entirely possible that the system would learn to "know" its different users, and would adapt its operation to suit their idiosyncracies. It might then be interesting and salutary if users could interrogate the system about its view of their behaviour as users.

A feedback of a measure of user satisfaction might be obtained simply by asking users to provide a satisfaction score before "logging out" of the system. It is possible it might be difficult to persuade users to provide the feedback consistently, and probably the times when they would be least cooperative would be when their responses would have been most valuable, namely when they are caught up in the excitement of some new development in their field of study.

Some indication of user satisfaction might be obtained otherwise than by asking the user to provide feedback. A user who presents a new question fairly soon after receiving an answer is probably satisfied with the first answer, one who goes away for a long time may be either satisfied or frustrated, and one who presents essentially the same question in a modified form was pretty certainly dissatisfied with the first response. Possibly an initial stage of learning would be to learn how to judge customer satisfaction.

2. Types of Feedback

Where the feedback is in fact provided by cooperative users of the system, it may be possible to obtain from them some indication of what would have constituted a more satisfactory response. Any such indication can greatly facilitate a process of automatic adjustment by indicating how the response can usefully be modified. Without such an indication, it is generally necessary for the system to superimpose experimental variations on its method of operation.

A paper by Andrew [4] has some relevance here. It compares learning systems with and without explicit internal models of the environment, and shows that the two may be mathematically equivalent. It is assumed that the feedback available to produce the adaptation is of the "measure of satisfaction" type with no associated indication of the direction in which the response could profitably be changed. In a learning system embodying an explicit model, the task of adjusting the model to correspond to the real environment is one for which the "direction-of-change" information is available, in contrast to the alternative approach of optimizing the system response without the intermediary of a model.

Gabor, Wilby, and Woodcock [12] adopted an operating principle for their "learning filter" which made no use of "direction-of-change" information. Presumably, their intention was to make a highly versatile device equally suitable for applications in which the "direction-of-change" information is present and those in which it is absent. However, the main applications they described were tasks of modelling or prediction, and for these "direction-of-change" information is available. Because of this, Lubbock (1961) was able to show that an alternative form of operation would converge much more rapidly than the Gabor filter. Lubbock's method is similar to the process of "regression analysis" in statistics, and to "stochastic approximation" in control theory.

3. Credit-Assignment

As Minsky [16] points out, the operation of a complex system to perform some task involved an enormous number of decisions. When feedback of a measure of satisfaction or goal-achievement in the task becomes available, it is not at all obvious how the credit (or blame) should be assigned to the constituent decisions. If the feedback is to produce "learning," or automatic improvement of performance of the task, it is necessary that credit assignment be made in some way, so that favourable decisions can be reinforced and unfavourable ones modified for future operations.

Since the ability to learn from experience is highly developed in people and animals, it is natural to look to the nervous system for clues about how to organise a learning system. Unfortunately, in our present state of understanding, looking at real nervous systems is not of much help. Many workers have built or simulated networks having properties of self-organization, or at least self-adjustment, and it is interesting and instructive to consider how the credit-assignment problem is overcome or evaded in these schemes.

The most widely-publicised type of self-adjusting network is the perceptron due to Rosenblatt [21] and very clearly described by Nilsson [18]. Minsky and Papert [17] have shown that there are severe intrinsic limits on what can be done by a Simple Perceptron. For the Simple Perceptron, the credit-assignment difficulty is evaded by arranging that the adjustments made in the course of learning are restricted to points which influence the output of the net in a very direct way. Thus there is no difficulty in determining the sensitivity of the output to a proposed change at any of the adjustable points. The learning filter of Gabor, Wilby, and Woodcock [12] and similar schemes proposed independently by Andrew [2] share this single-layer character.

There are, however, some systems described in the literature which allow for adaptation throughout a complex net in which some adjustable elements exert their effect on the output by acting through other adjustable elements. It is because of the difficulty of credit assignment, as discussed by Minsky, that work on

self-organising networks (sometimes termed the "Cybernetic Approach to Artificial Intelligence") has not progressed further than it has. Nevertheless, some methods can be referred to which go some way toward solving the problem in that they do allow automatic adjustment not restricted to a single functional layer. Some of these methods will be briefly reviewed, with some ideas for further study.

The methods are described, for the most part, in connection with networks of threshold elements of the type usually termed McCulloch-Pitts neurons. Their underlying principles can, however, be adapted to networks of other kinds and to the embodiment of a learning capability in a question-answering system. For various reasons, there is a tendency among workers in this area to think in terms of networks of model neurons. This is partly in the hope of producing systems having some direct correspondence to nervous-system functioning, even though McCulloch and Pitts [14] made it clear that their seductively simple model neurons are not purported to have properties corresponding closely to those of real neurons. Apart from possible correspondence to real neurons, however, these model neurons are attractive in their own right as network elements. In spite of their simplicity, they are, in fact, universal computing elements, since it was shown by McCulloch and Pitts that networks of them can compute anything which is computable. More important still, a given network can produce a very wide range of forms of behaviour, the transition from one form to another being produced by an accumulation of small changes in thresholds and synaptic weights.

Some approaches to learning networks not restricted to single-layer adaptation will now be reviewed.

4. Continuous-Discontinuous Adjustment

The "Pandemonium" scheme proposed by Selfridge [23] typifies one way in which adaptive changes can be introduced throughout a complex net. In this scheme, there is a primary adjustment process which operates continuously and is, in fact, a "perceptron training algorithm." This process is subject to the limitations of the Simple Perceptron, but it operates in conjunction with another (discontinuous) adjustment process as follows.

The elements of the net in which the primary adjustment takes place are termed by Selfridge "cognitive demons," the word demon being used in the same sense as in referring to a Maxwell demon, i.e., to indicate a small creature. The cognitive demons receive their input signals from "computational demons," and it is a simple matter to compute, from the parameter settings arrived at by the primary adjustment process, measures of the "worth" of the respective computational demons to the final decision. Demons of low "worth," i.e., those whose outputs play little part in the decisions taken at later stages, may be eliminated and replaced by other, different demons. This rearrangement of the computational demons according to the indications of "worth" constitutes the secondary, discontinuous form of adjustment.

There are difficulties in finding new types of "computational demon" likely to have high "worth" to replace those which are eliminated. Where the nature of the demons is simple, new ones may be formed randomly; this is what is done in the extension of the Simple Perceptron due to Roberts [19] in which the "computational demons" are perceptron association units receiving inputs from a randomly chosen set of sensory units. When an association unit proves to have low "worth," the connections it receives from sensory units are dissolved, and a new set is chosen at random.

For more sophisticated types of "computational demon," totally random generation is useless as it would have a negligible chance of producing new demons with sufficiently high "worth" to survive. Selfridge suggests two ways of generating new demons likely to have high "worth"; these are by processes which he terms "conjugation" and "mutated fission."

The idea of "conjugation" is simple that the outputs of two existing high-worth demons are combined (in any one of a number of ways), and the combining element constitutes a new "computational demon." An interesting aspect of this idea (not commented on by Selfridge) is that the measure of "worth" computed for the combining element must somehow pass on through it to contribute to the "worth" measures of the two elements whose outputs are being combined. Thus there is a need for a simple form of what will be discussed later on as "significance feedback."

The main thing to be said about "mutated fission" is that nobody really knows how to achieve it. It is simple to arrange if the changes constituting the "mutations" are of a simple preconceived form, e.g., changes in parameter-values, but then no qualitatively new demons can ever evolve. It is necessary that the demons be represented in a way which allows a suitable form of "heuristic connection" between the possible forms demons may take [16].

Forms of self-improvement having the continuous-discontinuous character typified by the Pandemonium could very readily be incorporated in a question-answering system. Points in the system at which decisions are made can be made to compute measures of "worth" for the subsystems from which they receive information. If the totality of the measures of worth for a particular subsystem proves to be small, the subsystem might be automatically modified or annihilated.

However, it seems reasonable to suppose that an adjustment mechanism operating in a uniform way throughout the system would be more effective than the continuous-discontinuous form of operation, and there have been various attempts to devise such a mechanism. Some of these will now be discussed.

5. Reduction of Redundancy

Barlow [9] and Uttley [26] have discussed ways in which adaptive changes in a network might be determined independently of any feedback from the environment indicating the effect of the net's output. Their suggestion is that the changes should operate to reduce the redundancy of signals passing through the net. Andrew [5,6] has argued that such locally determined adaptation can be of only limited value and must operate in conjunction with a process of true feedback.

Input data to the nervous system is often highly redundant, and the suggestion that redundancy is reduced in the early stages of processing agrees well with experimental data on the nervous system. The familiar observation that the response of many parts of the nervous system to abrupt changes in signal level is much greater than to sustained inputs is an example of redundancy reduction. The change may be abrupt in either time or space. A signal which does not indicate an abrupt change could have been inferred approximately by extrapolation of other signals, so is to some extent redundant and is therefore attenuated.

Reduction of redundancy of the requests for information must be an important part of the operation of a question-answering system, and consequently a self-modifying system might automatically improve its own performances in this respect. As in the case of nervous-system inputs, some simple forms of redundancy could be eliminated without reference to the overall operation of the system; for example, if there are pairs of words or phrases which only appear together in the inputs, one member of each pair is redundant. The recognition of other forms of redundancy must depend on overall feedback to indicate what is useful and what is redundant in the inputs. Nevertheless, the discussions by Barlow and Uttley are probably well worth keeping in mind in the development of a self-modifying question-answering system. Not only does redundancy-reduction simplify processing requirements, it may also facilitate the discovery of relationships between signals which were previously obscured.

6. Methods of Widrow and Stafford

Widrow [27] has discussed ways of achieving adaptation in neural nets not restricted to a single functional layer. In his paper, he begins by discussing the adjustment of a single-threshold element in a fashion very similar to that used in a Simple Perceptron. (There is one slight difference in that he favours a procedure in which an adjustment is made whether or not the response of the network was "correct." The usual method for perceptrons requires an adjustment only for "incorrect" responses).

Widrow then describes a method of adjustment for a net in which a number of adjustable elements produce outputs which impinge on one further threshold element whose output is the output of the net. The basis of the method is the rule that the net must

be adjusted to give the "correct" response to each input pattern, and that the adjustment needed to let this happen should be achieved with minimal disturbance of previous adjustments. Widrow requires that the number of adjustable elements which is altered at all should be the minimum which will produce the required effect, and where there is more than one equally large subset of elements which could be chosen, that which is used is the one requiring the smallest total parameter changes.

Widrow also discusses the extension of these ideas to networks giving multiple outputs. The aim is always to achieve the correction of the current response with small alteration of the net, since such alteration represents a disturbance of previous learning.

It is, of course, possible to adopt any of a number of distinct measures of the amount of disturbance represented by a given set of changes in the network. The measure could be the total amount of parameter change (where parameter is used to mean a synaptic weight or a threshold level), or the maximum value of parameter change, or the sum of squares of parameter changes. Widrow uses a two-stage criterion, with a number of elements affected as the primary consideration, and total magnitude of the changes as the second. The second consideration is only invoked if the first does not indicate a clear choice.

An alternative method due to Stafford [24,25] is also based on the general principle that the response to the current input should be corrected with as little disturbance as possible of the existing settings in the network. Stafford's methods (he describes two variations) seem rather more suitable than that of Widrow for incorporation in a neural net, since their operation depends on activity distributed over the net. Widrow's method, on the other hand, is difficult to implement without an "adaptation centre" computing the changes. The distributed form of operation seems more plausible as a possible model of nervous-system functioning, but either type could be useful in suggesting ways in which learning might be made to occur in a question-answering system.

Methods which rely on distributed activity are likely to be such that computational complexity increases approximately linearly with network size. For methods depending on an "adaptation centre," the complexity might increase much more steeply.

Andrew [7] has carried out simulation experiments in which randomly formed networks of neuronlike elements were allowed to modify themselves according to many variations of Stafford's two methods. The results were, in fact, disappointing since none of the variations solved the problem which was set.

7. Significance Feedback

Andrew [3,5-7] has used the term "significance feedback" to indicate an adaptation principle which would operate in a distributed fashion and avoids the need for "dummy runs" of the network as required in Stafford's methods.

The essential idea of one form of significance feedback was introduced when Selfridge's Pandemonium was discussed. In this, the feedback was simply of a measure of "worth" or importance attached to the signals in a pathway.

Another form of "significance feedback" indicates the current sensitivity of the output of the net to activity in the channel with which the feedback is associated. This can be achieved by letting every primary pathway have a feedback pathway associated with it, carrying a signal indicating the sensitivity of the output of the net to activity in the primary pathway. Every element in the net must perform a dual role; it must process the primary signals appropriately and must also process the sensitivity signals. For example, suppose an element has the primary function of multiplying two signals $\underline{x}(\underline{t})$ and $\underline{y}(\underline{t})$ to produce an output $\underline{z}(\underline{t}) = \underline{x}(\underline{t}) \cdot \underline{y}(\underline{t})$. Suppose also that a feedback signal $\underline{s}(\underline{t})$ is associated with the pathway conveying $\underline{z}(\underline{t})$ this signal being a measure of the sensitivity of the output to $\underline{z}(\underline{t})$. Then, at least for small variations in the primary signals, appropriate measures of sensitivity for the pathways conveying $\underline{x}(\underline{t})$ and $\underline{y}(\underline{t})$ are $\underline{s}(\underline{t}) \cdot \underline{y}(\underline{t})$ and $\underline{s}(\underline{t}) \cdot \underline{x}(\underline{t})$, respectively. If the multiplication element produces these feedback signals associated with its input pathways, and all other elements in the net process feedback signals in ways appropriate to their primary functions, there is a continuous automatic "sensitivity analysis" throughout the net. The sensitivity measures can be used to determine adaptive changes at points throughout the net.

For small networks of linear and quasi-linear (e.g., multiplicative) elements, the method has been found to work very well. However, the information-processing capabilities of such networks are not very interesting. Interesting behaviour comes from networks embodying strongly nonlinear elements such as threshold elements. For these, it is difficult to see how to implement "significance feedback" since the appropriate sensitivity measures depend on signal amplitudes. A variety of plausible variations of the "significance feedback" principle have been devised, suitable for application to networks of threshold elements. These have been tried out in simple networks which were required to adapt to perform a logical computation which could not be achieved as a linearly separable function of the inputs. Adaptation was by variation of the threshold levels and "synaptic strengths" in the elements. The attempt was made to "train" the networks by the presentation of a succession of random inputs together with a "correct answer" for each.

The conclusion from these experiments was that networks embodying these plausible approximations to the "significance feedback" principle had adaptive properties in that they would adjust themselves to perform the task after starting from various initial states. From other starting states, the adaptation was not successful; there are "trapping states" from which further progress is impossible.

8. Interelement Negotiations

It appears that for effective adaptation, it is necessary to have something else besides the feedback indicating the sensitivity measures. It is also necessary to have some sort of dialogue, or process of negotiation, between the elements of the nets to decide how to apportion the necessary changes among them.

The idea of negotiation implies some sort of conflict, or competition for something which might be called currency, among the elements. In order that the net should adjust itself to give the "correct answer" for all possible input states, its adaptation to each input should produce the least possible disturbance of the synaptic weights and thresholds previously existing. The amount of such change constitutes a currency which can be bargained over and which can form the basis of a "training algorithm" involving a dialogue or process of negotiation.

Andrew [8] has made some suggestions about the form this interelement dialogue might take. Simulation studies are being undertaken.

It can be seen that there is a need to sort out some fundamental questions in connection with learning networks. The general principle that adaptive changes should cause minimal disturbance of previous learning seems essentially sound, but it is not clear what measure of amount of change should be minimized, nor whether it should be differently weighted in different parts of the net according, say, to their proximity to input and output pathways.

It is also unclear whether it is optimal to let the parameters of the net (threshold levels and synaptic weights) represent the only between-trials storage of information in the net. In the usual perceptron training algorithm, as well as in the schemes of Widrow and Stafford, these parameters do represent the only between-trials storage. On the other hand, some learning schemes require other information storage so that correlations and other statistical measures can be computed, the net parameters being altered only if these measures exceed some significant level. The famous learning algorithm used by Samuel [22] to adjust the scoring polynomial used by his checker-playing program depends on the computation of correlation measures separately from the parameters adjusted.

It can be seen that there are some quite general questions relating to the automatic adjustment of neural networks to which we do not know the answers.

9. Other Approaches

A number of topics relating to learning or self-organizing systems have not been touched on. Hierarchical schemes have not been mentioned as such, though some of the discussion of multilayer networks could, perhaps, be rephrased in terms of one of the kinds of hierarchical structure treated by Mesarovic, Macko, and Takahara [15]. Also, there has been no mention of reverberating nets, as postulated by Hebb [13] for his "cell assemblies" and studied in the early simulation studies of Beurle [10] as well as Rochester et al [20]. For the learning of tasks not involving any sort of pattern in time, there is no obvious advantage of reverberating nets over static ones.

Farley and Clark [11] introduce a controlled amount of "noise" or random variation into their system.

The highly individual approach of Aleksander [1] has also been ignored. He has experimented with both static and reverberating networks, particularly in tasks of pattern recognition. He prefers to consider networks not consisting of model neurons but of what he terms SLAM units (standing for Stored-Logic Adaptive Microcircuit). Although this is a different type of basic element, most of what has been said about learning in neural nets remains applicable.

10. Discussion

It is, in fact, because the principles are applicable to networks of elements other than model neurons that they are believed to have relevance to question-answering systems. These are multilayer systems in the sense that many parts exert their effect on the output through other subsystems. If the different parts are all capable of self-modification as the system operates, the problems of multilayer adaptation will arise.

It would be useful to set up, as part of a program of work on question-answering systems, an investigation into self-organizing neural networks. Questions which would be studied initially would include the following:

a) What measure of network change should be minimized to preserve the effects of previous adaptation?

b) What form of "negotiation" between network elements produces changes which do, in fact, produce adaptation and are consistent with the requirements of (a).

c) Is there any advantage in letting statistical measures be computed separately from the network parameters to be adjusted?

Many other questions would arise as the projected progressed and the learning techniques were applied in the question-answering task.

This work would be particularly relevant to the wider aims of IIASA since all large systems achieve viability by adaptation and can only be understood in terms of their adaptive properties. Networks of model neurons provide a relatively simple environment in which to look for general principles of adaptation in complex systems.

References

[1] Aleksander, I. "Some Psychological Properties of Digital Learning Nets." Int. J. Man-Machine Studies, 2, 189, 1970.

[2] Andrew, A.M. "Learning Machines." In Mechanisation of Thought Processes. London: HMSO, 1959.

[3] Andrew, A.M. "Significance Feedback in Neural Nets." Report of Biological Computer Laboratory, University of Illinois, 1965.

[4] Andrew, A.M. "To Model or Not to Model." Kybernetik, 3, 272, 1967.

[5] Andrew, A.M. "The Ontogenesis of Purposive Activity." In A. Locker, ed., Biogenesis, Evolution, Homeostasis. New York: Springer, 1973.

[6] Andrew, A.M. "Significance Feedback and Redundancy Reduction in Self-Organizing Networks." In F. Pichler and R. Trappl, eds., Advances in Cybernetics and Systems Research, Vol. 1. London: Transcripta Books, 1973.

[7] Andrew, A.M. "Studies of Real and Hypothetical Nervous Systems and Their Implications for Social Systems." Conf. on Cybernetic Modelling of Adaptive Organizations, Porto, Portugal, 1973.

[8] Andrew, A.M. "Cybernetics and Artificial Intelligence." Third Int. Congress of Cybernetics and Systems, Bucharest, 1975.

[9] Barlow, H.B. "Possible Principles Underlying the Transformations of Sensory Messages." In W. Rosenblith, ed., Sensory Communication. Cambridge, Massachusetts: MIT Press and Wiley, 1961.

[10] Beurle, R.L. "Functional Organization in Random Networks." In H. von Foerster and G. Zopf, eds., Principles of Self-Organization. Oxford: Pergamon Press, 1962.

[11] Farley, B.G. and Clark, W.A. "Simulation of Self-Organizing Systems by Digital Computer." Trans. IRE, PGIT-4, 76, 1954.

[12] Gabor, D., Wilby, W.P.L., and Woodcock, R. "A Universal Non-Linear Filter, Predictor and Simulator Which Optimizes Itself by a Learning Process." Proc. I.E.E., PT B, 13, 422, 1961.

[13] Hebb, D.O. The Organization of Behaviour. New York: Science Editions 1961. (Originally Wiley, 1949.)

[14] McCulloch, W.S. and Pitts, W. "A Logical Calculus of the Ideas Immanent in Nervous Activity." Bull. Math. Biophys., 5, 115, 1943.

[15] Mesarovic, M.D., Macko, D., and Takahara, Y. Theory of Hierarchial, Multilevel Systems. New York: Academic Press, 1970.

[16] Minsky, M. "Steps Toward Artificial Intelligence." In E.A. Feigenbaum and J. Feldman, eds., Computers and Thought. New York: McGraw Hill, 1963.

[17] Minsky, M. and Papert, S. Perceptrons. Cambridge, Massachusetts: MIT Press, 1969.

[18] Nilsson, N.J. Learning Machines. New York: McGraw Hill, Chapter 5, 1965.

[19] Roberts, L.G. "Pattern Recognition with an Adaptive Network." IRE Int. Conv. Record, pt. 2, 66, 1960.

[20] Rochester, N., Holland, J.H., Haibt, L.H., and Duda, W.L. "Test on a Cell Assembly Theory of the Brain, Using a Large Digital Computer." Trans. IRE, PGIT-2, pt. 3, 80, 1956.

[21] Rosenblatt, F. "Two Theorems of Statistical Separability in the Perceptron." In Mechanisation of Thought Processes. London: HMSO, 1959.

[22] Samuel, A.L. "Some Studies in Machine Learning Using the Game of Checkers." In E.A. Feigenbaum and J. Feldman, eds., Computers and Thought. New York: McGraw Hill, 1963.

[23] Selfridge, O.G. "Pandemonium, a Paradigm of Learning." In Mechanisation of Thought Processes. London: HMSO, 1959.

[24] Stafford, R.A. "Multi-Layer Learning Networks." In J.E. Garven, ed., Self-Organizing Systems 1963. Washington: ONR, 1963.

[25] Stafford, R.A. "A Learning Network Model." In M. Maxfield, A. Callahan, and L.J. Fogel, eds., Biophysics and Cybernetic Systems. New York: Spartan Books, 1965.

[26] Uttley, A.M. "The Informon: A Network for Adaptive Pattern Recognition." J. Theor. Biol., 27, 31, 1970.

[27] Widrow, B. "Generalization and Information Storage in Networks of Adaline Neurons." In M.C. Yovits, G.T. Jacobi, and G.D. Goldstein, eds., Self-Organizing Systems 1962. New York: Spartan Books, 1962.

Appendix

The main argument of the paper is the heretical one that there is still a great deal to be learned from the study of self-organizing networks of neuronlike elements. In the context of artificial intelligence studies, the argument is perhaps more convincingly presented as follows than as in the main paper:

1. Relevance of SOS Studies

It has been argued that the computer programs produced by AI workers are subject to a fundamental limitation which is often expressed by saying they do not develop their own heuristics. Certainly the brain shows a degree of flexibility of behaviour which is far beyond anything shown by artifacts. The flexibility is such that people can engage in such activities as philosophy, mathematics, chess playing, or computer programing using brains which evolved as specialised organs of survival under relatively primitive conditions. One secret of the brain's success is undoubtedly its ability to switch rapidly among many different approaches to a problem. It is presumably because of this ability that people, even without special instructions, perform as well as they do in scheduling tasks such as the planning of school timetables or the operating of workshop or transport services. It has proved to be quite difficult to devise algorithms for computer programs to compete with human performance in these areas.

What is probably a highly significant aspect of the brain's versatility is its retention of its more primitive skills (or some of them) even when more advanced ones have been acquired. Even the purest of mathematicians is not completely helpless if he finds himself the sole survivor of a plane crash in the jungle. Certainly he is initially at a considerable disadvantage compared to people who have been jungle dwellers all their lives, but if he is lucky enough to make no fatal mistakes in his first few days, he will start learning the things they know.

There can be little doubt that human thinking depends on a multiplicity of mechanisms, of which formal linguistic reasoning is only one. Naturally these formal operations have received particular attention from workers in AI, since digital computers are themselves formal linguistic devices. The impossibility of studying all human thought processes in these terms is illustrated

by the reply of a chess champion who was asked how many moves in the look-ahead tree he considered before deciding his move in a game. His reply was reported to be: "Only one--the right one."

Poincaré [7] (see also discussion by Campbell [3]) refers to the mysterious nature of his own thought processes which led to his mathematical discoveries. Describing what happened when (contrary to his custom) he drank strong coffee late at night and could not sleep, he wrote: "Ideas rose in crowds; I felt them collide until pairs interlocked, so to speak, making a stable combination." He goes on to reflect on the mysterious nature of the filter which recognises some of these combinations as possibly useful and allows these to pass over the threshold of full consciousness. The filter presumably depends on something more fundamental, and, in a sense, more primitive than the formal linguistic reasoning with which it interacts.

Electrophysiological studies of the visual systems of higher animals provide some evidence for the coexistence of distinct mechanisms which come into play as required. Hubel and Wiesel [4] have found, in the visual cortex, neurons whose response is related in four main ways to the stimuli presented in the visual field. Some neurons, referred to as "concentric units" are influenced antagonistically by a small circular area and a concentric annular area in this field; these are the cells which receive inputs directly from the optic nerve (after its relay in the geniculate body). Then "simple units" respond to edges, dark bars, or light slits in the field, and must operate by combining the outputs of concentric units. Similarly, "complex units," which respond to movement of edges, slits or bars, combine the outputs of a number of simple units. "Hypercomplex units," which respond only to moving edges, slits or bars of limited length, must operate by combining the outputs of several complex units. One way in which visual perception works is through the following chain: concentric units to simple units, then to complex units and then to hypercomplex units. However, as Lettvin (of frog-vision fame, [6]) has pointed out, the whole mechanism revealed by these studies can be bypassed when required, otherwise it would be impossible to see the stars at night. (Hubel and Wiesel report that fibres go from units of all types, including concentric, into area 18 of the cortex for further processing.)

There is a case, then, for believing that a system to exhibit artificial intelligence should be able to drop back, when appropriate, to modes of behaviour which are of more general applicability than that which has evolved in connection with a specific task environment.

Some workers in artificial intelligence have tried to work at a very general level; work on conept formation by Amarel [1], Banerji [2], and others is at a much more general level than most approaches used in artificial intelligence. However, even the assumption that the operation of the system is to be described in terms of concepts and properties implies some restriction of generality as there are some tasks which are not readily described

in these terms. Examples are the acquisition of manual skill in, say, riding a bicycle or wood carving, and certain aspects of many tasks including that of satisfying the users of a question-answering system.

To be more general, it is necessary to choose some basic fabric to be modified by the learning process, and that of a network of neuronlike elements has some attractive features. One is that the networks studied may (but also may not) have some useful correspondence to real nervous systems.

2. Evolution by Stages

An approach which would be completely general would be to form a large network of model neurons, interconnected either randomly or according to a pattern, and to let it be modified by random mutations and selection according to a criterion of task fulfillment. Usually the available measures of task fulfillment will not be such that the system is "led in" by a hill-climbing process to a state in which it performs the task. Finding such a state must therefore depend on a search through a vast number of states produced by the random mutations, and, in fact, the approach is defeated by the "combinatorial explosion."

Living organisms are presumably the result of an evolutionary process which has continued over a much longer period of time than any artificial intelligence project can be allowed to consume. This process has to some extent defeated the "combinatorial explosion" effect by proceeding in stages, as the word "evolution" suggests.

It is not only the level of functional organization which has become greater at each stage; there has also been "adaptation to adapt." That is to say, at the successive stages there is selection of mechanisms which will facilitate further evolution. Finally the brain superimposes the results of learning during an individual's lifetime on those of the earlier "genetic learning." The brain does this with the benefit of the earlier evolution of mechanisms facilitating learning. The extreme versatility of the brain indicates that some of these mechanisms are very general in their applicability.

The aim of many workers who have tried to construct self-organizing networks has been to embody some of the special mechanisms so that adaptation may proceed without meeting a "combinatorial explosion." The adoption of these mechanisms immediately implies some compromise of the earlier insistence on extreme generality; some generality is traded for speed of adaptation. It is hoped that by a wise choice of the mechanisms embodied, a large benefit in speed can be obtained without serious infringement of generality.

3. Inductive Inference

The review which is attempted in the main paper is essentially of mechanisms which might be incorporated in networks to facilitate learning.

An aspect which is not explicitly mentioned in the main paper, nor in the papers it refers to, is the need for appropriate generalisation of the results of learning. It is likely that many of the workers quoted were tacitly assuming some kind of generalisation, since their aim is to produce networks which will learn complex tasks although the networks themselves are of modest size. Generalisation is needed, both for economy of representation and to allow inductive inference. It is only by condensing a large amount of experience into some fairly short description that a network can make appropriate responses to input configurations it has not previously experienced.

The problem of finding short representations which will allow inductive inference has been discussed by Banerji [2] in terms of predicate calculus, and by Amarel [1] in another formalism. The general approach of Banerji is being vigorously developed by Rothenberg [8].

The problem is far from simple, as it is not at all obvious what measure of "shortness" of representation is likely to be most conclusive to useful inductive inference. Banerji, in fact, argues (if I understand him correctly) that the measure of shortness is so much a function of the task environment that it has to be learned by experience.

Any scheme for self-organization in neural nets must have a tendency to seek short representations. Preferably, also, it should be able to evolve effective measures of shortness.

Methods for the automatic minimization of switching networks and finite automata are well known (see, for example, Kohavi [5]), and there is the possibility of an interesting field of study in relating these to the work of Rothenberg and others on concept formation. The switching- and automaton-theory approach is perhaps of limited value since the methods tend to rely on a global view of the network rather than the kind of locally acting mechanism which is usually visualised for self-organization. (What is more important in a practical application is that the amount of computation needed to let the mechanism operate is likely to become prohibitive for large nets if a method depending on a global view is adopted.)

Locally acting methods for the minimization of networks may in fact have much in common with the proposals for reduction of redundancy which were rather cursorily dismissed in the main paper. In a network which can undergo automatic simplification, it might be advantageous to let the network grow in complexity while new experience is being gained, and then to "digest" its

new structure into a simpler form when time permits. There have been some suggestions that the need for sleep by humans is related to some need to process sensory data, and that dreams are a side effect of the processing.

It certainly appears that the neural net approach still offers enormous and potentially fruitful areas of study.

References to Appendix

[1] Amarel, S. "On the Automatic Formation of Computer Program which Represents a Theory." In M.C. Yovits, G.T. Jacobi, and G.D. Goldstein, eds., Self-Organizing Systems 1962. New York: Spartan Books, 1962.

[2] Banerji, R.B. Theory of Problem Solving. New York: Elsevier, 1969.

[3] Campbell, D.T. "Blind Variation and Selective Survival as a General Strategy in Knowledge Processes." In M.C. Yovits and S. Cameron, eds., Self-Organizing Systems. Pergamon Press, 1960.

[4] Hubel, D.H. and Wiesel, T.N. "Receptive Fields and Functional Architecture of Monkey Striate Cortex." J. Physiol. 195, 215, 1968.

[5] Kohavi, Z. Switching and Finite Automata Theory. New York: McGraw-Hill, 1970.

[6] Lettvin, J.Y., Maturana, H.R., McCulloch, W.S., and Pitts, W.H. "What the Frog's Eye Tells the Frog's Brain." Proc. I.R.E. 47, 1940, 1959.

[7] Poincaré, H. "Mathematical Creation." In H. Poincaré, ed., The Foundations of Science. Translated by G.B. Halstead. New York: Science Press, 1913.

[8] Rothenberg, D. "Predicate Calculus Feature Generation." In T. Storer and D. Winter, eds., Formal Aspects of Congnitive Processes. New York; Springer, 1975.

A Computer Interview Procedure Which Reconstructs Generative Semantical Structures of Human Beings Using Modal Sets

Salomon Klaczko-Ryndziun and Karl-Heinz Simon

The isomorphy between the definition of a concept in classical logics and the definition of a set in set theory is applied to the computer-storage of judgments, originally formulated in natural language. For this purpose, the judgments are redefined as sets in intentional form.

The properties for such an intentional redefinition are classified either as necessary ones, which determine the invariant of the involved set, or as accidental ones. For n accidental properties p_i, each with a scale of m_i different values, imply a cyclic group of the order Πm_i as a transformation domain. This leads to the use of modal sets as an algorithmic tool for a generative semantic.

Stochastic redefinition of one or more necessary properties into accidental ones generates a more abstract superconcept than the original one. Random redefinition of one or more accidental properties into necessary ones generates a more concrete subconcept. The lattice of all the generable modal sets for a given vector of n properties would contain theoretically 3^n elements, since each property would have three possible states: necessary, accidental, or nonexistent. This is practically reduced by an amount of 2^n elements which would contain no necessary properties and which would be, therefore, so called "objects in themselves." From this sublattice, we extract the nonobject, having only nonoccurrent properties, so that our pragmatic lattice (hierarchical memory) would contain a generative potential of

$$A_n = \sum_{i=0}^{n} \left[\begin{matrix} n \\ i \end{matrix}\right] \cdot 2^{n-i} = 3^n - 2^n + 1 \quad ,$$

different modal sets, generable out of n different properties. (For $n = 5$, $A_n = 212$; for $n = 10$, $A_n = 58026$).

An interactive computer program, written in the list-processing language SIMULA, autonomously makes an interview with a human being used as experiment object (EP). The computer asks the EP to enumerate all the elements, organizations (persons, animals, objects, devices), involved in a given situation. Also, the properties--specified either as necessary or as accidental--of these elements are asked. The EP responds typing this data into the terminal which the computer uses for the interview.

A random subroutine produces intermittently a mix of properties of different objects and asks the EP if this mix (fictive object) corresponds to a real object in this or in another situation. If the answer is positive, the name of the new object is asked, and the object itself is stored in the cognitive memory. If the object can only occur under another situation, also the name of the new situation is stored, and another random subroutine can sometimes decide to ask questions about the objects of this other situation (an overlapping of situations is possible).

"Impossible" objects are stored in the "impossible situation." At the same time, the computer asks which attributes are in contradiction with which others in each "impossible object" in order to find the reason for the inconsistency of this object. Thus, if an attribute a is contradictory with an attribute b, then the negation of $\underline{a}$ must be consistent with $\underline{b}$ and may occur together with $\underline{b}$ in the field of necessary attributes of a real (existent or possible) object. Also, this type of element is being stochastically generated and presented to the EP for confirmation. A random generator decides--when new "impossible elements" are produced and presented to the EP to test--if it pays attention or if it is cognitively self-consonant. For this purpose, the cartesian product of the set of attributes with itself is used to mark those attribute pairs which were discovered to be contradictory ones.

Another strategy to promote man-machine conversation is the random selection of a set of names of elements to ask the EP about the common attributes over this set and, if possible, to ask if this set has a known name. The computer can also ask about other elements which, in the opinion of the EP, may belong to this random set. If these elements exist, their attributes will also be required.

A further strategy consists of random selection of a defined element and attachment to it of a nonprovided attribute. Thereby, it becomes possible either to confirm the self-consistency, or to discover a contradiction, or to state the incompleteness of the definition provided originally for the element.

Another subroutine contains the list of syllogistic forms, and it generates from randomly chosen elements and syllogisms different judgments, which are also offered to the EP for confirmation. Also chains of syllogistic inferences are generated from this subroutine.

All the above strategies are applied to expand or to correct systematically the lists of elements and attributes. Random generators decide in which sequence the different strategies are applied. For each element and property, an effective charge value, scaled between -100 and +100, is requested from the EP. This produces a projection of the motivational filters of the EP onto its own cognitive structure. Such a kind of motivational filter is of particular importance to another cognitive category which is also requested from the EP: change or transformation of elements or of situations. Also the feasibility of such a transformation, scaled between 0 and 1, is asked during the interview.

Dependent on the fact of whether a transformed element will remain in the original situation or not, the change will be conceived as an immanent or as a transcendent one. If certain elements are defined as instruments, being able to provoke a change in a situation, as long as these instruments interfere with the situation, then causal chains of transformations can be formulated. Cyclic transformation chains and sets of such chains (relationships) will correspond to stable situations (eventually subsituations of a situation). The simplest cyclic transformation, the flow of time, characterizes static relationships.

Stored thought structures of an EP concerning two or more cognitive fields and having different motivational charges are useful for the dynamic simulation of processes of self-contradiction and consciousness. Stored structures of two or more EP's about the same cognition field are useful for the simulation of cognitive intersubjectivity.

The system was implemented in the list-processing language SIMULA on the CD-3300-Computer of the Universität Erlangen-Nürnberg.

Cognitive Information Retrieval By Goal-Oriented Languages

Giuseppina Gini and Maria Gini

1. Introduction

Many programing tasks in artificial intelligence require the manipulation of a small data base, for example in question-answering and robot systems. These programs need to perform complex retrieval operations in the data base, and they also need some problem solving features.

Question-answering systems are complex systems with the capability of memorizing and retrieving information and with the possibility of inferring new information. A characteristic of these systems is to have a set of procedures which makes them capable of giving answers which are not explicitely present in the data base but are logical consequences of the facts stored there.

The information in the data base is not only about the facts but also about the relations between them; these connections have to be preserved in order to make the process of answer generation easy.

Another important aspect of question-answering systems is the facility of access to the system; an input language which is too formalized in a way not related to the subject has to be avoided in order to have a system which is easy to use.

The research in question-answering systems has been, for a period, related to the research in natural language understanding. The few results obtained at present in this difficult field and the computational weight of a natural language understanding system suggest leaving the natural language input in a question-answering system.

The idea developed in this paper is to allow an input which is formal but natural oriented. This is a feasible input which allows a natural like communication with the computer and doesn't present theoretical and practical difficulties.

The organization of the question-answering system in our proposal is discussed in three steps:

a) Input translation: this is the process which transfer the input language (natural-oriented) into the internal language in order to allow the typical question-answering functions.

b) Assertion: this is the process whose task is to introduce in the data base the new sentence and to define its links. This process must avoid the possibility of inconsistency in the information in the data base.

c) Answer extraction: this is the process for giving as output an answer to a question. The search is made in the data base to check whether the proposition proposed as question is a consequence of the data base or is in contradiction with it, or if the information isn't sufficient.

The very crucial choice is in the programing language for implementing the system; these steps are easy to program if the "deep structure" of the input language and of the data base sentences are expressed in the same language used for the program.

The most classical way to obtain these capabilities is based on first-order, predicate logic, theorem prover. The answer is viewed as a theorem to demonstrate and the data base is written in the form of expressions in the logical theory [3,8,22,23].

A more natural internal language is a PLANNER-like language. The system obtained is not very general, but it is suitable for representing and adding information expressed as procedures. The question is posed in form of "goal" to achieve, and the answer is then obtained by success or failure. During the execution, the achievement of the goal is attempted by a direct action; if that fails, it is then attempted by the introduction of some lower-level goals [13]. Many experiences in this procedural approach are analyzed, and the approach choice is selected [6,17,29].

In section 2, the central problem of artificial intelligence, i.e., knowledge representation in a computer, is discussed. The most important directions using formal logic and programing languages are analyzed.

In section 3, the project of a question-answering system is approached. The input language for such a system is proposed. The characteristics of a formal and natural-oriented language are outlined, and the problem of its realization is discussed. The implementation of a parser program is proposed as a way for translating the input language into the internal one.

In section 4, the problem of the memory organization is discussed, and some characteristics of existent goal-oriented languages are investigated. In particular, the steps of inserting new information and of extracting answers are examined.

Finally, in section 5, the most important research directions, now attempted by the authors in the general problem of programing tools for artificial intelligence, are proposed, and

their relationships with question-answering systems are pointed out.

In the appendix, there is a listing of a small system proposed as an example to this paper.

2. The Problem of Representing Knowledge in a Computer

The fundamental task of artificial intelligence is to represent and to use knowledge in order to automatically solve some problems [20]. Following the Nilsson definition, by reasoning in artificial intelligence we mean the major process involved in using knowledge-using it to make inferences and predictions, to make plans, to answer questions and to obtain additional knowledge.

The intelligent behavior of a system is also discussed in an important paper by McCarty and Hayes [16]. This behavior is based on an internal representation of the knowledge. The representation must account for two facts:

a) The factual knowledge about the world and the laws governing relationships and changes in it, which is the epistemological aspect.

b) The pragmatic knowledge necessary for solving problems, which constitutes the heuristic information.

In the design of such artificial intelligence programs, it is crucial to find a good formalism for this representation.

2.1. Assertions or Procedures?

Many recent works in artificial intelligence have been inspired by the discovery that a resolution-based theorem prover can be easily adapted to the generation of constructive answers to questions from data formulated in first-order predicate logic [8].

This sort of data base consists of a set of predicate calculus formulas. When the system is given an English sentence, it adds a predicate calculus formulation of the sentence to its data base. This method has been employed in question-answering systems and also in plan formation and automatic programing.

A different approach, begun with PLANNER language, is based on the idea of knowledge as being procedural rather than merely factual. Procedural embedding [12] means that any piece of knowledge has to be represented by a suitable procedure, which is executed by the system when it is relevant to the current problem. Hewitt argues that one must be able to discuss not just facts but also techniques for using them. This idea also supports the work of Winograd [29], whose system uses assertions for atomic facts and procedures for expressing formulas with quantifiers.

For a period, there was some controversy over whether knowledge should be represented assertionally or procedurally. The procedural approach began as one opposed to the logical approach; the basis of the criticism is related to the fact that the logical approach separates epistemological and heuristic information in axioms and fixed resolution strategy of theorem prover, while PLANNER-like languages allow one to control, partially or totally, the research of the solution.

This controversy seems to be settling down now to an acceptance of the value of a combination of assertional and procedural knowledge. There is not yet a theoretical basis for knowledge representation because this problem is not separable from the problem of using this knowledge.

Concerning heuristic information, there is also a lot of useful controversy about how to build intelligent systems. One direction is to use a universal system for all the tasks, for example, a theorem prover based on Robinson's resolution principle. An opposite proposal is to employ some adhoc methods; in this way each task should simply be solved by a program written in a suitable programing language.

2.2 Programing Languages for Artificial Intelligence

The programs for manipulating knowledge need to store, to have access to, and to manipulate lists of symbolic information. The first system for achieving several of these operations was obtained by list-processing languages, such as LISP [15].

Another common need for solving problems is the performance of operations such as search, expressions retrieval, and pattern matching. This exigency is the basis of a new generation of programing languages in which some of these operations are built into the languages themselves [5,11,18,21,27]. The prototype of these systems, in which the most typical features are already present, is Hewitt's PLANNER. It is a substrate for special purpose, theorem provers. It is also more data-base oriented than the preceding systems, and uses the idea of associating procedures with relation symbols, and executes these for storage, retrieval, and deduction.

Information can be represented in two different ways in PLANNER. First, a list of constantly called items can be stored in the data base by thassert function. This method is suitable for atomic facts which do not contain any variables or quantifiers. Second, information can be represented by theorems, i.e., programs called through pattern matching. Generally, theorems express laws valid in the world, and they allow obtaining new information from the assertions.

A PLANNER theorem is characterized by a pattern that represents the meaning of the theorem and that is very important in the process of system activation. A two-directions deduction is present in PLANNER in which there are consequent and antecedent theorems.

The most natural activation of a deduction process is the top-down, which is realized by consequent theorems. In this case, the theorem is written in a way that the body implies the pattern. If we want to demonstrate the pattern, we must execute with success the body of the theorem.

A PLANNER program is activated by assigning a goal to demonstrate; it uses a general high-level control mechanism, which is similar to a nondeterministic programing approach. If there are some different assertions or theorem patterns which can match with the assigned one, the system makes an arbitrary choice, and maintains the possibility of making another choice if there is a failure. This control mechanism is used in PLANNER interpreter (automatic backtracking), and is also modifiable in some ways by the user.

The criticism in this automatic control structure is one of the most interesting sources of new ideas in goal-oriented languages. A total user's control in his problem-solving strategies is realized in CONNIVER [18], and it is viewed as a central point in the project of new programing languages for artificial intelligence.

For an illustration of the other existent goal-oriented languages, there is a useful paper by Bobrow et al [1].

2.3 Interpreters and Proof Checkers

The processes of computation and deduction are substantially equivalent, though they often have been developed in different directions. The computation has been related to the construction of compilers and interpreters for programing languages, while the deduction has been developed in theorem provers for mathematical logic.

According to a paper by Hayes [9] an interpreter for a programing language and a theorem prover for a logical language are structurally indistinguishable. Theoretical results about this identity are present in the classical theory of computation.

In recent years, PLANNER and then its descendents have proposed again, in some very new aspects, the relations between programing languages and predicate lagic.

In the usual programing languages, the control information is sometimes represented implicitly (for example in the order of statements) and sometimes explicitly (for example in the procedure call). A conventional interpreter operates deterministically because it evaluates some definitions of functions which contain both the logical information on the function and the control information on the particular algorithm for computing that function.

In a first approximation, the difference between a theorem prover program and an interpreter is that in the latter case the

control is part of the input statements, while in the former the control is fixed in a theorem-prover strategy.

In PLANNER, the rejection of the theorem prover approach gives rise to a control structure that is, to a certain measure, problem dependent and modifiable; a total degree of responsibility for the user in defining his control structures is realized in CONNIVER [18].

We can represent the conventional meaning of computation and deduction in the following figure.

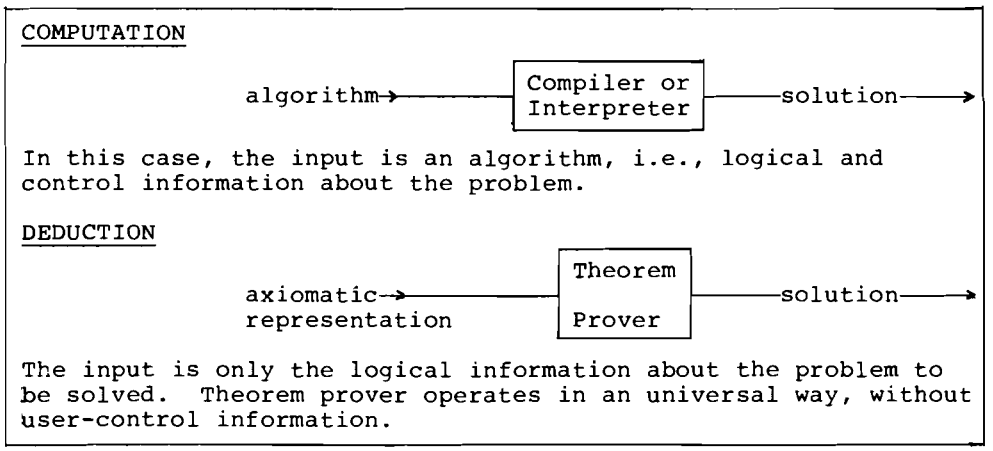


Figure 1.

PLANNER-like languages can be viewed as intermediate steps between the two approaches.

2.4. Why a Goal-Oriented Language?

The use of goal-oriented languages became prevalent in recent years as suitable tools in various artificial intelligence fields. In particular, these languages are employed for different reasons in question-answering programs.

One reason is their semantic power. Though some papers [10,14] point out that the behavior of such languages is analogous to that of theorem provers for first-order logic, they have some expressive advantages. For example, they are a simplified nucleus of higher-order logic languages, and, with them, one need not be forced to make distinctions among predicate, function, or variable symbols.

The interest in using higher-level logics for problem-solving activity has been pointed out by several researchers [4]. At the moment, there is the theoretical basis for a higher-order logic, theorem-prover system, but these results are not practicable. In using logical language, the solution is to simulate the higher-order logic behavior in first-order, predicate calculus.

The criticism about the rigid structure of theorem provers caused interest in the programability of the control structure of the system and in implementing new systems with the richness of programing languages. Theorem provers can be viewed as a first step in this evolution of artificial intelligence systems.

There are also some problems of efficiency, which are difficult to solve in classical theorem provers, and which have satisfiable solutions in goal-oriented languages. One of them is the frame problem. Though some solutions have been proposed [25], it is still a weak point in logic systems.

Another important feature for a question-answering system is its modifiability, i.e., the degree to which new information modifies the system. This feature, as illustrated in another section, is guaranteed in a simple way by goal-oriented languages.

3. Input Language and Translation

The choice of the input language for a question-answering system has to be connected with the general problem of representing knowledge in a computer in order to allow some reasoning activities.

In a paper by McCarthy and Hayes [16], some possible criteria for adequacy of a notation are outlined, and the following distinction is made: "A representation is called epistemologically adequate for a person or a machine if it can be used practically to express the facts that one actually has about the aspects of the world. A representation is called heuristically adequate if the reasoning processes actually gone through in solving a problem are expressible in the language."

For a question-answering system, an epistemologically adequate representation is, of course, the natural language, but there are also some other representations which are more suitable, we think, for the task of the question-answering system. For example, first-order, predicate logic or a PLANNER-like language are epistemologically adequate representations. Besides, a PLANNER-like language integrates epistemological and heuristic information and supplies an unique notation for expressing these together.

In order to discuss the choice of the input language for a question-answering system, we divide the problem of representing data into three parts [8]:

a) Determination of the relevant semantic content of the data. For example, we may decide that the semantics of the sentence, "John is father of Jim," is expressed by the binary relation "father of" applied to the objects John and Jim.

b) Choice of a language in which to express this semantic content. For example we may use the notation of mathematical logic.

c) 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 elements are arguments. A crucial factor in selecting this internal language is that one must be able to construct an answer computation program which can effectively produce correct answers. For this reason, the problems of input language and of internal language are very closely related.

3.1 A Formal Language as Input

The use of predicate logic as input language for a question-answering system has been proposed by Green, [8]. His system uses as input first-order, predicate logic and, as deductive mechanism, an internal language and a theorem prover based on resolution principle.

Some of the problems that one encounters while expressing natural language information in predicate logic are outlined by Sandewall [24].

The first problem is in reducing the natural language constructions, which are, in a certain sense, higher-order level, to first order.

For example, if we want to represent the sentence, "m is expensive," by the predicate expensive (m), then the sentence, "m is more expensive than n," might be expressed by the function "more (expensive) (m,n)." More is a second-order function which maps a monary first-order predicate into a binary first-order predicate.

The use of higher-order logic is useful in terms of human understanding, but is very difficult in programing.

The technology of automatic theorem prover has been developed slightly in theorem provers for higher-order logic.

There is a proposal for using these logic theories for plan formation [4], but it is an exception. In general, the tendency is to use present theorem provers for simulating higher-order logic.

Also, the higher-order construction in natural language should be expressed directly in first-order predicate calculus. The method is to reexpress what used to be predicate as individual, and to use a simple application predicate.

The first sentence of the previously proposed example is then expressed by, "Is (m, expensive)," and the second one by, "is (m, more than (expensive, n))," where "more than" is a function,

more than: {properties x objects} ⟶ {properties}.

In this way, the representation of sentences is obtained by the definition of a suitable set of functions, relations, and axioms.

An automatic translation from a simplified natural language to the first-order representation is proposed by Sandewall [24]. The well-known frame problem is present in these formal approaches. Several authors have discussed the frame problem and proposed solutions for it, particularly Raphael, Fikes and Nilsson (1971), McCarthy and Hayes (1969,1971), [25].

The solution to this problem, obtained by Hewitt with PLANNER, is very interesting because in his system the representation of the world is updated in an automatic and computationally simple way.

3.2 The Natural Language as Input

The first interesting example of a question-answering system with natural language input can be viewed in the natural language understanding system of Winograd [29].

This system contains the syntactic, semantic, and deductive capabilities required for representing information about a world of blocks. The key idea, which is typical of the procedural embedding thesis, is that descriptions in English can be translated into descriptions in the form of programs.

The system, written in LISP and MICROPLANNER for PDP-10, operates with a vocabulary of 200 words. It is very interesting to examine the organization of the system and its storage allocation, which are illustrated in Figures 2a and 2b.

The system occupies 80K words of core with about 12K words of free storage. All the information necessary for the deduction requires 15, 6K words, while the information necessary for understanding the natural language requires 36K words.

An evolution of Winograd's system can be viewed in TOPLE [17]. This system attempts to understand new sentences about a simple world by using a set of programs which embody a logical model of that world. It doesn't deal directly with English sentences but interprets simple semantic structures such as might be produced by a natural language parser.

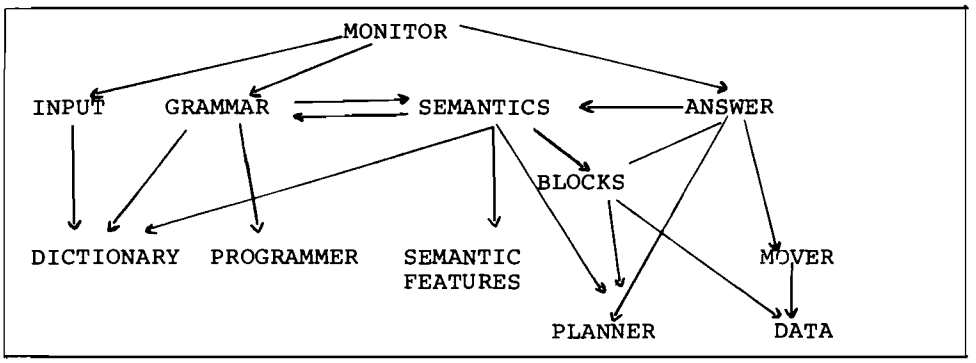


Figure 2a.

	Parser	Semantics	Deduction	Other
Interpreters 26.1 K	PROGRAMMER 5.8 K		PLANNER 5.5 K	LISP,DISPLAY 14.8 K
Knowledge of English 22.5 K	GRAMMAR 7.3 K	SEMANTICS 15.2 K		
Knowledge of subject 16.5 K	DICTIONARY 1.7 K	DICTIONARY 6.0 K	BLOCKS 8.8 K	
Data for Scene 2.5 K			DATA 1.3 K	DISPLAY 1.2 K
Total 67.6 K	14.8 K	21.2 K	15.6 K	16.0 K

Figure 2b.

There are now many other works on natural language understanding problem and several systems based on an internal language which is LISP or PLANNER-like or formal [24]. In general, these systems are able to understand only a small subset of the natural language, and there are many problems, theoretical and practical, not yet solved, connected with the increase of these systems.

All the works done in this field can be interesting for a question-answering system, but this subject isn't the main argument of this paper. Besides, our opinion is that a natural language comprehension problem is not related to question-answering problems; instead, it is possible to solve the problem of input in quite a satisfactory way by using a particular language, which we call "natural oriented."

This "natural-oriented" language maintains the advantages of the formal languages, since its "deep structure" is one of the internal language. It also has the advantages of the natural language in that it is easy and natural to express the information and the questions in it. We shall now explain our proposal.

3.3 A Natural-Oriented Language as Input

A PLANNER-like language can be viewed as a natural-oriented language. In PLANNER, in fact, it is possible to express directly some information in natural language which corresponds to a higher-level logic, i.e., the distinction between predicate variables and functions doesn't exist without having a formal language and all the related problems.

For this reason, a pattern in PLANNER is very similar to an expression in natural language. The previous example can be expressed by the pattern, "(m IS-EXPENSIVE)" and "(m IS-EXPENSIVE-MORE-THAN n)."

The way in which the information is expressed is important for the deductive process because every pattern represents a piece of knowledge or is a tag of a memory container, which allows the deduction of the information expressed in its tag. Therefore, it is important to have simple patterns and to have the possibility of matching the patterns of the theorems or assertions with those of the questions posed to the system.

It is also possible to have a system that is more natural-oriented than PLANNER without introducing the problems of the natural language.

The idea of this input for a question-answering system is present in CROMOS, a system developed at Austin [2]. The interesting aspect of this approach is in the translation from the external to the internal language. 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, and, after the evaluation, a small unparser program gives the output in an English form.

Also Davies's system is interesting along this line. The system has a very limited English input, and a parser translates it into POPLER, a PLANNER-like language [5]. In this system, it is possible to use negation and "quantifier" words, such as: each, every, any, all, some, a, not, there is, no one, something, and so on [6].

The response to any question takes place in two stages: the question is compiled into a piece of program which represents the meaning of the question, then the program runs and generates the answer. The system automatically eliminates the old items which conflict with new statements when new information is added to it. In Figure 3, the sequence of the actions is illustrated.

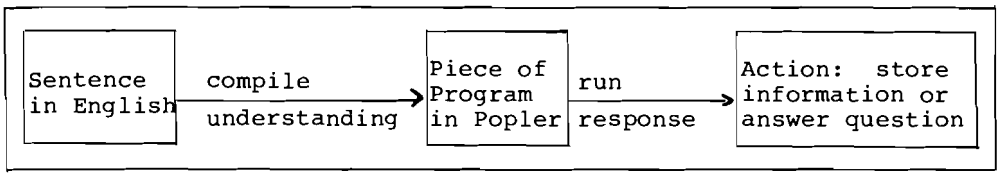


Figure 3.

Along this line we are now proposing as a natural-oriented language the actual implementation of PLANNER, called MICROPLANNER, with a suitable parser program [27]

The proposed parser can be understood looking at the parser which allows the use of LISP in a more natural way, employing the typical words of ALGOL, in a form which is called MLISP.

Figure 4 illustrates an example of a function in MLISP and its translation in LISP.

In a similar way, we are thinking of constructing a suitable parser for MICROPLANNER, which allows the reduction of the number of the necessary parenthesis--a very troublesome aspect of LISP!--and the introduction of some key words useful to the application of the language, such as the words introduced in Davies's system.

The insertion of the parser in the MICROPLANNER interpreter is very natural and doesn't present any particular problems. The fundamental loop of the interpreter is based on the application of the evaluation function THVAL to the READ expression with the values of the variables memorized in the stack THALIST. It is sufficient to substitute for the call of the function READ, the call of the function PARSER, which activates the program for realizing the parser. This change is illustrated in Figure 5.

3.4 MICROPLANNER as Inout for a Small Question-Answering System

We are now proposing, for example, a small system realized by MICROPLANNER, which memorises some information and extracts answers about the relationships in a family. For this system,

```
UOM-STANFORD MLISP
BEGIN
      NEW VAR;
      VAR$ = 10;
      WHILE VAR$ > 0 DO
            BEGIN
                  PRINT VAR;
                  VAR$ = VAR - 1
            END;
      PRINT > >;
      PRINT >ALL DONE>
END.
RESTART
****
LIST CODE? (Y OR N)
Y
(CSETO RESTART
(LAMBDA NIL
         (PROG < VAR >
               < SETQ VAR 10 >
               < & WHILE 'PROG2 '(GEQUAL VAR 0)'
                 (PROG MIL
                       <PRINT VAR>
                       <SETQ VAR (SUBI VAR)>)>
               < PRINT ' > >>
               < PRINT ' >ALL DONE >>)))
0 ERRORS DETECTED,        0 FUNCTIONS REDEFINED
END MLISP.  TIME:  244 MSEC.
```

Figure 4.

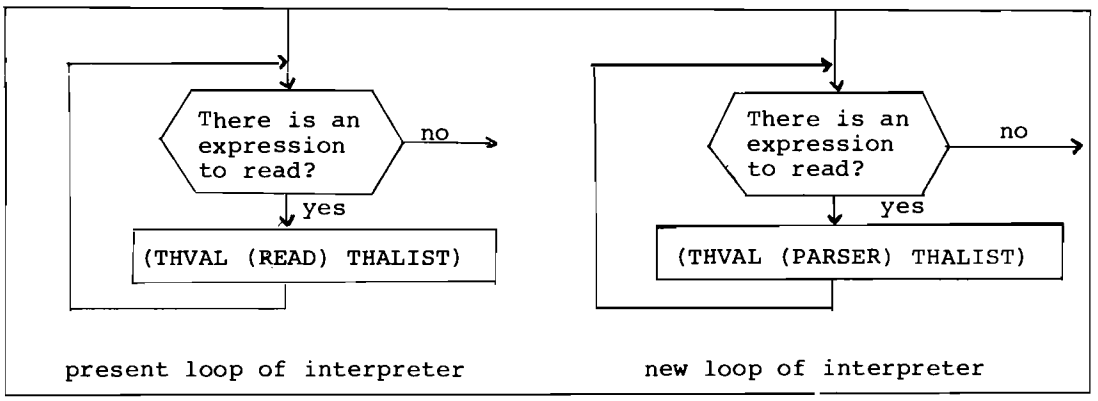


Figure 5.

we have employed MICROPLANNER both as input and as internal language, but, at this point, we are mainly interested in the problem of input language, i.e., the language in which new information is introduced in the system and questions are proposed.

The insertion of new information is realized in a standard way by assertions such as:

(THASSERT (x SON-OF y z) $T) ,

(THASSERT (x DAUGHTER-OF y z) $T) ,

Whose obvious meaning is: " x is son (daughter) of y and z. The father is y, and the mother is z."

The symbol "$T" has the task of indicating to the system that it is necessary to call some suitable theorems, which are the "antecedent" theorems whose pattern matches the pattern of the assertion. In this system, there are two theorems for each type of assertions:

a) One inserts in the data-base the information that "y is husband of z" and "z is wife of y." (The system does not foresee the divorce, but it is possible to introduce it!)

b) The other checks the data base in order to avoid inconsistency. For example, if there is the assertion, "x SON-OF y z," and we make the assertion, "x SON-OF yº z," the system, in order to avoid inconsistency, erases the previous assertion and gives a message to the user.

It is also possible to introduce the new information in many different ways by using other relationships; but, in this case, it is necessary to have a lot of suitable theorems in order to check the data base for inconsistency and also in order to try to express the information in the standard form. In fact, the standard form is very appropriate for all the following deductions and also for avoiding redundancy in the data base.

The definitions of the relationships are yet to be defined in the system, but it is possible to introduce some other ones with some suitable theorems. We examine in the next section the way in which to define the theorems.

The questions are posed to the system in the form of goals. It is possible to require information about the relationships between different persons in different ways, and to have different answers.

A typical form of the question is:

(THGOAL (x rel y) $T) ,

whose meaning is "if x is in the relationship rel with y, then the answer is the pattern of the goal, else the answer is NIL." It is important to note a limitation of the PLANNER system: the answer is "NIL" if the relationship isn't valid or if the system does not have enough information.

Another useful form of the question is:

(THPROG (x) (THGOAL (x rel y) $T) (THRETURN x)) ,

whose meaning is, "if a value of x exists, by which x is in the relationship rel with y, then the answer is the value of x or else the answer is NIL," i.e., this form acts like an existential quantifier on the variable x. In this case, the answer is determined by the expression which is the argument of the function THRETURN, and it is also possible to have a fuller answer.

An interesting possibility is given also by the form:

(THFIND (min max result) expression (varlist)

(step1)... (stepN)) ,

whose meaning is, "find a list of objects whose number is at least equal to min and at the most equal to max, obtained by the substitution for the variables in the expression, the variables which cause the program (step1).. (stepN) to succeed." The (min-max result) can be substituted, for example, by ALL when the names of all the persons which are in the relationship rel with the assigned one are required.

In addition, it is also possible to use some Boolean primitives, as THAND, THOR, THNOT, THAMONG, and THCOND, in order to have different combinations of the goals.

Some shortened forms for the main functions are often employed, for example:

(THASSERT) ⟷ $A

(THGOAL) ⟷ $G

(THTBF THTRUE) ⟷ $T .

The last expression is the simplest way to indicate to the system the necessity of examining not only the data base of assertions but also the theorems. There are other ways to give a more precise indication to the system. For example, we can give the name of the theorem to use or a filter, such that only the assertions of theorems which pass through the filter are employable. These informations, which are control informations, permit modifying the deductive process.

There also some other functions which are useful for the user. For example, THSTATE is a function which permits knowing

the state of the data base, i.e., it gives the names of all the theorems and all the assertions present in the data base. It is also possible to require only the assertions or the theorems of a specified type. TIME, for example, is a function whose value is the total execution time, expressed in msec. A third function, MEMORY, is one whose value is the number of words of memory currently being used from available memory.

For the relationships illustrated in Figure 6, we have employed the system. (The definition of the system, the insertion of the informations, the questions and the answers with the execution time and the memory occupation are in the appendix.)

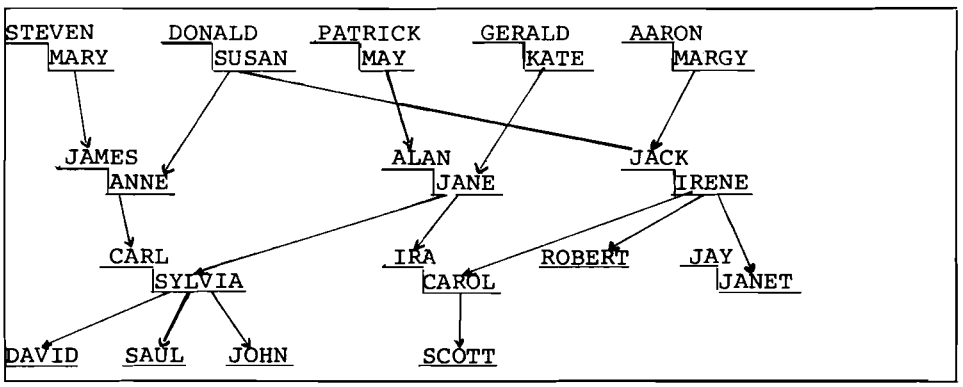


Figure 6.

4. Organization of the Activity in a Question-Answering System

In this section, we examine the problems related to the organization of the activity in a question-answering system, i.e., two fundamental steps:

a) the introduction in the data base of new information

b) the extraction of the answer to a question.

These two aspects, which are related to the memory organization and to the deductive process, are closely related, and they must be considered together. A fundamental problem in the design of a question-answering system is to decide "what, how, and where" the information has to be stored in order to have a suitable deductive process for the answer generation.

4.1 Memory Organization and Deduction

An important controversy among the designers of question-answering systems concerns how much knowledge is necessary to the program. At one extreme, there are some researchers who insist that the program should be given only some fundamental premises from which it must derive every bit of intermediate knowledge necessary in order to arrive at an answer. At the other extreme, there are some researchers who think that the programs must be explicitly provided with answers to all problems.

We think that it isn't possible to give a unique solution to these choices because in every system there are different problems and different requirements that determine its structure; also the computer and the programing language employed, the amount of data, the type of questions, etc., have to be considered.

In order to clarify this question it is important to emphasize the distinction between a general and a special-purpose question-answering system.

The goal in designing a special purpose question-answering system is to achieve good performance, measured in terms of running speed and memory utilization. In this case, the best approach is to construct a special data base and subroutines optimized for the particular area.

A general question-answering system, on the other hand, has to be constructed in such a way as to allow the addition of varied subject areas, and also to permit this during the process of answering a question. In this case, the main problem is not the optimization of time and memory [8].

We can consider a question-answering system based on an internal language such as predicate calculus and on a theorem prover as a general system in the sense that the user must provide only the factual knowledge about his domain and not the control knowledge about the subroutines which operate on the data base.

A question-answering system based on an internal language such as PLANNER is considered, on the contrary, as a special purpose system because the particular procedures necessary for every domain have to be written, and the user can organize his data base and the deductive process in the best way.

We don't consider now the totally special purpose systems, i.e., the systems totally constructed for a particular problem, because the internal language, the whole memory organization, and the deductive subroutines are designed for the particular domain and aren't employable for other domains.

We think that the second approach based on a language such as PLANNER is the most suitable for a question-answering system

because it unifies the flexibility and modifiability, typical of a general system, with the possibility of optimization given by the introduction of the control information and by the elimination of the problems connected with the use of the formal logic.

An important problem is related to the inconsistency of the information stored in the data base. In formal logic systems, the new information is checked for consistency before acceptance, while in nonformal systems there are no controls, and this task is given to the user.

Another important problem is related to the average amount of computation necessary to answer a question. One obvious measure of difficulty is the average distance of the answer from the question, measured, for example, in terms of the number of steps of inference necessary. This measure can be called depth of the question [8].

Another factor contributing to the search effort is the number of different questions that are answerable. Here we again find the initial controversy about the amount of information necessary to the system, and also the relation between the memory organization and the deduction process, which is the fundamental subject of this section.

In fact, to increase the number of different questions answerable, without increasing the depth of questions, it is possible to increase the size of the data base, or else to expand the capabilities of the answer computation mechanism, or at least to find a reasonable compromise between the two possibilities.

A very interesting feature is also the degree to which new information modifies the system. As the new information is entered, the performance of the system is modified. The new information can have an effect on how the questions are answered; for example:

a) The new information provides the answer to a new question.

b) The new information provides the information needed to get the answers to a new class of questions.

c) The new information provides a new procedure for answering a class of questions.

d) The new information modifies the representation of the information.

e) The new information modifies the strategies of the program.

The modifiability is realized in a very simple way in a system based on a PLANNER-like language. This aspect will be illustrated later.

4.2 Features of LISP and MICROPLANNER

In LISP, there is an interesting feature which permits memorizing of relations between different objects and which can be proposed as an example of the memory organization.

This feature is related to the use of the property list (p-list). In LISP, in fact, each atom can have a list of properties, i.e., a list of couples (indicator, value) which characterize the atom.

When we have a relation such as, "Jane, daughter of Gerald and Kate," we may place on the p-list of the atomic symbol, "Jane," the value, "(Gerald Kate)," under the attribute "daughter." The atomic symbol provides an entry point to the information, "Jane daughter of Gerald and Kate." The first argument of the relation, "Jane," is not stored explicitly with the relation, but is implied by the fact that the attribute-value pair occurs on the p-list of "Jane."

It is clear, at this moment, that in this way it is easy to find the information about the parents of Jane, but it is difficult to find the information necessary for answering a question about the name of "the daughter of Gerald." It is necessary, therefore, to have some cross references. This idea is on the basis of the memory organization in MICROPLANNER.

Assertions and patterns are stored on a list on the p-list of the items which appear in them; thus each item points to all the assertions and theorems in which it appears. Occurrences of variables are stored on the global atom named THVRB, just as if THVRB were an item at each variable occurrence.

An assertion is consigned to NIL or to something that is a property of the assertion. For example the assertion,

(THASSERT (JANE DAUGHTER-OF GERALD KATE)) ,

yields the structure ((JANE DAUGHTER-OF GERALD KATE)) which appears within the property of the items JANE, DAUGHTER-OF, GERALD and KATE under the attribute THASSERTION.

All the assertions or all the theorem names which have assertions or patterns of a certain length and which have an occurrence of a certain item at the same position within their assertions or patterns are on a list which is prefixed with two integers. The first integer is the length of the assertions or patterns, and the second integer is the count of the assertions or theorem names in this list. In the previous example, we have in the p-list of the atom DAUGHTER-OF:

(4 1 ((JANE DAUGHTER-OF GERALD KATE))) .

If we have defined a theorem FATHER, whose pattern is (x FATHER-OF y), we find, in the p-list of the atom FATHER-OF,

the structure--(3 1 FATHER).

All the previous structures which have a certain item occurring in the same position are posed in another list, prefixed with an integer which is the occurrence position of that item. In the example we have,

(2 (4 1 ((JANE DAUGHTER-OF GERALD KATE))))

(2 (3 1 FATHER)) .

At the end, all these lists are kept sorted into one of four possible lists, with a NIL consigned on the front, depending on whether the occurrence came from an assertion or from one of the three kinds of theorems. In this way, we have the structures which are illustrated in the appendix.

The deductive process in MICROPLANNER is closely related to this memory organization. The mechanism for the deduction is in the theorems of CONSEQUENT type. Each theorem of CONSEQUENT type is characterized by a pattern, like (x FATHER-OF y), and by a sequence of instructions, as in the previous example, (THOR (THGOAL (y SON-OF x z)) (THGOAL (y DAUGHTER-OF x z))). The theorem is written in a way such that "the body implies the pattern"; if we want to demonstrate the truth of the pattern, we must demonstrate, i.e., execute succesfully, 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 given because it is possible with these instructions to call other theorems. It is possible also to guide the search process by giving some information about the theorems to call.

When a question is posed to the system in form of goal, the system researches by pattern matching among the assertions (and later among the theorems) the piece of knowledge which allows the question to be answered. If there are different assertions or theorems whose pattern matches the one of the goal, the system makes an arbitrary choice, backing up and trying another automatically if one of them leads to a failure. In this process the importance of the organization of the knowledge previously discussed is particularly clear.

The solution proceeds normally in top-down or goal-oriented way. It reduces problems to subproblems, with the goal of reducing the original problem to a set of solved subproblems.

There is also the possibility of bottom-up behavior. In this case, new assertions are derived from the old ones with the goal of deriving a solution of the original problem. This behavior is realized by the use of ANTECEDENT theorems and of ERASING theorems, but it isn't suitable from the point of view

of memory and of time. In general, the CONSEQUENT theorems are employed for making the deductions, while the ANTECEDENT theorems allow expanding an assertion and ERASING an erasure.

From the point of view of the modifiability, a MICROPLANNER system is very flexible in a natural and simple way.

a) With the introduction of a new assertion, it can provide the answer to a new question.

b) With the introduction of a new theorem, it can provide the answer to a new class of questions.

c) With the introduction of a new theorem, whose pattern is the same as an existent one, it can provide a new procedure for answering a class of questions.

d) With the request of erasing present information (assertion or theorem) or of modifying it, it can modify the representation.

e) With the request of modifying a theorem, it can modify the strategies of the program.

4.3. Organization of a Small Question-Answering System

The proposed system, which allows answering questions about the relationships in a family, is totally realized in MICROPLANNER (compiled version) and runs on UNIVAC 1108.

The knowledge is memorized in:

a) assertions, which define the relationships "SON-OF" and "DAUGHTER-OF",

b) theorems of CONSEQUENT type, which allow deducing the other relationships on the basis of the informations supplied by the assertions,

c) theorems of ANTECEDENT type, which define the relation "WIFE-HUSBAND" and which check the data base for inconsistencies.

An interesting feature of this system is related to the inconsistency problem. This is a very difficult problem to solve in MICROPLANNER because the system isn't a formal system and therefore doesn't provide this analysis. In this case, the choice of having a standard form for the assertions in input allows us to define a suitable ANTECEDENT theorem which erases the assertion nonconsistent with the newly introduced one and gives a message to the user.

In the appendix, this system is shown with particular attention given to the execution times, memory occupation and

organization, and with some questions and their related answers.

There is another important note: the way in which the theorems are defined is very important for the deductive process. Below is a simple example.

We can define a theorem GRANDFATHER in this way:

```
(PUT 'GRANDFATHER' 'THEOREM' (THCONSE (X Y Z)
(x GRANDFATHER-OF y)
(THGOAL (x FATHER-OF z) $T)
(THOR (THGOAL (z FATHER-OF y) $T)
(THGOAL (z MOTHER-OF y) $T))))  ,
```

or in this way:

```
(PUT 'GRANDFATHER' 'THEOREM' (THCONSE (X Y Z V W)
(x GRANDFATHER-OF y)
(THGOAL (z w PARENTS-OF y) $T)
(THOR (THGOAL (z SON-OF x v))
(THGOAL (w DAUTHTER OF x v))  ,
```

for achieving the goal:

```
(THFIND ALL x (X) (THGOAL (x GRANDFATHER-OF ROBERT) $T))  .
```

With the first theorem, 99 subgoals are activated, while with the second only 6.

5. Conclusions and Further Goals

There are now two important directions for developing the outlined results. The first one is based on an improvement of the existent systems in order to have a more suitable system for the question-answering problem.

In this paper, we have not discussed a complete question-answering system, but only some ideas for organizing such a system, and we have shown some choices for the internal and external language. The implementation of the proposed language can be obtained by writing a parser program, which allows a natural-oriented input language. A simple syntax is sufficient in order to have a reasonable facility for the user, and there are no theoretical difficulties.

Further important goals can be outlined not only for the question answering, but also for a general improvement of the artificial intelligence software.

At the moment, we are thinking of inserting the parser program in MICROPLANNER interpreter, but our intention also is

to employ another language equipped with a richer and more programable control structure. For this reason, we are now studying the use of MAGMALISP [19], a LISP-like language with a general control structure implementing the Bobrow and Wehbreit model for control structures.

MAGMALISP is intended as a machine language for artificial intelligence. Its use can produce some interesting results because it is possible, for example, to write in it an interpreter of a simple goal-oriented language whose behavior is very similar to the CONNIVER one.

The interest is in the fact that the writing of this interpreter is simpler in MAGMALISP than in conventional LISP. In this way, it is possible to have a tool which acts like the bulk of CONNIVER, but which is simpler and therefore, intelligible. We remember that a very important problem of CONNIVER is related to its semantics [7].

The second direction is related to the project of new systems, and it is based on some new results obtained in software and in computational logic.

The results are related to the analysis of the relations between computation and deduction, given by Kowalsky [14], and to the investigation of the sematic meaning of the representation formalism by Hayes [10].

In order to obtain a new qualitative step in software, it can be useful to formalize some concepts of model logic; this idea is proposed again by the implementation of FOL [28], a nonresolution theorem prover, which also can be applied to some extensions of the predicate calculus. The mechanization of nonclassical logics is then proposed again.

The criticism in classical theorem provers and the experience in goal-oriented languages and control structures models also can be a useful basis for the implementation of a new goaloriented language--one semantically more rich than first-order, predicate logic, more rigorous than PLANNER [97], and with a programable control structure which can interact with the factual information.

References

[1] Bobrow, D.G., and Raphael, B. "New Programing Languages for Artificial Intelligence Research." Tutorial lecture, 3rd IJCAI. Stanford, California, 1973.

[2] Bruce, B.C. "A Model for Temporal References and Its Application in a Question-Answering Program." Artificial Intelligence 3, 1972.

[3] Colmerauer, A., et al. "Un système de communication homme machine en français." Rapport du groupe de Recherche en Intelligence Artificielle, UER de Luminy. Université d'Aix Marseille, France, 1973.

[4] Darlington, J.L. "Deductive Plan Formation in Higher-Order Logic." In Moltzer and Michie, leds, Machine Intelligence, vol: 7. Edinburgh: Edinburgh University Press, 1972.

[5] Davies, D.J.M. "POPLER 1.5 Reference Manual." TPU Report 1. Edinburgh: Edinburgh University Press, 1973.

[6] Davies, D.J.M. "Representing Negation in a PLANNER system." Proc. AISB Summer Conference, University of Sussex, England, 1974.

[7] Gini, G., and Gini, M. "CONNIVER Programs by Logical Point of View." Proc. MFCS 1975, CZECHOSLOVAKIA, 1975.

[8] Green, C.C. "The Application of Theorem Proving to Question-Answering System." Tech. Report CS 138, Memo AI-96. Stanford, California: Stanford University Press, 1969.

[9] Hayes, P.J. "Computation and Deduction." Proc. MFCS 1973, Czechoslovakia, 1973.

[10] Hayes, P.J. "Some Problems and Non-Problems in Representation Theory." Proc. AISB Summer Conference, University of Sussex, England, 1974.

[11] Hewitt, C. "PLANNER: A Language for Proving Theorems in Robots." Proc. 1st IJCAI. Washington, D.C., 1969.

[12] Hewitt, C. "Procedural Embedding of Knowledge in PLANNER." Proc. 2nd IJCAI. London, 1971.

[13] Hewitt, C. "Description and Theoretical Analysis (using Schemata) of PLANNER: A Language for Proving Theorems and Manipulating Models in a Robot." AI Memo 251. Cambridge, MASSACHUSETTS: M.I.T. Press, 1972.

[14] Kowalski, R. "Logic for Problem Solving." Dept. of Computational Logic Memo 75. Edinburgh: Edinburgh University Press, 1974.

[15] McCarthy, J., et al. "LISP 1.5 Programmer's Manual." Cambridge, Massachusetts: M.I.T. Press, 1965.

[16] McCarthy, J., and Hayes, P.J. "Some Philosophical Problems from Standpoint of Artificial Intelligence." Machine Intelligence, vol 4. New York: Elsevier, 1969.

[17] McDermott, D. "Assimilation of New Information by a Natural Language Understanding System." Workshop on Information Processing and Psychology, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 1973.

[18] McDermott, D.V., and Sussman, G.J. "The CONNIVER Reference Manual." A.I. Lab. Memo, 259a. Cambridge, Massachusetts: M.I.T. Press, 1972.

[19] Montangero, C., Pacini, G., and Turini, F. "MAGMA LISP: A Machine Language for Artificial Intelligence." Internal Note I.E.I. and I.S.I. Pisa, Italy, 1975.

[20] Nilsson, N.J. "Artificial Intelligence." IFIP Congress 1974. Stockholm, Sweden, 1974.

[21] Rulifson, J.F., Derksen, J.A., and Waldinger, R.J. "QA4: A Procedural Calculus for Intuitive Reasoning." A.I. Center, Tech. Note 73, Stanford Research Institute, Menlo Park, California, 1972.

[22] Sandewall, E.J. "A Programming Tool for Management of a Predicate Calculus Oriented Data-Base." Proc. 2nd IJCAI. London, 1971.

[23] Sandewall, E.J. "Formal Methods in the Design of Question-Answering Systems." Artificial Intelligence, 2, 1971.

[24] Sandewall, E.J. "Representing Natural Language Information in Predicate Calculus." In Maltzer and Michie, eds., Machine Intelligence, Vol 6. Edinburgh: Edinburgh University Press, 1972.

[25] Sandewall, E.J. "An Approach to the Frame Problem and Its Implementation." In Meltzer and Michie, eds., Machine Intelligence, Vol 7. Edinburgh: Edinburgh University Press, 1972.

[26] Sussman, G.J. and McDermott, D.V. "From PLANNER to CONNIVER. A Genetic Approach." Proc. FJCC, 1972.

[27] Sussman, G.J., Winograd, T., and Charniak, E. "MICROPLANNER Reference Manual." A.I. Lab., AI-TR-203. Cambridge, Massachusetts: M.I.T. Press, 1970.

[28] Weyrauch, R.W., and Thomas, A.J. "FOL: A Proof Checker for First Order Logic." A.I. Lab., Memo AIM-235. Stanford, California: Stanford University Press, 1974.

29] Winograd, T. "Procedures as a Representation for Data in a Computer Program for Understanding Natural Language." MAC TR-84, (Thesis). M.I.T. Cambridge, Massachusetts.

Appendix

```
?THE VALUE IS THE NUMBER OF WORDS OF MEMORY CURRENTLY BEING USED
? FROM AVAILABLE MEMORY
(MEMORY)

VALUE:
2

EXPRESSION TO EVALUATE:
? THE VALUE IS THE TOTAL EXECUTION TIME IN MSEC
(TIME)

VALUE:
9

EXPRESSION TO EVALUATE:
? THE VALUE IS THE OCTAL ADDRESS OF THE LAST WORD OF MEMORY
? CURRENTLY ALLOCATED TO THE SYSTEM
(GROW)

VALUE:
105777Q

EXPRESSION TO EVALUATE:
(LOAD '(LISP.PLNR))

VALUE:
PLNR LOADED

EXPRESSION TO EVALUATE:
? THE SYSTEM ADD 30 BLOCKS OF 512 WORDS TO THE MEMORY AVAILABLE
(GROW 30)

VALUE:
143777Q

EXPRESSION TO EVALUATE:
(PLNR)

THVAL:
? THIS IS A METHOD FOR KNOWING THE STATE OF THE DATA-BASE
? THERE ARE NO INFORMATIONS IN THE DATA BASE
(THSTATE)

VALUE:
DONE

THVAL:
(MEMORY)
```

```
VALUE:
10063

THVAL:
(PUT 'FATHER 'THEOREM '(THCONSE (X Y Z)
((THV X) FATHER-OF (THV Y))
(THOR (THGOAL ((THV Y)SON-OF (THV X)(THV Z)))
    (THGOAL ((THV Y) DAUGHTER-OF (THV X)(THV Z))))))  ))))

VALUE:
FATHER

THVAL:
(MEMORY)

VALUE:
10175

THVAL:
? THE THEOREM IS MEMORIZED IN THE P-LIST OF THE ATOM FATHER
(GET 'FATHER 'THEOREM)

VALUE:
(THCONSE (X Y Z) ((THV X) FATHER-OF (THV Y)) (THOR (THGOAL
((THV Y) SON-OF ((THV Y) DAUGHTER-OF (THV X) (THV Z)))))

THVAL:
? THE THEOREM IS NOT ASSERTED.  THERE IS NO INFORMATION IN THE
? P-LIST OF THE ATOM, WHOSE NAME IS IN THE PATTERN OF THE THEOREM
(GET 'FATHER-OF 'THCONSE)

VALUE:
NIL

THVAL:
(THASSERT FATHER)

VALUE:
FATHER

THVAL:
(MEMORY)

VALUE:
10328

THVAL:
? WHEN THE THEOREM IS ASSERTED THERE IS AN INFORMATION STORED
? IN THE P-LIST OF THE ATOM FATHER-OF UNDER THE INDICATOR VALUE
(GET 'FATHER-OF 'THCONSE)

VALUE:
(NIL (2 (3 1 FATHER)))
```

```
THVAL:
? FOR THE VARIABLES THE SYSTEM EMPLOYES AN UNIQUE ATOM, WHOSE
? NAME IS THVRB
(GET 'THVRB 'THCONSE)

VALUE:
(NIL (3 (3 1 FATHER)) (1 (3 1 FATHER)))

THVAL:
? THIS IS A METHOD FOR KNOWING THE STATE OF THE DATA-BASE OF
? THEOREMS THCONSE
(THSTATE $TC)
(FATHER)

VALUE:

DONE

THVAL:
(TIME)

VALUE:
949

THVAL:
(PUT 'MOTHER 'THEOREM '(THCONSE (X Y Z)
((THV X) MOTHER-OF (THV Y))
(THOR (THGOAL ((THV Y) SON-OF (THV Z)(THV X)))
      (THGOAL ((THV Y) DAUGHTER-OF (THV Z)(THV X))))))  ))))

VALUE:
MOTHER

THVAL:
(TIME)

VALUE:
966

THVAL:
(THASSERT MOTHER)

VALUE:
MOTHER

THVAL:
(TIME)

VALUE:
976

THVAL:
(GET 'THVRB 'THCONSE)
```

```
VALUE:
(NIL (3 (3 2 MOTHER FATHER)) (1 (3 2 MOTHER FATHER)))

THVAL:
(MEMORY)

VALUE:
10663

THVAL:
(TIME)

VALUE:
989
```

... there are the definitions of the other theorems

```
THVAL:
(MEMORY)

VALUE:
13620

THVAL:
(TIME)

VALUE:
1435

THVAL:
? THIS IS A METHOD FOR KNOWING THE STATE OF THE DATA-BASE OF
? THEOREMS THCONSE
(THSTATE $TC)
(AUNT)
(UNCLE)
(FATHER-IN-LAW)
(SON-IN-LAW)
(DAUGHTER-IN-LAW)
(SISTER-IN-LAW)
(BROTHER-IN-LAW)
(GRANDSON)
(GRANDDAUGHTER)
(NIECE)
(NEPHEW)
(COUSIN)
(GRANDFATHER)
(GRANDMOTHER)
(BROTHER)
(SISTER)
(MOTHER)
(FATHER)
(PARENTS)
```

```
VALUE:
DONE

THVAL:
? THIS IS A METHOD FOR KNOWING THE STATE OF THE DATA-BASE OF
? THEOREMS THCONSE (THSTATE $TA)
(DCHECK)
(SCHECK)
(WIFE)
(HUSBAND)

VALUE:
DONE

THVAL:
(MEMORY)

VALUE:
14860

THVAL:
(TIME)

VALUE:
1796

THVAL:
($A (SCOTT SON-OF IRA CAROL )$T)

VALUE:
((SCOTT SON-OF IRA CAROL))

THVAL:
(MEMORY)

VALUE:
15251

THVAL:
(TIME)

VALUE:
1844

THVAL:
(GET 'SON-OF 'THASSERTION)

VALUE:
(NIL (2 (4 1 ((SCOTT SON-OF IRA CAROL)))))

THVAL:
? THIS IS A METHOD FOR KNOWING THE STATE OF THE DATA-BASE OF
? ASSERTIONS
(THSTATE THASSERTION)
((SCOTT SON-OF IRA CAROL))
```

```
VALUE:
DONE

THVAL:
(MEMORY)

VALUE:
15341

THVAL:
(TIME)

VALUE:
1906

THVAL:
(GET 'SON-OF 'THASSERTION)

VALUE:
(NIL (2 (4 1 ((SCOTT SON-OF IRA CAROL)))))

THVAL:
(ßA (IRA SON-OF          ALAN JANE ) ßT)

VALUE:
((IRA SON-OF ALAN JANE))

THVAL:
(ßA (SYLVIA DAUGHTER-OF ALAN JANE) ßT)

VALUE:
((SYLVIA DAUGHTER-OF ALAN JANE))

THVAL:
(ßA (JAMES SON-OF STEVEN MARY ) ßT)

VALUE:
((JAMES SON-OF STEVEN MARY))

THVAL:
(ßA (ANNE DAUGHTER-OF DONALD SUSAN) ßT)

VALUE:
((ANNE DAUGHTER-OF DONALD SUSAN))

THVAL:
(ßA (IRENE DAUGHTER-OF DONALD SUSAN) ßT)

VALUE:
((IRENE DAUGHTER-OF DONALD SUSAN))

THVAL:
(ßA (JACK SON-OF AARON MARGY) ßT)
```

```
VALUE:
((JACK SON-OF AARON MARGY))

THVAL:
(#A (ALAN SON-OF PATRICK MAY) #T)

VALUE:
((ALAN SON-OF PATRICK MAY))

THVAL:
(#A (JANE DAUGHTER-OF GERALD KATE) #T)

VALUE:
((JANE DAUGHTER-OF GERALD KATE))

THVAL:
(#A (DAVID SON-OF CARL SYLVIA) #T)

VALUE:
((DAVID SON-OF CARL SYLVIA))

THVAL:
(#A (SAUL SON-OF CARL SYLVIA) #T)

VALUE:
((SAUL SON-OF CARL SYLVIA))

THVAL:
(#A (JOHN SON-OF CARL SYLVIA) #T)

VALUE:
((JOHN SON-OF CARL SYLVIA))

THVAL:
(#A (CAROL DAUGHTER-OF JACK IRENE) #T)

VALUE:
((CAROL DAUGHTER-OF JACK IRENE))

THVAL:
(#A (JANET DAUGHTER-OF JACK IRENE) #T)

VALUE:
((JANET DAUGHTER-OF JACK IRENE))

THVAL:
(#A (ROBERT SON-OF JACK IRENE) #T)

VALUE:
((ROBERT SON-OF JACK IRENE))

THVAL:
(THPUTPROP 'JANET 'JAY 'WIFE)
```

```
VALUE:
JANET

THVAL:
(THPUTPROP 'JAY 'JANET 'HUSBAND)

VALUE:
JAY

THVAL:
? IN THIS WAY IT IS POSSIBLE TO HAVE A TRACE OF THE THEOREMS,
? ASSERTIONS ERASURES AND BREAKPOINTS
(THTRACE THEOREM THGOAL THASSERT THERASE THBKPT)

VALUE:
T

THVAL:
(ȿA (CARL SON-OF J1M MARY) ȿT)
>ASSERTING A1: (CARL SON-OF JIM MARY)
  >THANTE SCHECK:  (CARL SON-OF JIM MARY)
    >GOAL G2: (CARL SON-OF (THV F) (THV M))
    <G2 SUCCEEDED:  ((CARL SON-OF JIM MARY))
    <G2 FAILED
  <SCHECK FAILED
  >THANTE HUSBAND:  (CARL SON-OF JIM MARY)
  <HUSBAND SUCCEEDED:  THNUVAL
<A1 SUCCEEDED

VALUE:
((CARL SON-OF JIM MARY))

THVAL:
(ȿA(CARL SON-OF JAMES ANNE )ȿT)
>ASSERTING A3:  (CARL SON-OF JAMES ANNE)
  >THANTE SCHECK:  (CARL SON-OF JAMES ANNE)
    >GOAL G4:  (CARL SON-OF (THV F) (THV M))
    <G4 SUCCEEDED:  ((CARL SON-OF JAMES ANNE))
    <G4 SUCCEEDED:  ((CARL SON-OF JIM MARY))
    >ERASING E5:  (CARL SON-OF JIM MARY)
    <E5 SUCCEEDED
    >BKPT B6:  THE ASSERTION CARL SON-OF JIM MARY HAS BEEN ERASED
  <SCHECK SUCCEEDED:  THNOVAL
  >THANTE HUSBAND:  (CARL SON-OF JAMES ANNE)
  <HUSBAND SUCCEEDED:  THNOVAL
<A3 SUCCEEDED

VALUE:
((CARL SON-OF JAMES ANNE))

THVAL:
(THSTATE)
((DAVID SON-OF CARL SYLVIA))
((JOHN SON-OF CARL SYLVIA))
```

```
((SAUL SON-OF CARL SYLVIA))
((JAMES SON-OF STEVEN MARY))
((CAROL DAUGHTER-OF JACK IRENE))
((JANET DAUGHTER-OF JACK IRENE))
((IRENE DAUGHTER-OF DONALD SUSAN))
(MOTHER-IN-LAW)
(AUNT)
(UNCLE)
(FATHER-IN-LAW)
(SON-IN-LAW)
(DAUGHTER-IN-LAW)
(SISTER-IN-LAW)
(BROTHER-IN-LAW)
(GRANDSON)
(GRANDDAUGHTER)
(NIECE)
(NEPHEW)
(COUSIN)
(GRANDFATHER)
(GRANDMOTHER)
(BROTHER)
(SISTER)
(MOTHER)
(FATHER)
(PARENTS)
(DCHECK)
(SCHECK)
(WIFE)
(HUSBAND)
((SCOTT SON-OF IRA CAROL))
((IRA SON-OF ALAN JANE))
((ROBERT SON-OF JACK IRENE))
((JACK SON-OF AARON MARGY))
((ALAN SON-OF PATRICK MAY))
((JANE DAUGHTER-OF GERALD KATE))
((CARL SON-OF JAMES ANNE))
((ANNE DAUGHTER-OF DONALD SUSAN))
((SYLVIA DAUGHTER-OF ALAN JANE))

VALUE:
DONE

THVAL:
(MEMORY)

VALUE:

21202

THVAL:
(TIME)

VALUE:
2834
```

```
THVAL:
(THPROG (V Z) (THGOAL ((THV V)(THV Z) PARENTS-OF JACK) $T)
(THRETURN (LIST (THV V)(THV Z))))))))))))))
>GOAL G7:  ((THV V) (THV Z) PARENTS-OF JACK)
  >THCONSE PARENTS:  ((THV V) (THV Z) PARENTS-OF JACK)
    >GOAL G8:  (JACK SON-OF (THV X) (THV Y))
    <G8 SUCCEEDED:  ((JACK SON-OF AARON MARGY))
  <PARENTS SUCCEEDED:  THNOVAL
<G7 SUCCEEDED:  ((AARON MARGY PARENTS-OF JACK))

VALUE:
(AARON MARGY)

THVAL:
(MEMORY)

VALUE:
21602

THVAL:
(TIME)

VALUE:
2888

THVAL:
(THUNIRACE THEOREM THGOAL THASSERT THERASE THBKPT)
(THBKPT T NIL)
(THERASE T NIL)
(THASSERT T NIL)
(THGOAL T NIL)
(THEOREM T NIL)

VALUE:
T

THVAL:
(THFIND ALL (THV X) (X) ($G((THV X)GRANDFATHER-OF ROBERT) $T))

VALUE:
(DONALD AARON)

THVAL:
(MEMORY)

VALUE:
22125

THVAL:
(TIME)

VALUE:
2970
```

```
THVAL:
(⊄G (IRA BROTHER-IN-LAW CARL) ⊄T)

VALUE:
((IRA BROTHER-IN-LAW CARL))

THVAL:
(⊄G (JOHN COUSIN-OF SCOTT) ⊄T)

VALUE:
((JOHN COUSIN-OF SCOTT))

THVAL:
(THPROG (V) (⊄G((THV V)MOTHER-IN-LAW SYLVIA) ⊄T) (THRETURN (THV
V)))))

VALUE:
ANNE

THVAL:
(THFIND ALL (THV X) (X) (⊄G((THV X)BROTHER-OF DAVID) ⊄T))

VALUE:
(SAUL JOHN)

THVAL:
(THFIND ALL (THV X) (X) (⊄G((THV X) NEPHEW-OF DONALD) ⊄T))

VALUE:
NIL

THVAL:
(⊄G (JAMES UNCLE-OF CAROL) ⊄T)

VALUE:
((JAMES UNCLE-OF CAROL))

THVAL:
(⊄G (SYLVIA NIECE-OF IRENE) ⊄T)

VALUE:
NIL

THVAL:
(⊄G (CARL NEPHEW-OF IRENE) ⊄T)

VALUE:
((CARL NEPHEW-OF IRENE))

THVAL:
(MEMORY)

VALUE:
28926
```

```
THVAL:
(TIME)

VALUE:
3876

THVAL:

:STOP
END LIST.  TIME:  3879  MSEC.
```

An Experimental Environment for the Implementation of Question-Answering Systems

Georg Nees

1. Introduction

Question-answering systems (QASs) can be defined as advanced information retrieval systems which are distinguished by the following special features:

a) Input to and output from the QAS is formulated in subsets of a natural language.

b) A structured data base conserves the knowledge, which has been fed to the QAS in the form of explicit input sentences or which has been gathered by the QAS by help of its inductive and deductive capabilities.

c) The QAS answers queries by searching its data base for a direct answer, or by considering the query sentence as a theorem which has to be proved using the data base as an axiom system. Thus it follows that the QAS must possess searching and/or theorem-proving apparatus.

Information exchange with the QAS happens by way of dialogs. It is feasible, therefore, to consider QASs as special artificial-intelligent dialog systems. This paper describes a general technique for the construction of intelligent dialog systems which is based on the following assumptions and demands:

a) Very often there is a thesaurus of algorithms already available that are qualified to serve as building stones of a dialog-system-kit, e.g., a theorem prover might exist that is strong enough to deliever the necessary inferential power for the dialog or QA system. The task is then to assemble the parts of the kit and to frame them by a dialog-organizing routine. It is desirable that at least sometimes noncomputer scientists should be able to perform this task. Hence, the implementation method must be comfortable, and the implementation language should be easy to learn.

b) In most cases, noncomputer scientists are the people who order the needed dialog system. Experience proves that it is necessary that the orderer share in the responsibility for the features of the system in order to minimize later complaints. Very often the dialog behavior is the

feature of the system which is decisive for its practical usability. Therefore, the source code has to be explained to the orderer on a macroscopic level. If this is possible, the source code can be used as the obligatory text for documentation.

c) The implementation language will have modes of expression rich enough to enable the analysis of complicated input texts.

Item (a) can be fulfilled by making the implementation language as simple as, e.g., the language BASIC. Item (b) leads to the use of the description of dialogs by state diagrams [3]. States are mapped onto labels in a DIABAS program. A simple situation M (i.e., a state or vertex M) in a dialog will be taken as an example, where a set of keywords controls the branching of the program.

For example, the input 'Bonn' has a jump to label Bonn as its consequence, etc. Figure 1 shows the graphical representation as well as the translation of this situation to DIABAS by means of a switch statement.

Finally, item (c) is fulfilled by DIABAS by the structure of any DIABAS program, which is a system of procedures where one procedure may call another one quite arbitrarily, e.g., recursively.

The DIABAS compiler translates the source code to an intermediate language called Dzs, which is evaluated by interpretation. The Dzs procedures are reentrant and the procedure body is strictly separated from its corresponding data block. This allows the construction of a control mechanism that does not follow a stack discipline. Section 5 deals with a mechanism of this kind.

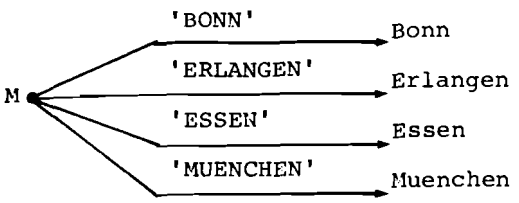
2. Syntax and Semantics of Dialog Systems

DIABAS procedures analyze the input line in a dialog and start processes accordingly. The set of algorithms, which are controlled in this way, together with data structures interacting with them, is called the semantical base. There is a unique interface between the DIABAS procedure and the semantical base that consists of an integer array of suitable length. The front segment of this array contains the input text. Figure 2 shows the structure of the DIAGIP system in a way which is independent of any special implementations. The system is portable, especially when coded, e.g., in FORTRAN IV.

The semantical base is called by statements of the format

(1) $\quad$ Basis($p_1,\ldots,p_n$) ,

where n may be $\emptyset$. A consequence of a call (1) is the storing of $p_1,\ldots,p_n$ in the upper half of the interface. In a FORTRAN




```
procedure Stadt(x); &x;

...

M:

switch x = ('BONN')/Bonn,-
           ('ERLANGEN')/Erlangen,-
           ('ESSEN')/Essen,-
           ('MUENCHEN')/Muenchen; go to N;

...

Bonn:  Basis(1); ...

...

Erlangen:  Basis(2); ...

...

Essen:  Basis(3); ...

...

Muenchen:  Basis(4); ...

...

N:  ...

.
```

Figure 1.

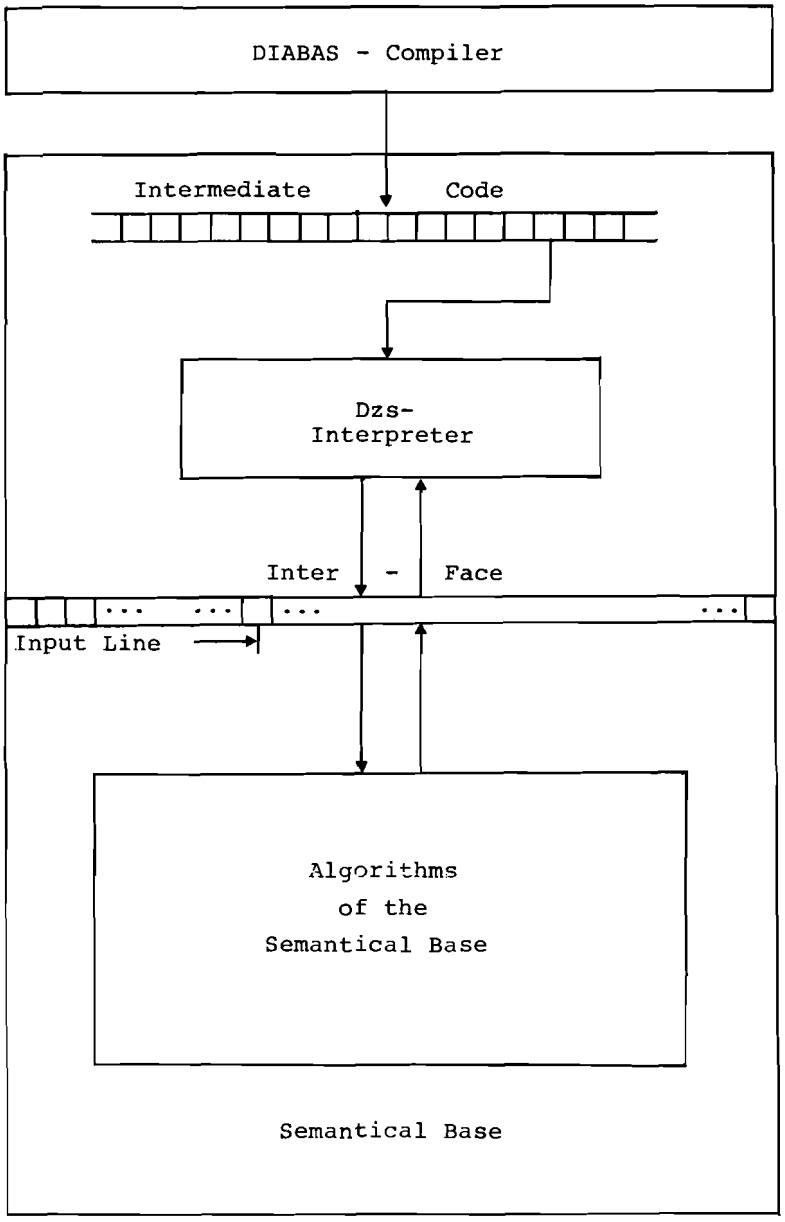

DIABAS - Compiler
Intermediate
Code
Dzs-
Interpreter
Inter
-
Face
Input Line
Algorithms
of the
Semantical Base
Semantical Base

Figure 2.

environment, a DIABAS application might have the structure shown in Figure 3, where the common-area Regist is identical with the interface.

3. The Language DIABAS

The left column of Table 1 shows the primitive DIABAS statements. The right column contains the translation to Dzs.

Every DIABAS procedure begins with a line

(1) procedure P(q_1,...,q_n) ;

where P is the procedure name. The parameter list in (1) may be empty. Names of parameters and other variables may consist of as many significant characters as one likes, a feature which is very important for the easy understandability of the program text. Every procedure is finished by the full-stop character. The procedure head (1) is followed by a sequence of declarations of the local variables

(2) &V_1,...,V_m ;

where V_i is of the structure V or V(n), where n is nonnegative. For example, in the case of the declarations

(3) &A(2),B,C(Ø),D(3);

the corresponding data block will contain the registers

(4) A(Ø),A(1),A(2),B(Ø),C(Ø),
D(Ø),D(1),D(2),D(3) .

Hence it follows that the array, determined by an indexed variable, begins with place Ø. At the same time, every variable is an array so that, e.g.,

(5) B(3) = ('ABC') ,

is meaningful and has as its consequence the storing of the address of the string 'ABC' in place D(1). Every variable may be indexed by any other variable or by an integer constant. If, for example, (A has the value 3, then A(A) addresses B (see (3) and (4)).

Besides local declared variables, every procedure has access to global variables, denoted by $j, where j is an integer. The value of $j is the content of register j of the interface to the semantical base (see Figure 2).

The DIABAS statements in Table 1 are almost self-explanatory. If declaration (3) is valid, then, e.g.,

(6) A = D(Ø..2) ,

is equivalent to,

```
(2)             program Diagip
                common Regist
                integer*2 Regist(1999)
                ...
                call Interp
                ...
                end


                subroutine Interp
          C This is the Dzs-Interpreter
                ...
                call Basis(p1,...,pn)
                ...
                end


                subroutine Basis
                common Regist
                integer*2 Regist(1999)
          C Conditioned calls
          C of procedures P1,...,Pk
          C follow
                ...
                Call P1
                ...
                Call Pk
                ...
                end
```

Figure 3.

Table 1. Primitive DIABAS statements and their translations to the intermediate language Dzs.

DIABAS	Dzs
1. Counters	
+ n,x	1Ø,n,x
- n,x	11,n,x
2. Assignments	
z = x	1,z,x
z(a) = x	2,z,a,x
x = z(b)	3,z,b,x
x(a) = z(b)	4,z,a,x,b
x = z(b..c)	5,z,b,c,x
x(a) = z(b..c)	6,z,a,x,b,c
z = (AfA) where A is string-quote, f string	7,z,h where h is the string f
z(a) = (AfA)	8,z,a,h
3. Unconditioned jumps	
GOTO g	4Ø,g
GOSUB q(p_1,...,p_k) where k = Ø is allowed	6Ø,k,q,p_1,...,p_k
BASIS(p_1,...,p_k) where k = Ø is allowed	2ØØ,k,p_1,...,p_k
RETURN	7Ø
4. Conditioned jumps	
(x.W.y)GOTO g	4w,g,x,y
(x(a).W.y)GOTO g	4w1,g,x,y,a
(x.W.y(b))GOTO g	4w2,g,x,y,b
(x(a).W.y(b))GOTO g	4w3,g,x,y,a,b
(x.W.y(b..c))GOTO g	4w4,g,x,y,b,c
(x(a).W.y(b..c))GOTO g	4w5,g,x,y,a,b,c
(x.W(AfA))GOTO g	4w6,g,x,h
(x(a).W.(AfA))GOTO g	4w7,g,x,a,h

Table 1 (continued).

DIABAS	Dzs
5. Input statements	
GET	94
6. Output statements	
PUT x(a..b)	96,x,a,b
Out h	95,h
*AfA	95,h
7. Stop statement	
STOP	7ØØ

(7)
```
A(Ø) = D(Ø)
A(1) = D(1)
A(2) = D(2)   .
```

GOSUB calls another procedure by call by name. GET gets the next input line. PUT prints the text which is stored character by character in the places a to b. OUT(h) prints the string which is addressed by h.

Figure 4 shows a DIABAS program that checks the correctness of a bracketing, the semantical base being empty in this case. A general feature of DIABAS is the equivalence of $ to $1; this is used in the third line of Figure 4. The highest procedure in a hierarchy of DIABAS procedures is called by the interpreter itself. In the program of Figure 4, the codings 44, 41, 4Ø are used for the characters '(', ')', '.'. The DIABAS system is using its own character code, which results from the enumeration of the sequence of characters,

(8)
```
Ø123456789
ABCDEFGHIJKLMNOPQRSTUVWXYZ
+-/=.)*,('&!"#$%:;<> ?@[]^
```

beginning at Ø.

Table 2 contains the remaining two nonprimitive DIABAS statements which are translated into sequences of Dzs instructions (see section 4.1). A nonprimitive DIABAS statement is used in Figure 4, a switch in Figure 1.

Table 2. Nonprimitive DIABAS statements.

1. Nonprimitive conditioned jumps (---.W.---)s where s is nonconditioned
2. Switch SWITCH x = $e_1/g_1,\ldots,e_k/g_k$

4. The Intermediate Language Dzs and Its Interpretation

4.1 The Intermediate Language

The DIABAS compiler takes DIABAS procedures as input and generates Dzs procedures as output. Table 1 shows the primitive statements and their translation into Dzs instructions. Each primitive statement is mapped onto one Dzs instruction. Nonprimitive statements are mapped onto sequences of Dzs instructions.

```
*PROCEDURE RAHMEN; &S
*$ØØ =-1; GET; GOSUB KG;
*S = $($ØØ); (S.EQ.4Ø) GOTO KORREKT;
**('ZU VIELE RECHTE KLAMMERN'); STOP;
*KORREKT:  *('KLAMMERGEBIRGE KORREKT'); STOP.


*PROCEDURE KG; &S
*A:  + 1,$ØØ; ($ØØ.GT.72) GOTO ERROR;
*S = $($ØØ); (S.EQ.44) GOTO DOWN;
*(S.EQ.41) GOTO UP; (S.EQ.4Ø) RETURN; GOTO A;
*DOWN:  *('DOWN'); GOSUB KG; GOTO A;
*UP:  *('UP'); RETURN;
*ERROR:  *('ZU WENIG RECHTE KLAMMERN'); STOP.


*(().
 DOWN
 DOWN
 UP
 ZU WENIG RECHTE KLAMMERN


*(X()))Y.
 DOWN
 DOWN
 UP
 UP
 UP
 ZU VIELE RECHTE KLAMMERN


*().
 DOWN
 UP
 KLAMMERGEBIRGE KORREKT
```

Figure 4.

Branching operations of the format

(1) (---.W.---)s

are nonprimitive. In (1), s must be neither a conditioned jump nor a switch; W is one of the six conditions EQ, NE, LE, GE, LT, GT that are mapped one by one onto the integers w = 1 to w = 6 (see Table 1).

For any W, let Q be the negation of W. Statements (1) are mapped onto the Dzs equivalent of the sequence

(2) (---.Q.---)GOTO g; s; g:

A very comfortable nonprimitive statement is the switch

(3) SWITCH x = e_1/g_1, e_2/g_2,...,e_k/g_k

which is mapped onto

(4) (s.NE.e_1)GOTO l_1; GOTO g_1;

l_1:(x.NE.e_2)GOTO l_2; GOTO g_2;

...

l_k:(x.NE.e_k)GOTO g; GOTO g_k; g:

The translations (2) and (4) of (1) and (3) are advantageous because their consequence is a shorter running-time of the Dzs program if some branching coadditions of the corresponding source statement are not fulfilled. This can, for example, be seen in the case of the switch

(5) (x.EQ.('HAMBURG')) y = 1 .

The statement which follows (5) is reached at once if the string to be compared with does not begin with an 'H'.

Every Dzs instruction in Table 1 begins with an operator, which is followed by no or some operands. Each operator or operand is stored in exactly one machine word. The implementation for the SIEMENS 4004/151 computer is tuned to halfwords, i.e., 16-bit-bytes. DIABAS is suited also for cross-compilations, especially to minicomputers which use a 16-bit-word. Certainly the target computer of the cross-compilation must be able to lodge a DIABAS interpreter, perhaps in the form of firmware.

4.2 The Structure of the Dzs-Procedure

The structure of any procedure in the Dzs-intermediate language, is the following:

(1) Word Ø: begin-token
1 to 3: procedure-name

```
                4:  address l of last word
                5:  address of first instruction
                6:  address c of the pool of constants
                7:  length s of the data block
                8:  address of last parameter (in the data
                    block, beginning with Ø)
          9 to 12:  reserved for later use
        13 to c-1:  instructions
         c to l-1:  constants
                l:  end token
```

The storage block (1), consisting of l + 1 words, is called procedure block. The procedure block consists of the procedure head (words Ø to 12), the instruction block (words 13 to c-1), the constant block (words c to l-1(and the end token. Any use of the term procedure, as far as we deal with the intermediate language Dzs, is to be prefixed by "intermediate-" or "Dzs-" by definition.

During the time in which the Dzs interpreter has access to a procedure, the content of the procedure block is never changed. All variables or addresses of variables, which are used by the procedure, are stored in a data block that has a length of s words. When a procedure has been called and is running, there is an incarnation of the procedure defined, which can be considered to be the ordered pair of the procedure block and a data block. If coroutines or collateral processes are organized by the DIABAS mechanism, there may exist many incarnations for some procedures at interpreting time.

As an example for a DIABAS-procedure consider

```
(2)     *PROCEDURE P(A); &B(1);
        *(A.EQ.('DM')) GOTO M; B = 1322; A = B;
        *RETURN; M:  $2 = 9999; RETURN  .
```

The compiled version (augmented by some comment in DIABAS) of (2) is

```
(3)     Ø.. -32767 |
        1..   2599 |
        2..   9999 |
        3..   9999 |
        4..     3Ø |
        5..     13 |
        6..     28 |
        7..      3 |
        8..      Ø |
        9..      2 |
       1Ø..      Ø |
       11..      Ø |
       12..      Ø |
       13..  32416 | (A.EQ.('DM')) GOTO M
       14..     24 |
       15..      Ø |
```

```
16..    -28
17..  32ØØ1   | B = 1322
18..      1
19..    -28
20..  32ØØ1   | A = B
21..      Ø
22..      1
23..  32Ø7Ø   | RETURN
24..  32ØØ1   | M:  $2 = 9999
25.. -3ØØØ2
26..    -29
27..  32Ø7Ø   | RETURN
28..   1322   | CODE
29..   9999   | OF 'DM'
3Ø.. -32768
```

Any data block used by (3) consists of s = 3 words:

(4) Ø: A
1: B(Ø)
2: B(1) .

The address Ø of the single parameter A in the data block (4) is equal to the address of the last parameter in this special case, which address can be found in word 8 of (3). The information of whether a variable is a parameter or not is used by the interpreter for its organizing the procedure calls by name.

It should be mentioned that the constant block, which begins at word 28 of (3), is used multiply. The two words which code the string 'DM' are at the same time constants addressed by words 19 and 26. In Dzs, constants are discriminated by a minus-sign prefix. Word 25 contains the address of the global variable $2. If $n is global, then -(n + 3ØØØØ) is the Dzs address of $n.

4.3 The Compiler and the Interpreter

A closer inspection of Table 1 will prove that the implementation of a compiler from DIABAS to Dzs can be very easily done. The 4004-DIABAS-compiler is written in FORTRAN IV.

In order to understand the working of the interpreter, one has to know some details of the way procedure blocks and data blocks are stored. Both types of data areas are contained in a one-dimensional array, which is called main stack. The main stack has the following structure:

(1) area of the global variables
area of the data blocks
area of the procedure blocks .

If the interpreter is implemented in a higher programming language, then one and the same symbolic index can address any of the three different types of data in the main stack: on the one hand global variables, parameters, and local variables in the data blocks, on the other hand constants in the procedure blocks. By way of the differentiation of addresses, which was explained in section 4.2, the interpreter can decide, for any address, in which part of the main stack the addressed piece of data is situated.

Before the interpreter starts, all procedure blocks are input to the main stack. The procedure name of each procedure is entered in the list of procedure names, together with the address of the procedure in the main stack. If a procedure call

(2) GOSUB $P(q_1,\ldots,q_n)$

occurs, at first the address of the procedure block, which belongs to P, is taken from the list of procedure names. In a second step, the interpreter finds in word number 7 of the procedure block the length of the data block wanted. Hence, the interpreter is able to claim space for the data block on the main stack. In a third step, the address of the data block of the calling procedure and the value of the instruction counter of the interpreter is stacked.

5. An Escape-Resume Mechanism

When communicating with the computer, it is often desirable to be able to escape one spot of operation and to go elsewhere, at a later time to be able, however, to resume and to continue the previous work at the old state of completion. Such a situation can be described in a slight extension of the hitherto version of DIABAS. If, for example, control went to a procedure q by a call

(1) GOSUB $q(p_1,\ldots,p_n)$,

it is possible to initiate inside q an escape by an instruction

(2) ESCAPE k .

In (2), k is the identification of the k-th escape-resume mechanism organized in this way since starting the program. Figure 5 shows the scheme of the control and data flow that has to be handled. The numbers in the small circles count the successive steps to be performed:

1 GOSUB $q(p_1\ldots,p_n)$

2 A data block for q is claimed on the stack of data blocks

3 The return address of the GOSUB is stacked

4 Control goes to procedure q

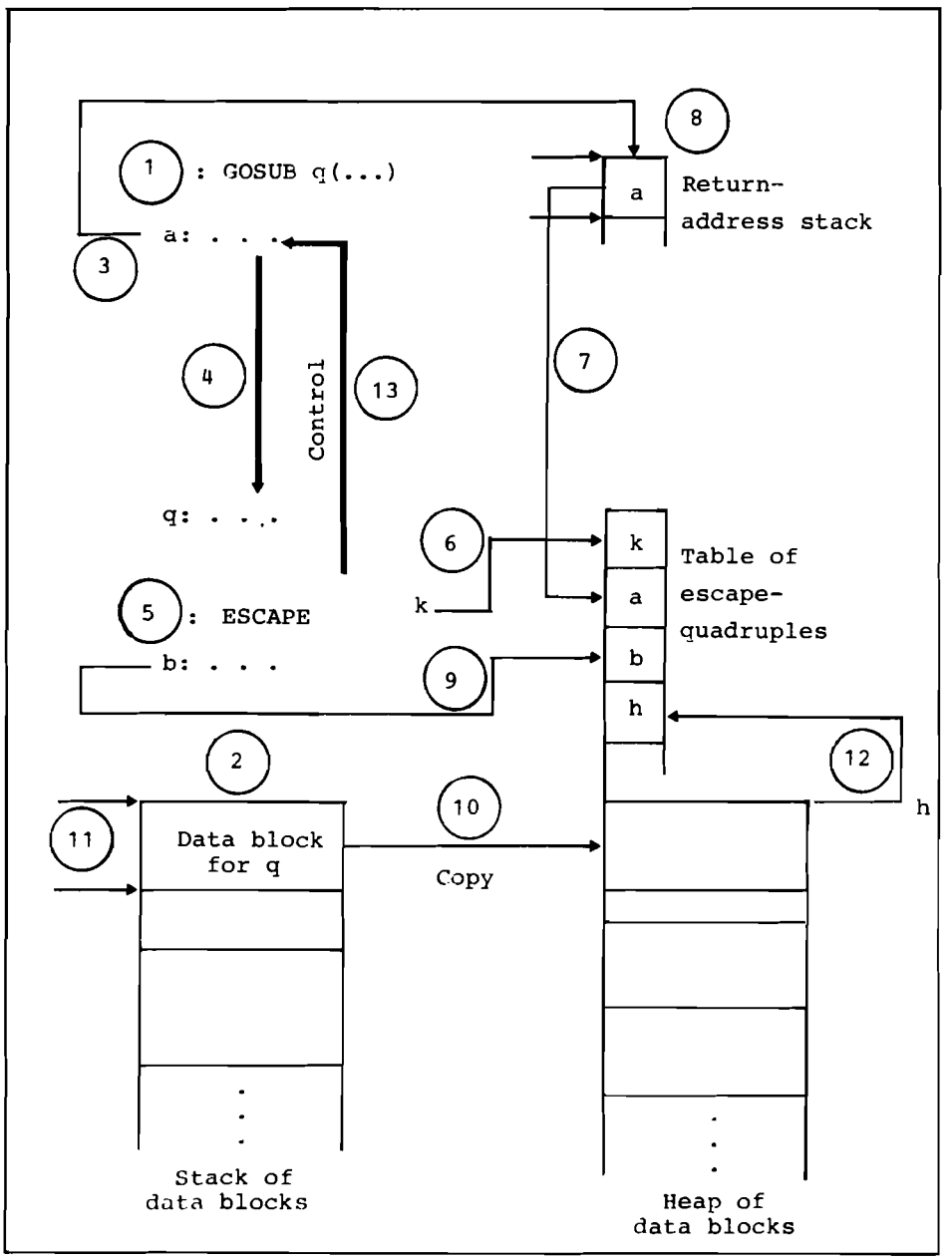


Figure 5. Escape mechanism identified by k.

5 ESCAPE k

6 A quadruple is claimed on a heap, and k is stored

7 The return address a is transferred to the quadruple

8 The return address stack is popped up

9 A resume address b is transferred to the quadruple

10 The data block of q is copied to a heap of data blocks

11 The stack of data blocks is popped up

12 The address of the new data block of q is transferred to the quadruple

13 Control goes back to the procedure which called q

In this, moment q is transformed from a subprocedure to a coprocedure.

If at a later stage control will resume at state b in the coprocedure, the operations shown in Figure 6 must take place:

1 RESUME k

2 By a search on the table of quadruples (which is a heap) for k, the resume address b is found

3 So is the data block of coprocedure q which has been retained

4 Control goes to the coprocedure.

References

[1] Conway, M.E. "Design of a Separable Transition-Diagram Compiler." Comm. ACM 6, 1963.

[2] Nees, G. "Beschreibung Kognitiver Systeme mit Hilfe des λ-Kalküls." In Lecture Notes in Economics and Mathematical Systems, Heidelberg: Springer, 1973

[3] Newman, W.M. "A System for Interactive Graphical Programming." Spring Joint Computer Conference, AFIPS, Montvale, N.J., 1968.

[4] Woods, W. "Augmented Transition Network Grammars for Natural Language Analysis." Comm. ACM 13, 1970.

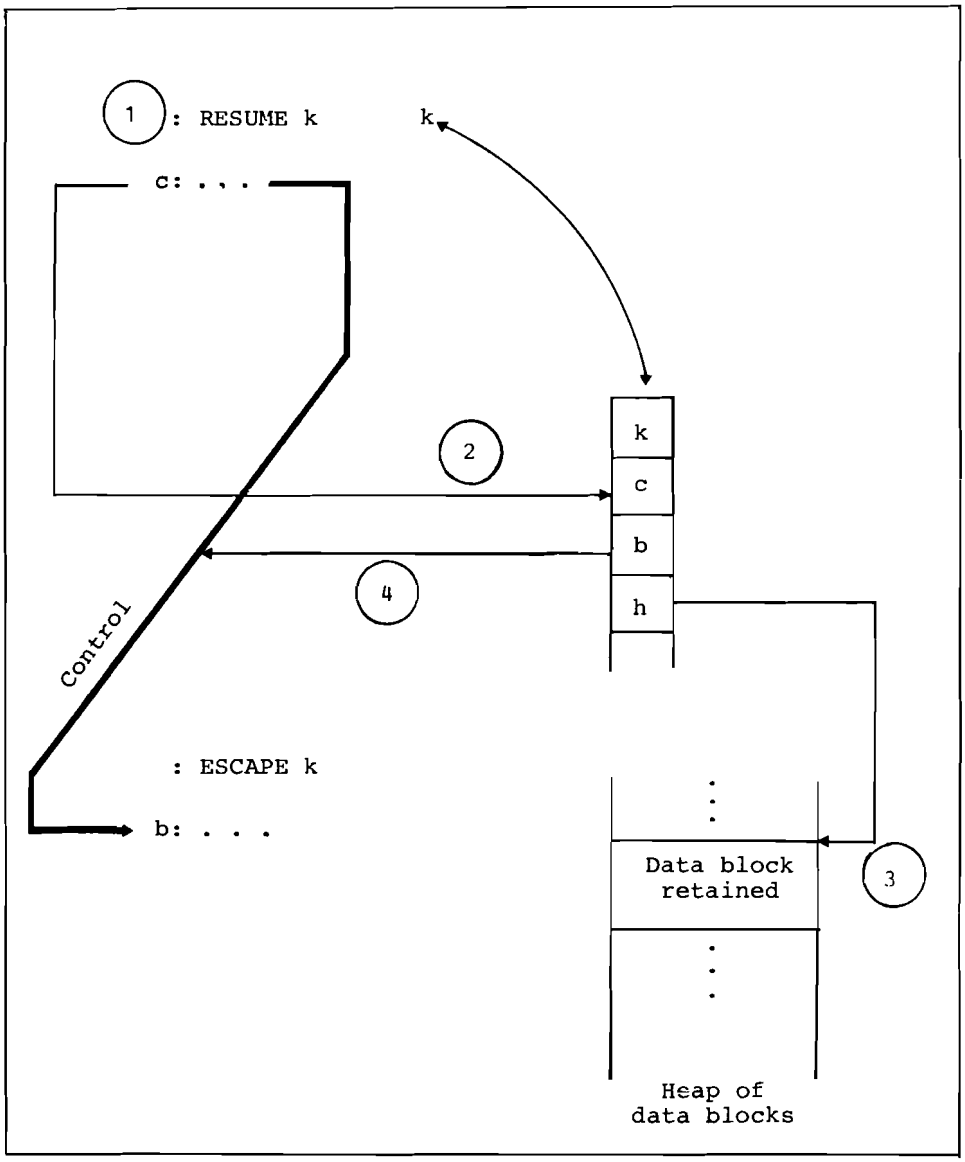


Figure 6. Resume mechanism identified by k.

PLATON - A New Programing Language for Natural Language Analysis

Makoto Nagao and Jun-Ichi Tsujii

1. Introduction

In this paper, we describe a new programing language which is designed to make the writing of natural language grammars easy. A simple structural analysis program using this language is given as an example. There are two key issues in analyzing natural language by computer: one is how to represent the knowledge (semantics, pragmatics) and the state (context) of the world. The other is to program technology appropriate to the syntax-sematics, syntax-context interface. The point in designing a programing language is to make such programing less painful.

Traditional systems, which represent grammars as a set of rewriting rules, usually have poor control mechanisms, and flexible interactions between syntax and other components cannot be expressed in them. On the other hand, the systems in which rules of grammars are embedded in procedures make it possible to intermix the syntactic and semantic analyses in an intimate way. However, these systems are apt to destroy the intelligibility and regularity of natural language grammars because in these systems both the rules and their control mechanisms are contained in the same programs.

Recently various systems for natural language analysis have been developed. T. Winograd's "PROGRAMMER" is a typical example of procedure-oriented systems. In this system, syntactic and other components are able to talk together closely in the course of analyzing sentences. However, details of the program are lost in the richness of this interaction. LINGOL, developed by V. Pratt at MIT, is a language appropriate to the syntax-semantics interface. With LINGOL, it is easy to write grammars in the form of rewriting rules. The TAUM group at Montreal University has evolved a programing language named System-Q, in which expressions of trees (strings and lists of them) can be matched against partial expressions (structural patterns) containing variables, and they can be transformed in an arbitrary way. The augmented transition network (ATN) proposed by W. Woods gives a good framework for natural language analysis systems. One of its most attractive features is its clear distinction between grammar rules and the control mechanism. We can evolve the model by adding various facilities to its control mechanism.

This model has the following merits:

a) It provides power of expression equivalent to transformational grammars.

b) It maintains much of the legibility of the context-free grammars.

c) Rules of a grammar can be easily changed. So we can improve them through a trial and error process while writing the grammar.

d) It is possible to impose various types of semantic and pragmatic conditions on the branches between states. By doing this, close interactions between syntax and other components can be easily accomplished.

It, however, has the following shortcomings, especially when we apply it to the parsing of Japanese sentences:

a) It scans words one by one from the leftmost end of an input sentence, checks the applicability of a rule, and makes the transition from one state to another. This method is well suited for English sentences. But because the order of words and phrases in Japanese sentences are relatively free, it is preferable to check the applicability of a rule by a flexible pattern-matching method.

b) Without a pattern-matching mechanism, a single rewriting rule of an ordinary grammar is to be often expressed by several numbers of rules belonging to different states in Woods' ATN-parser.

c) ATN-model essentially performs a kind of top-down analysis of sentences. Therefore, how it recovers failures of its predictions is one of the most difficult problems. Wood's ATN-parser seems to pay no attention to this problem.

Considering these factors, we developed PLATON (Programing Language for Tree Operation), which is based on the ATN-model and has various additional capabilities such as pattern matching, flexible backtracking, and so on. As in System-Q and LINGOL, PLATON's pattern-matching facility makes it easy to write a rewriting rule. Moreover, it extracts substructures from the inputs and invokes appropriate semantic and contextual checking functions which may be arbitrary LISP functions defined by a user, and arguments of which are the extracted substructures.

A backtracking mechanism is also necessary for language understanding as well as for other fields of artificial intelligence. During the analysis, various sorts of heuristic information should be utilized properly. At each stage, the information which may be driven from the syntactic, semantic, or contextual consideration may give an ureliable criterion,

but the result which fulfills all the criteria will be a correct one. The program may choose each time the most satisfactory rule from many candidates by certain criteria at hand. But in further processing, if the choice is found to be wrong by other criteria, the program must be able to backtrack to the point at which the relevant decision was made. In PLATON, we can easily set up arbitrary numbers of decision points in the program. And if the processing results in some failure, the control will come back to the points relevant to the cause of the failure.

2. Pattern-Matching

Before proceeding to the detailed description of the specifications of PLATON, it is necessary to explain the representation scheme of input sentences and parsed trees. The process of analyzing a sentence, roughly speaking, may be regarded as the process of transforming an ordered list of words to a tree structure, which shows explicitly the interrelationships of each word in the input sentence. During the process, trees which correspond to the parts already analyzed and lists which have not been processed yet may coexist together in a single structure. We, therefore, should be able to represent such a coexisting structure of trees and lists. A list structure means one in which the order of elements is not changeable. On the other hand, a tree structure consists of a single root node and several nodes which are tied to the root by distinguishable relations. Because such relations between the root and the other nodes are explicitly specified, the order of nodes in a tree, except the root which is supposed to be placed on the leftmost end in our expression, is changeable. Different matching schemes should be applied on trees and lists.

Now we show the formal definition of such a coexisting structure. The <structure> is the fundamental data structure into which all of the objects processed by PLATON must be transformed. Hereafter, we call it simply structure.

Formal definition of <structure> is as follows.

<structure> :: = <tree>|<list>

<list> :: = (*<structures>)

<structures> :: = |<structure> <structures>

<tree> :: = <node> | (<node> <branches>)

<branches> :: = <branch>|<branch> <branches>

<branch> :: = (<relation> <tree>)

<node> :: = <list>|ARBITRARY LISP-ATOM

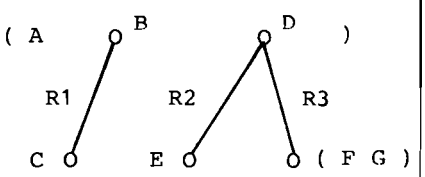
<relation> :: = ARBITRARY LISP-ATOM

A simple example following this definition is shown in Figure 1. Two lists which have the same elements but a different order of them--for example, (*A B C) and (* A C B)--

should be regarded as different structures. On the contrary, two tree structures such as (A (R1 B)(R2 C)) and (A (R2 C)(R1 B)) are regarded as being identical. Besides the usual rewriting rules which treat such strings, structural patterns which contain variable expressions are permitted in PLATON. PLATON-interpreter matches such structural patterns against the structure under processing, and checks whether the specified pattern is found in it. At the same time, the variables in the pattern are bound to the corresponding substructures.

Variables in patterns are indicated as :X (X is an arbitrary LISP atom), and the following parts can be expressed by variables in the above definition of <structure>:

a) arbitrary numbers of <structures>, that is to say, list elements in the definition of <list> (Figure 2, example 1). We can also specify the number of list elements by using the variables :X + number. For example, the variable :K2 will match with two elements in a list.



Coexisting Structure of Trees and Lists

```
(* A (B (R1 C)) (D (R2 E)
        (R3 (* F G))))
```

Corresponding Expression in PLATON

Figure 1. Expression of Structure in PLATON.

b) arbitrary numbers of <branches>, in the definition of <tree> Figure 2 example 2).

c) <tree> in the definition of <branch> Figure 2, example 3).

We shall call such structural patterns <structure-1>. By means of using the same variable several times in a pattern, we can

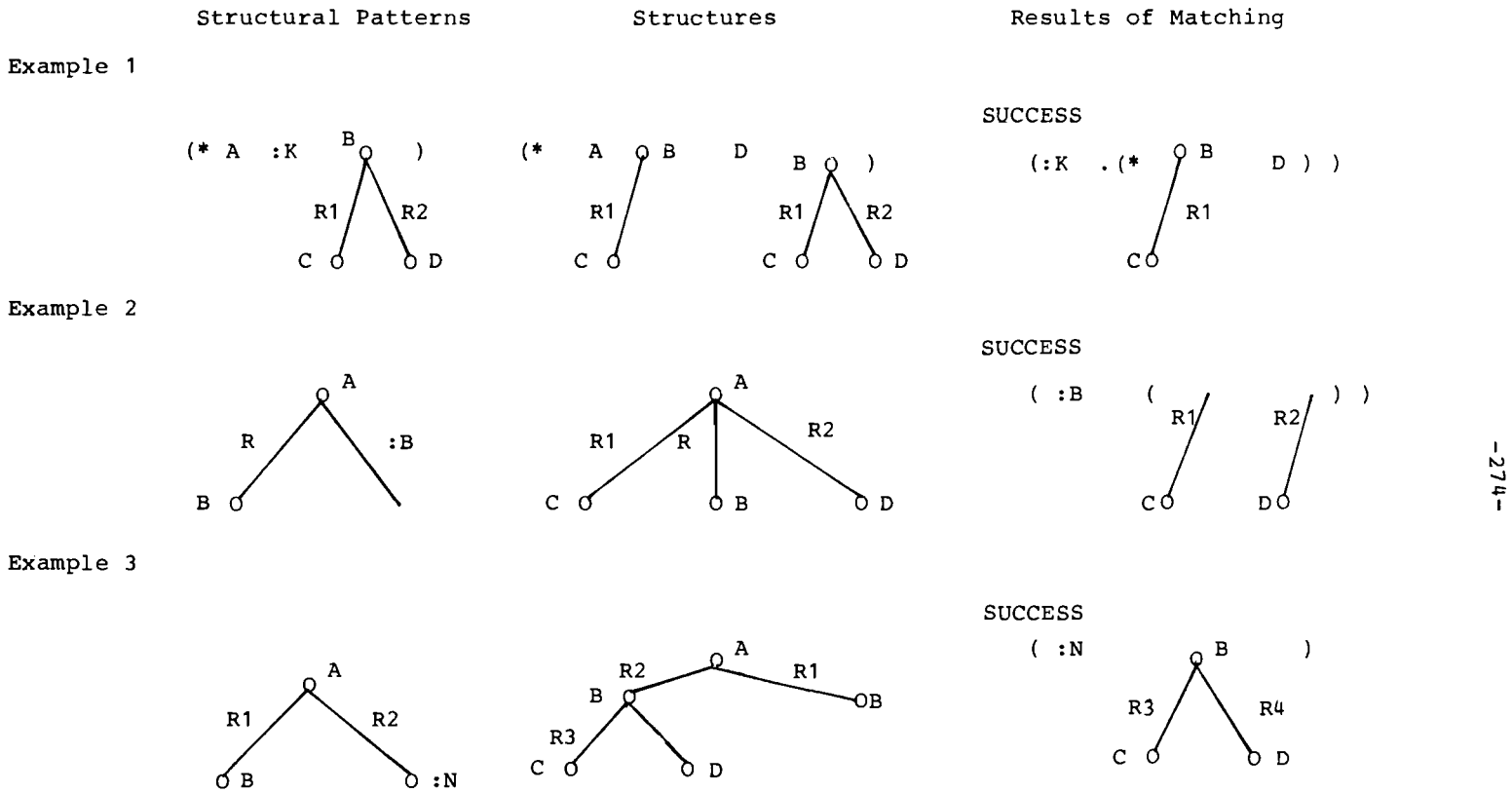


Figure 2. Illustration of Matching.

express a structure in which the same substructure appears in two or more different places. The character ! in a list indicates that the next element following the character is optional.

3. Basic Operations of PLATON

A grammar, generative or analytical, whichever it may be, is represented as a directed graph with labeled states and branches in which there is one distinguished state called the start state, and a distinguished set of states called final states. Each branch is a rewriting rule and has the following elements:

a) applicability conditions of the rule, typically represented as a structural pattern,

b) actions which must be executed, if the rule is applicable,

c) structural pattern into which the input structure should be transformed.

Each state has several numbers of branches ordered according to the preference of the rules. When the control jumps to a state, it checks the rules associated with the state one by one until it finds an applicable rule. If such a rule is found, the input structure is transformed into another structure specified by the rule, and the control makes the state transition.

In addition to the above basic mechanism, the system is provided with push-down and pop-up operations. In applying a rule, several substructures are extracted from the whole structure by variable binding mechanisms of pattern matching, and each is analyzed from a different state (push-down). After each is analyzed appropriately, the control will come back to the suspended rule and continue to execute it (pop-up). Using these operations, embedded structures can be easily handled (Figure 3).

Table 1 shows the formal definition of a grammar of PLATON. It shows that branches or rewriting rules in an ATN-parser correspond to six-lets, that is, <pcon>, <strx>, <con>, (<trans>), (<acts>), and <end>, <strx> of a rule corresponds to the left side of a rewriting rule and indicates a structural pattern on which the rule is applicable. By definition, <strx> is,

a) / or

b) structure-1

Definition a)/, shows that the rule is applicable whatever the structure under processing is. The variables used in <structure-1> are bound to corresponding substructures when the matching succeeds. The result of example 1 in Figure 2 indicates that the variable :K is bound to the substructure (*(B(R1 C))D).

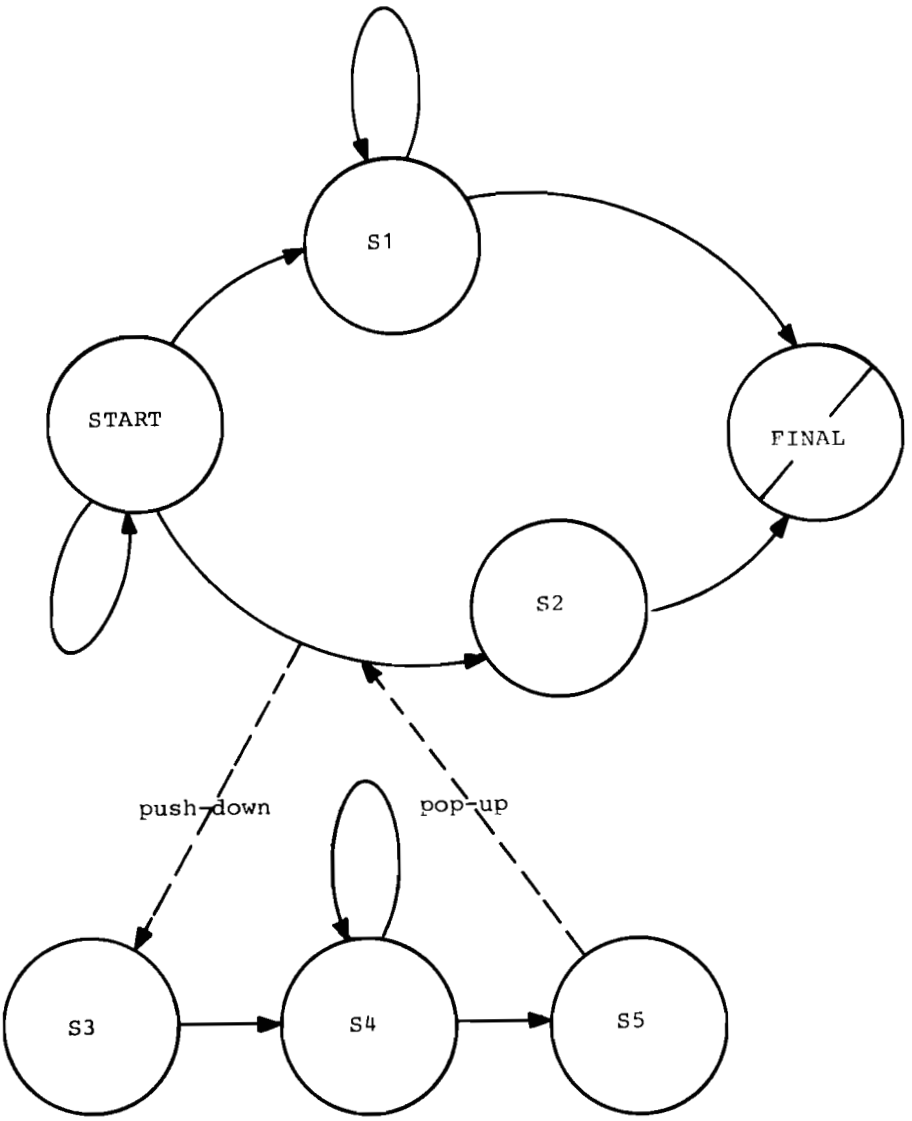


Figure 3. State diagram.

Table 1. Formal definition of grammar in PLATON.

```
<grammar> :: = (<states>)
<states> :: = <state>|<state><states>
<state> :: = (<state-name><rules>)
<rules> :: = |<rule><rules>
<rule> :: = (<pcon><strx><con>(<trans>) (<acts>) <end>)
<trans> :: = |<transit><trans>
<transit> :: = ((<state-name><structure-2> <register-name>/<variable-name> <errorps>
<errorps> :: = |<errorp><errorps>
<errorp> :: = (<failure-message><act><pros>)
<pros> :: = <pro>|<pro><pros>
<pro> :: = (EXC <trans>)|(TRANS (<state-name><stry>))
<end> :: = (NEXT <state-name><stry>)
     |(NEXTB <state-name><stry>)
     |(POP <stry>)|(FM <failure-message>)
<acts> :: = |<act><acts>
<act> :: = <form>|(SR <register-name><form>)
     |(SU <register-name><form>)
     |(SD <register-name><form>)
<strx> :: = <structure-1>↑/
<stry> :: = <structure-2>/
<pcon>,<con> :: <form>
<form> :: = (GR <register-name>)|(GV <variable-name>)
     |(TR <structure-2>)|(TR /)|ARBITRARY LISP FORM
<variable-name> :: = :X(X is an arbitrary LISP-atom.)
<register-name> :: = /X(X is an arbitrary LISP-atom.)
```

The scope of this binding is limited to the inside of the rule. The same variable name in different rules has different interpretations. In this sense, :X-type variables in <structure-1> are called local variables. On the other hand, we can store certain kinds of results of a rule in registers and refer them to different rules. These kinds of variables, which we call registers, are represented by symbols /X(X is an arbitrary LISP atom).

Besides the pattern matching, <pcon> and <con> can also check the applicability of a rule. Certain parts of the results of previous rules are contained in registers, not in structure. We can check the contents of these registers by using <pcon>-part functions such as GR, GU, and so on (these functions are listed in Table 2) and other LISP-functions which were defined by the usual LISP-function, DEFINE.

We can check semantic and contextual coordination between substructures by using appropriate functions in <con>-part of a rule. Semantic and contextual analyses cannot be expressed in the form of a simple rewriting rule. These analyses require different frameworks, such as lexical information of words which may represent the knowledge of the world, and a contextual one which may express the state of the world. We can use arbitrary LISP-forms in <con>-part, according to whichever semantic and contextual models we choose.
For example, suppose:

```
strx = ((ADJ(TOK :N))(N TOK :N1)) :I)

con = (SEM :N :N1).
```

Here TOK is a link of a word and its part of speech. :N and :N1 are the words of an input sentence. SEM is a function defined by the user which checks the semantic coordination between the adjective :N and the noun :N1. By this SEM function, we can search, if necessary, through both lexical entries and contextual data bases.

In this manner, it is possible that if a certain syntactic pattern is found in the input structure, an appropriate semantic function will be called. So the intimate interactions between syntactic and semantic components can be easily obtained without destroying the clarity of natural language grammars.

Arbitrary LISP-forms can be also used in <act>-part. They will be evaluated when the rule is applied. If necessary, we can set intermediate results into registers and variables by using the functions listed in Table 2.

Table 2. Functions of PLATON.

Function	Argument	Effect	Value
SR	<register-name> LISP - <form>	SR stores the result of the evaluation of the second argument in the register.	The result of the evaluation of the second argument
SV	<variable-name> LISP - <form>	SV stores the result of the evaluation of the second argument in the variable.	The result of the evaluation of the second argument
GR	<register-name>	GR gets the content of the register.	The content of the register
GV	<variable-name>	GV gets the value of the variable	The value of the variable
TR	<structure-2> or /	TR transforms the variables and registers in the structural pattern into their values.	The transformed structure
SU	<register-name> LISP - <form>	SU sets the register of the higher-level processing.*	The result of the evaluation of the second argument
SD	<register-name> LISP - <form>	SD sets the register of the lower-level processing.*	The result of the evaluation of the second argument
GU	<register-name>	GU gets the content of the register of the higher level*.	The content of the register
PUSHR	<register-name> LISP - <form>	PUSHR is defined as: (SR register-name (CONS form (GR <register-name>)))	The result of the evaluation of the second argument

The <end> type comprises four varieties, and rules are divided into four types according to their <end> types.

a) NEXT-type: The <end> is in the form (NEXT<state-name> <stry>). The <stry> corresponds to the right side of a rewriting rule, and represents the transformed structure. A rule of this type causes statetransition to the <state-name>, when it is applied.

b) NEXTB-TYPE: rule which also causes state-transition. But unlike the NEXT-type, state-saving is done and if further processing results in some failures, control comes back to the state where this rule is applied. The environments, that is, the contents of various registers, will be restored, and the next rule belonging to this state will be tried.

c) POP-type: <end>-part of this type is in the form (POP<stry>). When it is applied, the processing of this level is ended and the control returns to the higher level with the value <stry>.

d) FM-type: <end>-part of this type is in the form (FM<failure-message>). The sideeffects of the processing at this level, that is, register settings and so on, are cancelled (See section 4.).

In <stry>, we can use two kinds of variables, that is, the variables used in <strx> and registers. This structural pattern is called <structure-2>. This expression is more suitable for writing a transformational rule than Wood's BUILDQ-operation. For example,

```
input string          = (*CDE (A (R1(*B))) FG)

strx                  = (*:I (A (R1 :N)) :J)

stry                  = (*(A (R1 (* :N))(R2 /REG)) :J)

the content of /REG = (G (R3 H))  .
```

As the result of matching, the variables :I, :N, and :J are bound to the substructures (*CDE), (*B), and (*FG), respectively. The result of evaluating the <stry> is,

```
(* (A (R1 (*CDEB))(R2 (G (R3 H)))) FG).
```

If the rule is a POP-type one, then this structure will be returned to the higher-level processing. If it is NEXT- or NEXTB-type, then the control will transit to the specified state with this structure.

4. Push-Down and Pop-Up Operations

By means of NEXTB-type rules, we can set up decision points in a program. We can also do this by using push-down and pop-up operations. A rule in PLATON will find out a certain

syntactic clue by its structural pattern<strx>, and, at the same time, extract substructures from the input string. The structural pattern predicts that those substructures may have certain constructions, that is, they may compose noun phrases, relative clauses, and so on. Therefore, it is necessary to transfer them to the states appropriate for analyzing these predicted constructions, and to insert the resultant structures given back from these states into the appropriate places. In PLATON, these operations can be described in the <trans>-part of a rule. For example, suppose <trans>-part of a rule is,

```
( ((S1 :K :K)) ((S2 (* :I :J)/REG)) )  .
```

when the control interprets this statement, the substructures corresponding to the variable :K and (* :I :J) are transferred to the states S1 and S2, respectively. If the processings from these states are normally ended (by POP-type rules), then the results are stored into the variable :K and the register /REG. In this manner, by means of push-down and pop-up mechanisms, substructures can be analyzed from appropriate states. Processing from these states, however, may sometimes result in failure. In other words, predictions such that certain relationships must be found among the elements of substructures may not be fulfilled. In this case, the pushed-down state will send up an error message according to the cause of the failure by FM-type rule. An FM-type rule points out that a certain error occurs in the processing. If the NEXTB-type rules are used in the previous processing at this level, the control will go back to the most recently used NEXTB-type rule. If the NEXTB-type rules are not used at this processing level, the error message specified by the FM-type rule will be sent up to the <trans>-part of the rule which directed this push-down operation. (See Figures 4a and 4b.).

According to these error messages, the control flow can be changed appropriately. For example, we can direct such processings by describing the trans<trans>-part in the following way:

```
( ((S1 :K :K)(ERRI (EXEC ((S5 :K :K)) ((S6 (* :I :J) /REG))))))
                (ERR2 (TRANS ( S8 /)))
                ((S2 (* :I :J) /REG))  .
```

In the above example, the processing of the substructure :K from the state S1 will result in one of the following. (According to the type of the returned value, the appropriate step will be taken.)

a) Normal return: the processing of :K is ended by a POP-type rule. The result is stored into the variable :K and the next push-down is performed, that is, (* :I :J) will be transferred to the state S2.

b) Return with an error message: the processing of :K results in some failure, and a FM-type rule sends up an error message. If the message is ERR1, then :K and (* :I :J) will be analyzed from the states 5 and S6,

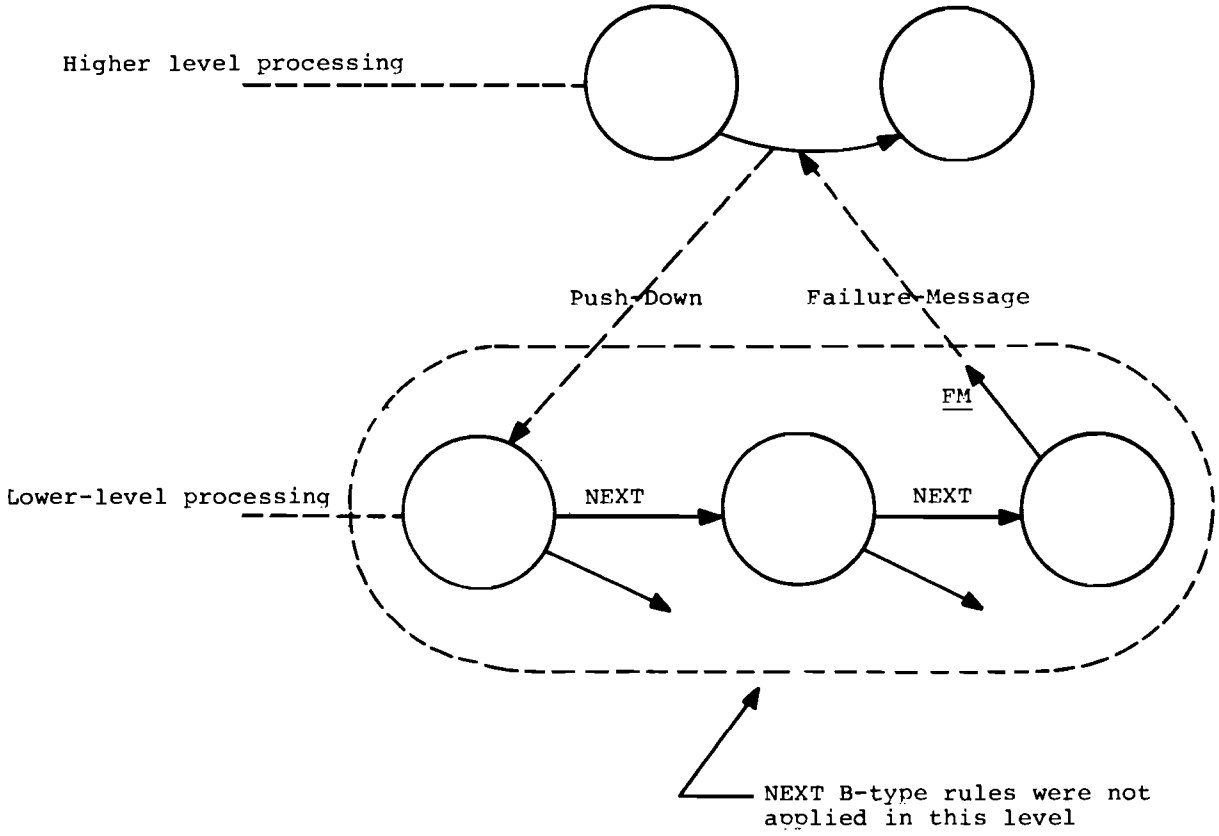


Figure 4a. Illustration of backtracking.

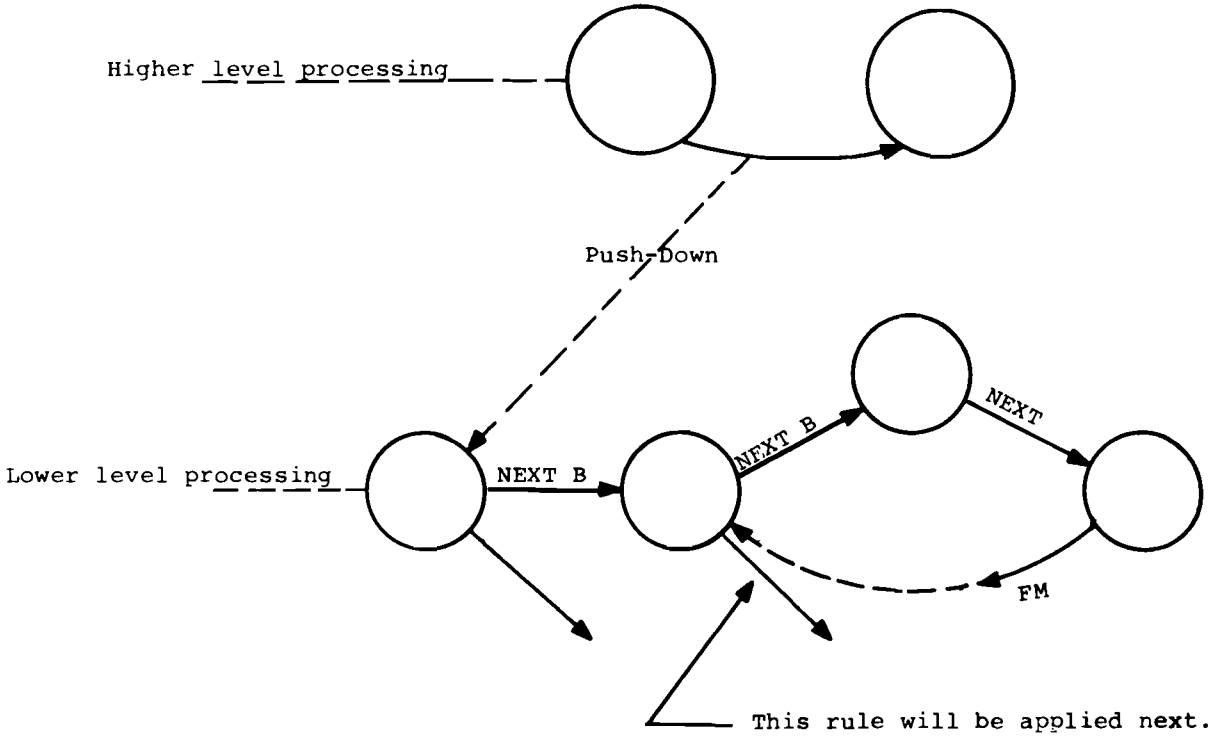


Figure 4b. Illustration of backtracking.

respectively (EXEC-type). If it is ERR2, the interpreter will give up the application of the present rule and pass the control to another state S8 (TRANS-type). If it is neither ERR1 nor the same step as (c) will be taken.

c) Return with the value NIL: the processing from the state S1 will send up the value NIL, if it runs into a blind alley, that is there are no applicable rules. The interpreter will give up the application of the present rule and proceed to the next rule attached to this state.

Mechanism in which the control flow can be appropriately changed according to the error messages from lower-level processings are not found in Woods's ATN-parser. We can obtain flexible backtracking facilities by combining these mechanisms with NEXTB-type rules.

5. A Simple Example

We are now developing a deductive question-answering system with natural language inputs--Japanese sentences. The internal data-base is assumed to be a set of deep case structures of input sentences. We adopted and modified Fillmore's case grammar to analyze the input Japanese sentences. Japanese is a typical example of a SOV language in which object and other constituents governed by a verb usually appear before the vert in a sentence. This makes Japanese very different from English and European languages. A typical construction of a Japanese sentence is shown in Figure 5. A verb may govern several noun phrases preceding it. A relative clause modifying a noun may appear in the form "--verb + noun--." The right end of the scope of the clause is easily identified by finding the verb. But the left end of it is often ambiguous. In Figure 5, the noun phrase NP_{i+1} is a case element of the verb V_1. On the other hand, the noun phrase NPi is governed by the verb V_2. Because the projection rule is kept in Japanese as in other languages, all the noun phrases between NP_{i+1} and V_1 are governed by V_1, and the noun phrases before NP_i are governed by V_2. However, in the course of analysis, such boundaries cannot be determined uniquely. The analysis program fixes a temporary boundary and proceeds to the next processing. If the temporary boundaries are not correct, the succeeding processing will fail, and the control will come back to the point at which the temporary boundary was fixed.

Now, we will show a simple example of strucutral analysis by PLATON. The example explains how the backtracking facility is used in analyzing Japanese sentences. Because we want to visualize the operations of PLATON without bothering with microscopic characteristics of Japanese sentences, we will take an imaginary problem as an example.

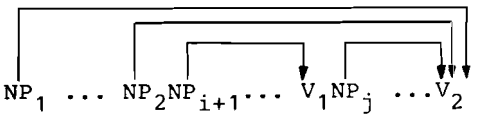


Figure 5. Typical construction of a Japanese sentence.

An input string is assumed to be a list. The elements of the list are integers and trees in the form of (X (SUM 0)). Here X may be regarded as a term modified by SUM 0. These two kinds of elements are arranged in an arbitrary order, except that the last element is the tree (X(SUM 0)). Figure 6. is an example of an input string.

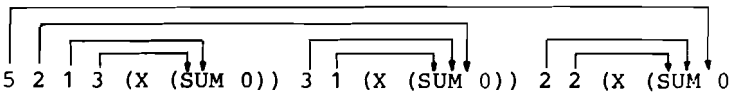
```
(* 5 2 1 3 (X (SUM 0)) 3 1 (X (SUM 0))2 2 (X (SUM 0)) )
```

Figure 6. An example string to be analyzed.

The result of the transformation is expected to be in the following form:

```
(* (X (SUM 4)) (X(SUM 6) (X (SUM 9)) )  .
```

This result is regarded as representing the following relationships between integers and X:



The number associated with an X by the relation SUM shows the sum of the integers which are governed by the X. We can look upon the relations between integers and an X as the relations between noun phrases and a verb in Japanese sentences. The result of the analysis is assumed to satisfy the following conditions:

a) Governor-dependent relationships between integers and an X must obey the projection rule.

b) As an imitation of a sematic restriction, we attach a condition that the sum of the integers governed by an X should not exceed ten.

c) As an imitation of a contextual restriction, we attach a condition that a result (* (X (SUM num-1)) (X (SUM num-2)).....(X (SUM num-N))) should maintain the relation, num-1≤num-2≤....≤num-N.

A set of rules is shown in the following. The corresponding statediagram is shown in Figure 7.

```
SUMUP 1: strx = (* :I :I1 (X (SUM :N)) :J)
         con  = (GREATERP 10 (PLUS :N :I1))
         act  = (SV :N (PLUS :N :I1))
                (PUSHR /REG :I1))
         end  = (NEXT SUMUP (* :I (X (SUM :N)) :J))
      2: strx = (* :I (X (SUM :N)) :J)
         con  = (CONTEXTCHECK /RESULT (TR (X (SUM :N))))
         act  = NIL
         end  = (NEXT BACKTRACK /)
      3: strx = (* :I (X(SUM :N)) :J)
         con  = T
         act  = NIL
         end  = (FM ERROR)
      4: strx = (*)
         con  = T
         act  = ((SR /RESULT (CONS 'X /RESULT)))
         end  = (POP /RESULT)

BACKTRACK 1: strx = (* :I (X(SUM :N)) :J)
         con  = T
         act  = ((SR /REG NIL)
                (SR /RESULT (APPEND /RESULT (TR (X (SUM :N)))))
         end  = (NEXTB SUMUP (*:I :J))
      2: strx = (* :I (X (SUM :N)) :J)
         con  = T
         act  = ( (POPR /TEMP /REG)
                (SV :N (MINUS :N /TEMP)))
         end  = (NEXT BACKTRACK (* :I /TEMP (X(SUM :N)) :J))
```

The input string is supposed to be the list shown in Figure 6. Since the start state is SUMUP, the first rule attached to this state is applied. This rule will find the leftmost X and an integer just before the X (by SUMUP-1-, strx). The variable :I1 is bound to this integer. This integer is added to the sum of the integers, :N, if the total does not exceed ten (SUMUP-1-, con). PUSHR used in <act>-part is a PLATON function which puts the second argument on the head of the first argument (SUMUP-1-, act). After this rule is applied, the control will enter the state SUMUP again (SUMUP-1-, end). That is, this rule is applied until there are no integers before the first X or the sum of the integers exceeds ten. As a result, the environment is the following:

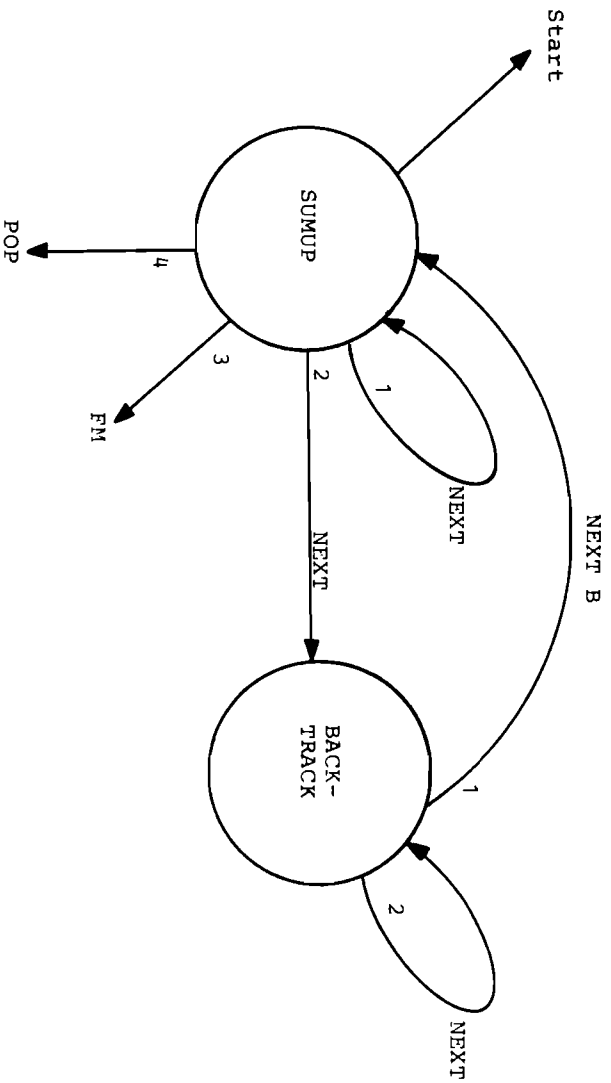
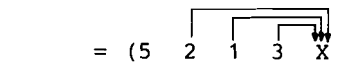


Figure 7. State diagram for a simple example.

structure under processing
= (*5 (X(SUM 6)) 3 1 (X(SUM 0)) 2 2 (X (SUM 0))) ,

relationships temporarily fixed between integers and X



content of /REG

= (2 1 3) .

The second rule of SUMUP will be applied next. This rule checks by its <con>-part whether the result at hand satisfies the third condition, that is, the contextual restriction. Because the content of /RESULT is NIL, the function CONTEXTCHECK returns the value T (SUMUP-2-, con). So this rule is applicable. The control makes the state transition to the state BACKTRACK (SUMUP-2-, end). Because the first rule of BACKTRACK is a NEXTB-type rule, state saving is performed. That is, the following environment is saved:

content of /REG = (2 1 3) ,
content of /RESULT = NIL ,
structure under processing
= (* 5 (X (SUM 6)) 3 1 (X (SUM 0)) 2 2 (X (SUM 0))) .

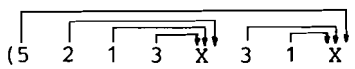
By this rule, the registers /REG and /RESULT are set as follows (BACKTRACK 1, act):

/REG = NIL
/RESULT = ((X (SUM 6))) ,

and the structure is transformed to:

(* 5 3 1 (X (SUM 0)) 2 2 (X (SUM 0))) .

A NEXTB-type rule causes the state transition as a NEXT-type rule. So the control returns to the state SUMUP (BACKTRACK-1-, end). At this state, the similar process as described above is performed. As the result, the following governor-dependent relationships are established:



Here, the bold lines indicate the newly established relationships. By the first rule of BACKTRACK, the following environment is saved:

content of /REG = (5 3 1) ,
content of /RESULT = ((X (SUM 6))) ,
structure under processing = (* (X (SUM 9)) 2 2 (X (SUM 0))) ,

and /REG and /RESULT are set as the following (BACKTRACK 1

```
/REG ; = NIL.
/RESULT ; = ( (X (SUM 6)) (X (SUM 9)) )  .
```

The transformed structure is (BACKTRACK-1-, end):

```
(* 2 2 ( X (SUM 0)) )  .
```

The control is transferred to the state SUMUP. By applying the first rule of this state repeatedly on the above structure, the following structure is obtained:

```
(* (X SUM 4)))  .
```

However, this result does not satisfy the contextual restriction. So the application of the second rule of SUMUP fails because the function CONTEXTCHECK used in <con>-part returns the value NIL (SUMUP-2-, con). That is,

```
CONTEXTCHECK [( (X (SUM 6))(X (SUM 9)) ); (X (SUM 4))] = NIL  .
```

The third rule, therefore, will be applied next. Because this rule is a FM-type rule (SUMUP-3-, end), it causes an error, and the control comes back to the point at which a NEXTB-type rule was most recently applied. The saved environment is restored. That is,

```
/REG ; = ( 5 3 1)  ,
/RESULT ; = ( (X (SUM 6)) )  ,
```

structure under processing; = (* (X (SUM 9)) 2 2 (X (SUM 0))) .

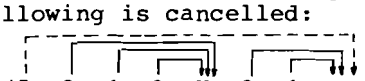
Then, by means of applying the second rule of BACKTRACK, the governor-dependent relationship last established in the previous process is cancelled. The structure and the register /REG are changed as below (BACKTRACK-2-, act):

```
/REG ; = (3 1)  ,
Structure under processing; = (* 5 (X (SUM 4)) 2 2 (X (SUM 0)))  .
```

The control enters to the state BACKTRACK again. The application of the first rule saves the environment such as:

```
content of /REG = (3 1)  ,
content of /RESULT = ((X (SUM 6))  ,
structure under processing = (* 5 (X (SUM 4)) 2 2 (X (SUM 0)))  .
```

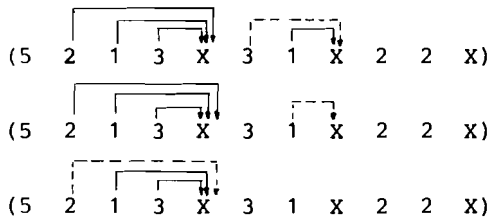
That is, the relationship indicated by the dotted line in the following is cancelled:



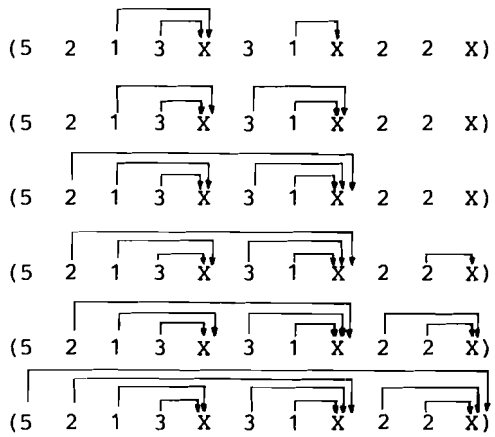
```
(5  2  1  3  X  3  1  X  2  2  X)  .
```

The control transits to the state SUMUP (BACKTRACK-1-, end), and a similar process is performed. However, because the governor and dependent relationship between the integer 5 and the second X is canceled, the sum of the integers governed by the first X,

(2 1 3), is greater than that of the second X, (3 1). The contextual condition, therefore, is not fulfilled, and the application of the second rule of SUMUP will not succeed. So the temporarily established relationships will be canceled one by one as follows:



After these relationships have been canceled, the desirable result is obtained by the following sequence:



At the final stage of the processing, the fourth rule of SUMUP, a POP-type rule, is applied and returns the value,

(* (X (SUM 4)) (X (SUM 6)) (X SUM 9))).

6. Conclusion

We have described a programing language for natural language processing. The language has several additional capabilities using the ATN-parser of W. Woods.

Grammars written by the language not only maintain the clarity of representation but also adequately provide the natural interface between syntax and other components. By means of the pattern-matching facility, we can write grammars in a quite natural manner, and, by its variable binding mechanism, semantic and contextual LISP functions are easily incorporated in syntactic patterns.

Flexible backtracking mechanisms and push-down operations make the complicated nondeterministic processing possible in a very simple way.

We are now developing an analysis program of Japanese using this language. The program can accept fairly complicated sentences from a textbook of elementary chemistry. It can utilize the lexical and contextual information of chemistry adequately during the analysis. Such information in our system is expressed in the form of a semantic network similar to that of R. Quillian.

Perhaps, PLATON itself must be equipped with more semantics and context-oriented operations such as specified lexical descriptions and functions using these. However, what description method is most efficient, and, moreover, what semantic information must be stored in lexicon are still not clear enough. So, as the first step, PLATON leaves many parts of these problems to user's specification by LISP programs. PLATON is written in LISP1.5 and implemented on FACOM 230-60 at Kyoto University computer center and a mini-computer TOSBAC-40 in our laboratory. The interpreter of PLATON itself requires only 4.5 Kcells.

References

[1] Bobrow, D. and Fraser, B. An Augmented State Transition Network Analysis Procedure. Proc. 1st IJCAI, 557-68, 1969.

[2] Colmerauer, A. "Les systemès-q ou un formalise pour analyser et synthétiser des phrases sur ordinateur." University of Montreal, TAUM 71, 1971.

[3] Filmore, C.J. "The Case for Case." In Bach and Harms, eds., Universals in Linguistic Theory. New York: Holt, Rinehart 1968.

[4] Hewitt, C. "PLANNER: A Language for Manipulating Models and Proving Theorems in a Robot." In Artificial Intelligence, 1969

[5] Nagao, M. and Tsujii, J. "Mechanism of Deduction in a Question-Answering System with Natural Language Input." Proc. 3rd IJCAI, 1973.

[6] Nagao, M. and Tsujii, J. "Programming Language for Natural Language Processing - PLATON." J IPSJ, 15 654-61, 1974.

[7] Pratt, V. "A Linguistic Oriented Programming Language." Proc. 3rd IJCAI, 372-81, 1973.

[8] Rulifson, J. et Al. "QA4-A Language for Writing Problem-Solving Programs." Technical Note 48, Stanford Research Institute, 1970.

[9] Throne, J. Brately, P., and Dewar, H. "The Syntactic Analysis of English by Machine." In Michie, ed., Machine Intelligence vol 3. New York: Elsevier, 1968.

[10] Winograd, T. "Procedures as a Representation for Data in a Computer Program for Understanding Natural Language." Massachusetts Institute of Technology Thesis, 1971.

[11] Woods, W. "Augmented Transition Network Grammars for Natural Language Analysis." CACM, 13, 591-602, 1970.

[12] Woods, W. "Procedural Semantics for a Question-Answering Machine." Proc. FJCC., vol. 33, 457-71.

APPENDIX

Mechanism of Deduction in a Question-Answering System with Natural Language Input

Abstract

We have constructed a deductive question answer-system which accepts natural language input in Japanese. The semantic trees of assertional input sentences are stored in a semantic network and interrelationships--conditional, implicational, and so forth--are established among them. A matching routine looks for the semantic trees which have some relations to a query, and returns the mismatch information (difference) to a deduction routine. The deduction routine produces subgoals to diminish this differences. This process takes place recursively until the difference is completely resolved (success), or there is no other possibility of matching in the semantic network (failure). Standard problem solving techniques are used in this process. As the result, the system is very powerful in handling deductive responses. In this paper, only the part of the logical deduction is explained in detail.

Descriptive terms: question answering, deduction, natural language, semantic network, problem solving.

1. Introduction

There are a few deductive question-answering systems using natural language, almost all of which use logical expressions, especially the first order predicate calculus expression, as an intermediate language. However systems which use formal logics have problems:

a) Syntactic and semantic analyses of natural language input are necessary to transform the input to logical expression without ambiquity.

b) The axiom set must be clearly defined and must not be contradictory.

c) Predicates and variables must be fixed beforehand. This is a problem for the system's expansion. Also this prevents mixing the first and higher order, predicate calculus systems.

d) Deduction using the resolution principle is cumbersome. Usually question answering does not require a deep deductive process.

e) Good quality of natural language output is very hard to obtain from a logical expression.

To avoid the above problems we have used a kind of semantic reprsentation of natural language sentences as an intermediate expression. Our system has the following characteristic features.

a) The question-answering system is a composite of subsystems for language analysis, deduction, and language generation.

b) The parsed trees of sentences are permitted to have some ambiguities. Ambiguities are resolved in the process of logical deduction.

c) During the question answering process, the deduction ability is increased and the area which the system can deal with is also expanded. The deduction ability of a system depends on how many theorems the system can use, and on how efficiently it can deal with them. We have constructed a system in which the available theorems increase during the question-answering process.

d) Facts can play the role of theorems. We think the distinction between facts and theorems is not clear enough. A statement can be used as a theorem at one time and as a fact at another time. For example,

A human is an intelligent animal.

plays the role of a theorem to answer

Is Smith intelligent?

because Smith is an instance of a variable "human." On the contrary it plays the role of a fact to the question

Is a man an animal?

because "a human" is treated as an instance of a variable "man." In our system the assertions given by a user, which correspond to facts in usual systems, can play the role of theorems. This is accomplished by allowing a higher-concept term to be a variable to its lower-concept term. There is no distinction between them, and both facts and theorems have the same structures in the data base. This is the most significant character of the system we have developed.

e) In order to deal with a large data base, the system has a well-organized data structure and relevant information to a question is accessed by a technique of indexing and retrieval.

f) The deduction process is similar to that of humans. It allows introducing many heuristics into the deduction process.

In this paper, the details of deduction subsystem alone are explained. The other two subsystems will be published elsewhere in the near future.

2. System Organization

A block diagram of our system is shown in Figure 1. The internal data base of the system is divided into two parts:

a) semantic representations (semantic trees) of input sentences

b) network (mutual connection) of a).

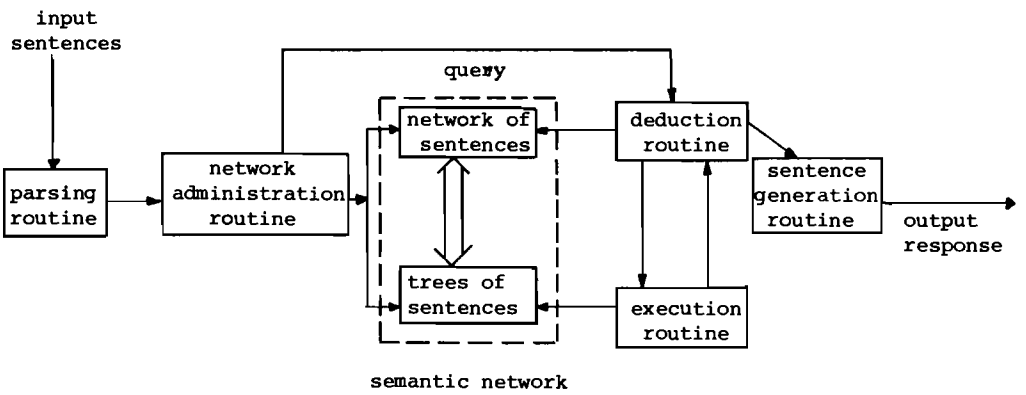


Figure 1. Organization of the system.

The mutual connection consists of interrelationships such as conditional, implicational, and so forth. An input sentence is analyzed into a semantic tree, and it is read into the semantic network if it is an assertion and is not in the network yet.

Thus knowledge accumulates in a very natural way in the question-answering process. An inverted file of keywords makes it easy to extract information relevant to the question.

The parsing routine performs syntactic and semantic analyses of an input query sentence, and produces the parse tree. A network administraction routine accepts the tree and relates it to the semantic network which contains sentences already accepted.

To accomplish a deduction, there are two main parts: the execution routine and the deduction routine. The execution routine, which plays the central role in the deduction process, searches through the network for sentences relevant to the current goal and matches them one by one against it. The deduction routine manages the global information in the problem-solving process such as goal-subgoal relationships, variable bindings (for example the word man is bound to the word Smith), and so forth. This routine also directs the execution routine to determine which sentence must be verified first.

3. Knowledge Structure

3.1 Semantic Trees

We have applied a kind of dependency analysis to the input Japanese sentences. A noun modified by an adjective is transformed into a kernel sentence having another kernel sentence related to the noun. The sentence

KINBEN NA HITO WA SEIKO SURU
(A diligent man will succeed.)

is divided into two sentences like

HITO WA SEIKO SURU
(A man will succeed.)

and,

HITO WA KINBEN DA
(A man is diligent.)

The parsed tree structure of this sentence is shown in Figure 2.

Some sentences in Japanese have two possible subject phrases, that is, one which contains the reference particle "GA" and the other which contains "WA." We consider the relational phrase with the particle "WA" as indicating what the sentence talks about; the phrase with "GA" is the subject phrase corresponding to the predicate in the sentence:

ZO WA HANA GA NAGAI
(Elephant has a long nose.)

is a typical example. Its literal translation is, "As for elephant the nose is long." The tree structure of it is shown in Figure 3.

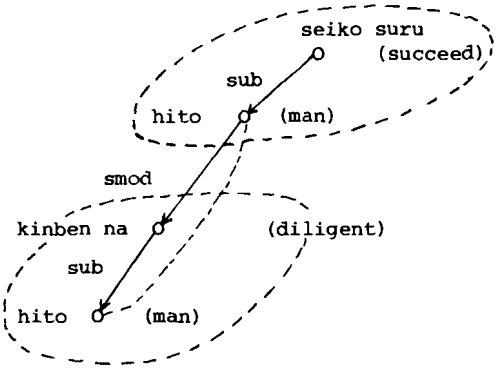


Figure 2. Kinben na hito wa seiko suru.
(A diligent man will succeed.)

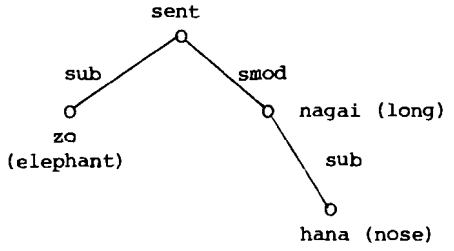


Figure 3. Zo wa hana ga nagai.
(Elephant has a long nose.)

Sentences connected by and or or are represented in the tree structure as shown in Figure 4.

A sentence which contains upper concept terms replaceable by their lower concept terms is considered as a theorem available to prove a statement which has the lower concept terms in it. So upper-lower concept relationship among words plays an important role in our system. The input sentence in the form of

" A WA B DA" meaning A is a lower concept of B, and B is an upper concept of A, has a special structure to express the relationship clearly. "NINGEN WA KASHIKOI DOBUTSU DA" (A man is an intelligent animal.) is parsed as shown in Figure 5.

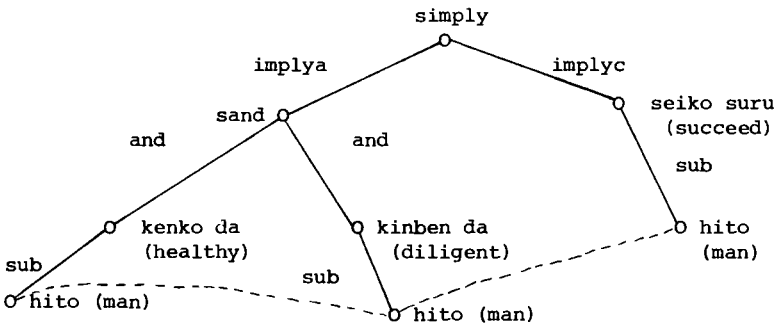


Figure 4. Kenko de kinben na hito wa seiko suru. (A man who is healthy and diligent will succeed.)

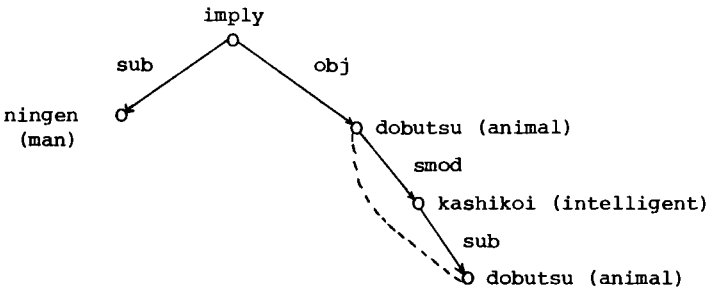


Figure 5. Ningen wa kashikoi dobutsu da. (A man is an intelligent animal.)

Properties of sentences are attached to the top node of the parsed tree structure. The properties we treated are potential, active, passive, subjective, tense, and so forth. The assertion sentence is regarded as true, so that a sign T is given to the property part of the parsed tree. The signs F and U in the property part indicate false and undetermined respectively.

3.2 Semantic Network

The network is constructed in the following way.

a) In the case of an assertion sentence S_1, it is stored in the form shown in Figure 6a.

b) In the case of a negation sentence, schematically written as "not S_2," it is stored in the same form as Figure 6a, but the property part is written as F.

c) If a sentence is "If S_1, S_2," it is stored in the form shown in Figure 6b.

d) If a sentence is "Because S_1, S_2," it is stored in the form shown in Figure 6c.

e) If the sentences S_1 and S_2 in (1)--(4) are found in the semantic network, they are not stored newly, but the stored ones are used. For example, the following sentences are stored in the network as shown in Figure 6d.

Because S_1, S_2.
If S_1, then S_3.

In this case because S_1 is asserted as true, S_3 is also true.

The network and parsed trees have the following internal constructions.

a) Branches in the network and trees are bidirectional for flexible transformation and for efficient search in the deduction process.

b) Words are not stored in nodes of the parsed trees but by a pointer to the lexical entry of the word (Figure 7).

c) The lexical entry of a word, called NLIST, contains not only lexical information about the word, but also a list of sentences (pointers to the entries of the sentences in SLIST) which contains the word. NLIST is a kind of inverted file of keywords.

d) The node of the network is indicated by a pointer from a table, called SLIST, which contains information about the sentence. The information of whether the sentence is true (T), false (F), or undetermined (U), and so forth, is stored in this list.

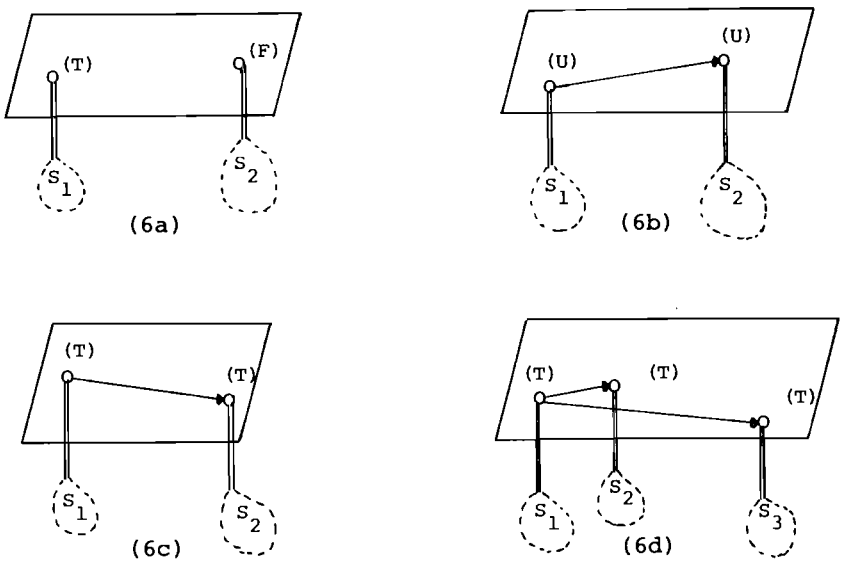


Figure 6. Relations in semantic network.

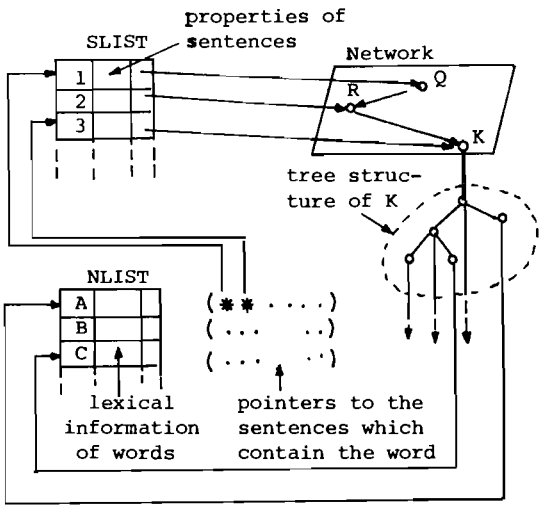


Figure 7. Internal data base structure.

e) Different nodes in the network correspond to different sentences. As a result, information about a sentence can be retrieved from a single node in the network.

4. Execution Routine

Among many intellectual abilities of humans, we have implemented in this study the deduction ability based on the use of "the law of substitution" and "the law of implication." This is realized by the exectuion routine and the deduction routine. The execution tries to match a sentence structure against another one, regarding an upper concept as a variable over its lower concepts. The deduction routine produces subgoals and tells the execution routine which sentence must be verified first. The execution routine searches through the network for the sentences which are equivalent to the goal sentence given by the deduction routine. It consists of three main parts: keyword search, matching, and resolving differences.

4.1 Keyword Search

The system has an inverted file of words called NLIST. By using this file, the execution routine takes out the sentences which contain words in the goal sentence. These selected sentences are presumed to be relevant to the current sentence.

4.2 Matching Method

The matching algorithm is constructed so that two parsed trees which are different in the sequence of branches (Figure 8) will be matched successfully by the branch labels on the parsed trees. Matching between two parsed trees fails for various reasoı

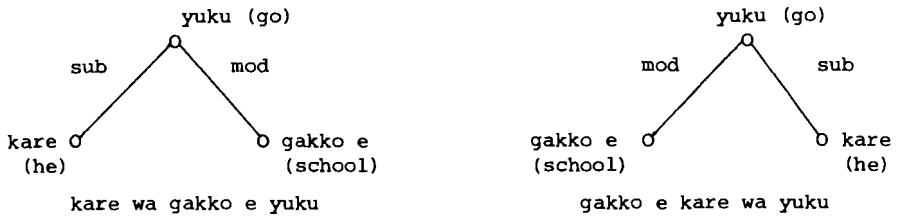


Figure 8. Change of word order.

The causes of mismatch, named differences, are classified into the following four classes.

a) N-difference: The words which are attached to the corresponding node are different in the two sentences. Figure 9a shows an example, where the difference is expressed as (N (*C *D)). *C shows the pointer to the node C.

b) S1-difference: One structure (first argument) has extra branches which the other does not have. Figure 9b shows an example of this category, abbreviated as (S1 ((*R4) *B)), which shows the branch R4 is the extra one.

c) S2-difference: One structure (second argument) has extra branches. Figure 9c is an example and this difference is shown by (S2 (*C (*R5))).

d) S0-difference: Both structures have extra branches. An example is shown in Figure 9d.

The matching subroutine tries to match its first argument against its second one. If the matching succeeds, the subroutine returns "success" to the deduction routine. If not, it returns the differences.

4.3 Resolving Differences

The execution routine first picks up sentences expected to be relevant to the given sentence by using NLIST, and then tries to match them against the given sentence. If the same sentence is stored in the data base, the execution routine picks it up and the matching ends in success. If there is no complete match, but a difference, N-routine or S-routine is activated according to the kind of difference to resolve the difference.

a) N-routine

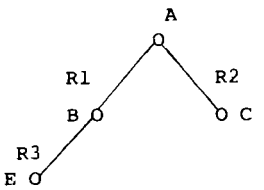
An N-routine arises from mismatch of words. Let us suppose that the sentence

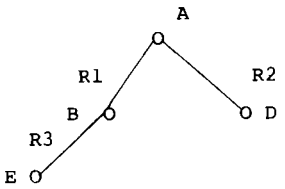
TARO WA SEIKO SURU
(Taro will succeed.)

is what the deduction routine tells the execution routine to prove, and the sentence

NINGEN WA SEIKO SURU
(A man will succeed.)

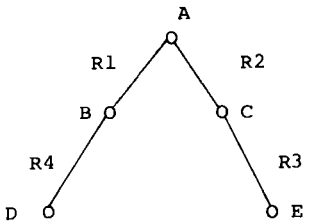
is stored in the data base. The matching between these does not succeed and the difference is (N (*TARO *NINGEN)). This difference is transferred to N-routine and the routine tries to check whether the word TARO is a lower concept of NINGEN (man) by searching through

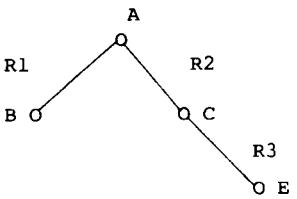




(N (*C *D))

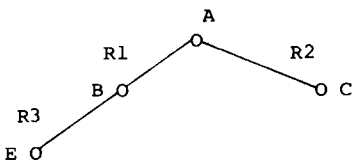
9a)

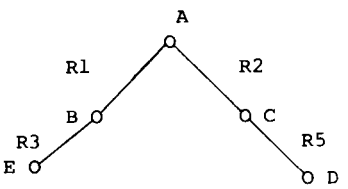




(S1 ((*R4) *B))

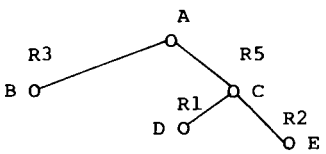
9b)

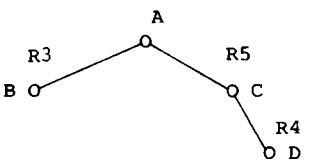




(S2 (*C (*R5)))

9c)





(SO ((*R1 *R2) (*R4)))

9d)

Figure 9. Differences in matching.

the network for the sentence "TARO WA NINGEN DA," which means "TARO is a lower concept of NINGEN." If such a sentence is found, NINGEN can be looked upon as a variable which can take the value TARO, and then the difference is resolved. This is considered as the process of substitution. By this process, the system can deduce specific facts from generalized knowledge.

N-routine basically searches the sentence "A WA B DA," which means "A is B," in order to resolve the difference (N (*A *B)), but many sentences in the network are in such forms as "A WA b NA B DA," which means "A is B modified by b," and "aNA A WA B DA," which means "A modified by a is B." The differences to be resolved also take the forms of (N (*(a NA A) *B)) and (N (*A *(b NA B))). Four cases are possible.

a1) Difference : (N (*a NA A) *B))
In the data base : A WA B DA

In a logical representation,

the goal to be proved is: $a(x) \wedge A(x) \rightarrow B(x)$,
the fact in the network is: $A(x) \rightarrow B(x)$.

and the difference is resolved immediately.

a2) Difference : (N (*A *(b NA B)))
In the data base : A WA B DA

In a logical representation,

the goal to be proved is: $A(x) \rightarrow b(x) \wedge B(x)$,
the fact in the network is: $A(x) \rightarrow B(x)$.

In this case whether A satisfies the condition b or not is produced as a subgoal.

b1) Difference : (N (*A *B))
In the data base : A WA b NA B DA

In a logical representation,

the goal to be proved is: $A(x) \rightarrow B(x)$,
the fact in the network is: $A(x) \rightarrow b(x) \wedge B(x)$.

So the difference is resolved.

b2) Difference : (N (*A *B))
In the data base : a NA A WA B DA

In a logical representation,

the goal to be proved is: $A(x) \rightarrow B(x)$,
the fact in the network is: $a(x) \wedge A(x) \rightarrow B(x)$.

In this case a subgoal is produced.

b) S-routine

S-routine resolves S1-, S2-, and S0- differences. These differences arise from mismatch of branches. S-routine is given two different sentence structures, one is called S-structure and the other is called T-structure. Using grammatical rules (especially transformation rules), this routine transforms the S-structure into several transformationally equivalent structures, and matches them against the T-structure. At present not so many transformational rules are prepared. Figure 11 is an example (as shown in Section 6). If the matching succeeds, the two structures, S-structure and T-structure, are equivalent and the difference is resolved.

5. Deduction Routine

The deduction routine controls the whole of the deduction process. This routine has a global knowledge of the process. This knowledge contains the goal-subgoal organization, variable binding, and so forth. The deduction routine tells the execution routine which sentence must be verified and which sentence, if the first trial fails, has to be verified next.

5.1 Goal Organization

The deduction method in our system takes a question Q as a goal and tries to verify it by means of matching it with the sentences stored in the network. If the trial fails, the deduction routine searches through the network for such sentences as $P \rightarrow Q$. Those sentences P's, if any, are considered as subgoals to accomplish the previous goal. In the same manner, sub-subgoals are produced to accomplish the subgoals. As the process advances, many goals are produced hierarchically. An AND-OR tree structure is used to remember the hierarchically organized relationships among goals.

Subgoals are created in various cases.

a) If a goal sentence G can not be determined to be true or false, subgoals are created by means of searching through the network for the sentences which are antecedents of G.

b) In the same case of a), the negations of consequences of G are taken as subgoals. If they are proved to be true, the sentence G is determined to be false.

c) If the matching between two parsed trees is incomplete, subgoals to diminish the mismatches are created.

In addition to these cases, subgoals are also produced when a goal is divided into several subgoals. For example,

"KARE WA KINBEN DE SHOJIKI DA" (He is diligent and honest.) is divided into, "KARE WA KINBEN DA" (He is diligent.) and, "KARE WA SHOJIKI DA" (He is honest).

The goals are tried one by one, and when there remains no goal, the deduction process stops with a failure message. A goal which has several subgoals will succeed or not, depending upon whether the subgoals will succeed or not. A goal keeps some information for itself. For example, it has the information of whether it is an AND-type or an OR-type. Depth of goal shows the depth between the top-goal (that is, a question given by a user) and the present goal. The depth of the top-goal is 0 and the depth of the immediate subgoal is 1.

The deduction routine chooses a goal, the depth of which is the smallest of all, and tells the execution routine to verify it. The indicators such as KOTEI (positive assertion), HITEI (negative assertion), MATCH (to be matched), and so forth show the effects of the goals' results to be transferred to their previous goals. KOTEI (HITEI) shows that if this goal succeeds, the sentence corresponding to its previous goal is proved to be true (false). The subgoals which are produced in order to resolve the mismatch between two parsed trees have the indicator MATCH.

5.2 Variable Binding

To use the law of substitution is one of the most important abilities in this system. This is carried out by considering an upper concept as a variable over its lower concepts. A word behaves as a constant when it is a lower concept of another word, and as a variable when it is an upper concept of another word. We do not introduce unary predicates such as "human(x)," "animal(x)," which are usually used in the predicate calculus system in order to restrict the range of variables.

We regard all words as variables which have their own domains of values. We illustrate this by the following example.

(1) HITO GA KENKO NARA-BA HITO WA SEIKO SURUR
(If a man is healthy, the man will succeed.)

(Q) Smith WA SEIKO SURU KA?
(Will Smith succeed?)

The system searches through the network to find out the sentence (1) which is expected to answer the given question. The matching between the consequent part of (1) and the question fails at first. The cause of mismatch is N difference between "Smith" and "HITO (man)." N routine is called to find out that HITO is an upper concept of Smith, which is proved by the information "Smith is a man" in the network. Thus a subgoal, the antecedent of (1), in which HITO is replaced by Smith is produced,

that is, "Is Smith healthy?" As the deduction process proceeds, several such bind conditions are produced. Each goal must be tried taking into consideration the related bind conditions produced during the former process.

The deduction routine has a stack to remember these conditions. This stack is illustrated in Figure 10. Each goal has a pointer to this stack and the routine can retrieve the corresponding bind condition of a goal. If a goal fails, then the bind condition generated during the trial of the goal is abandoned. On the other hand, if a goal succeeds, its condition is memorized for use in the succeeding process.

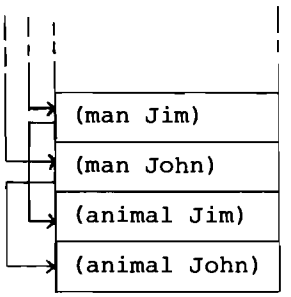


Figure 10. Stack for variable binding.

6. Comparison with the Systems Using Predicate Calculus

Those systems which use predicate calculus translate the input into a predicate calculus formula, store it in the data base, and use a universal method of deduction such as the resolution method. In those systems, common subexpressions appearing in different sentences are stored as many times as they appear in different logical formulas. This is not efficient. In our system, the same subexpressions are stored only once, and their relations to the other parts of sentences are stored by links. So these interrelationships can be utilized in the deduction process. Especially when the system deals with a great amount of data and only a relatively small portion of the data has a direct relation to the given question, the quick access to these related expressions is very important in the deduction process.

Which sentences or formulas are available for the current problem needs to be recognized easily, and to do this, a well-organized data base is necessary. It is tempting to try to incorporate the use of property lists to speed up resolution. For example, one may find it useful for each object symbol c to have access to a chained list of all literals or clauses where c occurs.

A difficult but more important problem is to recognize how a meaningful unit is related to another unit. It is desirable for the data base to contain information about the interrelationships among the meaningful units. In our system, the deduction procedure can retrieve from a node those sentences which have some relation to the sentence corresponding to the node.

Another is that disambiguation is done not only in the parsing phase but also in the deduction phase. For example, the sentence "A NO B WA..." may have more than four different structures in deeper levels, according to the words A and B. That is:

KARE NO KANE (he) (money) — the money which he has

KYOSHI NO KARE (teacher) (he) — he, who is a teacher

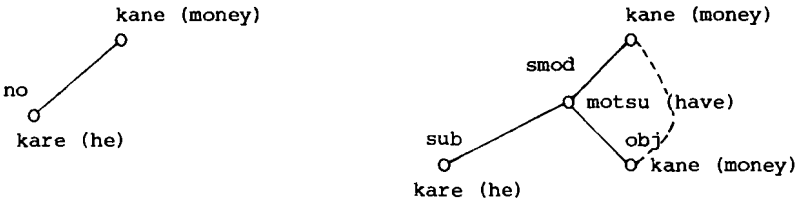
KYONEN NO SENYO (last year) (election) — the election which was taken place last year

The parsing and translation program in predicate-calculus system must choose one of these structures at the input and parsing stage because predicate-calculus formulas never permit ambiguous expressions. But it is almost impossible to classify each word into a certain semantic category, and to decide which of the above structures is proper to the sentence according to the information that the word A belongs to a certain category and B belongs to another.

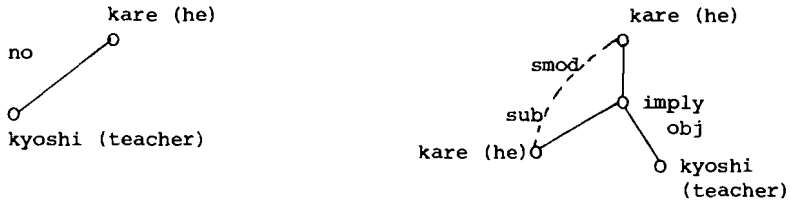
In our system, "A NO B WA ..." is stored as shown in Figure 11. The ambiguity is left in its structure. The matching routine transforms the sentence into several different structures by using grammatical knowledge, and tries to match them one by one against the object structure. All of them except one correct structure may not match against it. Thus, ambiguous structures are resolved during the deduction process. This is also one of the excellent features of using semantic structures of sentences which permit ambiguous structures as an internal data representation.

7. Examples

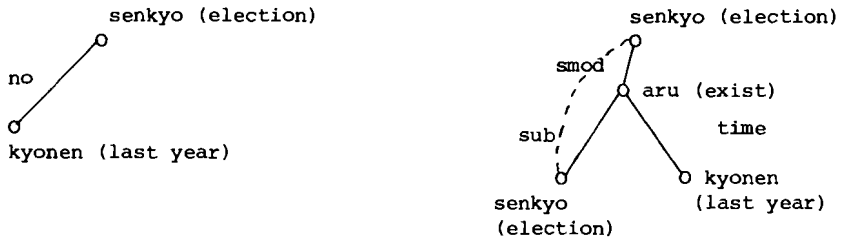
Example 1
Input sentences:
HITO WA KENKO DE KINBEN NARA SEIKO SURU.
(If a man is healthy and diligent, the man will succeed.)
HITO WA sportsman NARA KENKO DESU.
(If a man is a sportsman, the man is healthy.)
Jim WA sportsman DESU.
(Jim is a sportsman.)



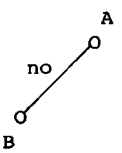
A1)



A2)



A3)



B)

Figure 11. A) Several possible deep structures for "A no B."
A1) the money which he has.
A2) he, who is a teacher.
A3) the election which took place last year.
B) Internal representation in our system.

Jim WA KINBEN DESU.
(Jim is diligent.)
Jim WA KASHIKOI HITO DESU.
(Jim is a clever man.)
Question given to the computer
Jim WA SEIKO SHIMASU KA?
(Will Jim succeed?)
Response from the computer

Jim WA KENKO DE KINBEN KA?	(Is Jim healthy and diligent?)
Jim WA KENKO KA	(Is Jim healthy?)
Jim WA sportsman KA?	(Is Jim a sportsman?)
Jim WA KINBEN KA?	(Is Jim diligent?)
HAI, Jim WA SEIKO SURU.	(Yes, Jim will succeed.)

These outputs except the last are the intermediate ones from the computer, to which no answers are necessary.

Example 2
Input sentences
Jim was killed by John.
A man-A who killed a man-B is punished.
Jim is a man.
John is a man.
Question given to the computer
Is John punished?
Responses from the computer
Did John kill a man-B?
Yes, John is punished.

Example 3
Input sentences
Whale bears a child.
An animal which bears a child is a mammal.
If an animal is a mammal, the animal is a vertebrate.
A vertebrate has a backbone.
Question given to the computer
Has whale a backbone?
Responses from the computer
Is whale a vertebrate?
Is whale a mammal?
Does whale bear a child?
Yes, whale has a backbone.

In these examples, intermediate responses are to show the deduction processes, which do not need answers from a man.

References

[1] Quillian, M.R. "The Teachable Language Comprehender; A Simulation Program and Theory of Language." CACM, 12, 456, 1972.

[2] Raphael, B. and, Green, C. "The Use of Theorem Proving Techniques in Question Answering System." JACM, 169, 1968.

[3] Sandewall, E.J. "Formal Methods in the Design of Question-Answering System." J. Art. Inter., 237, 1972.

[4] Shapiro, S.C. "A Net Structure for Semantic Information Storage, Deduction and Retrieval." AI Conf., 71, 512, 1971.

The TGS-4000 Translator - Generator System

D.D. Alexandrov

1. Introduction

The implementation of a compiler is a task that involves a considerable amount of effort. A large number of programing systems called compiler-compilers (or translator-generators) have been developed in an attempt to make the production of compilers a less onerous task. An overall view of techniques used in such systems is given by Feldman and Gries [3]; that report also contains a large number of references.

A compiler-compiler can be considered as a programing system in which a source program is the formal description of a language and the object program is the translator for that language. As such, the source program for a compiler-compiler is merely a formalism for describing a translator. Consequently, the source program must contain explicitly or implicitly a description of the lexical analyzer, the syntactic analyzer, the code generator and/or interpreter, and the various other phases of the translator to be constructed. The translator-generator is a tool providing an environment in which these descriptions can be easily written down.

This paper presents a translator-generator system for RC-4000 computer, producing syntax-directed translators for context-free languages.

2. Architecture of the Generator

Figure 1 represents the structure of the TGS-4000 system. Source text for the translator T_M^{λ} is the formal definition of the syntax of a language L in terms of metasyntactic language γ, and the definition of the semantics in terms of metasemantic language σ^{λ}.

The translator T_M^{λ} transforms this text into machine-oriented tables $\{\tau\}$, representing the syntax of the language in compact form, and text $\{\lambda\}$ of syntax-directed translator for the language L in terms of the programing language λ. Futhermore, the compiler T_{λ} translates the text $\{\lambda\}$ into object code--translator T_L for The language L, driven by the tables $\{\tau\}$.

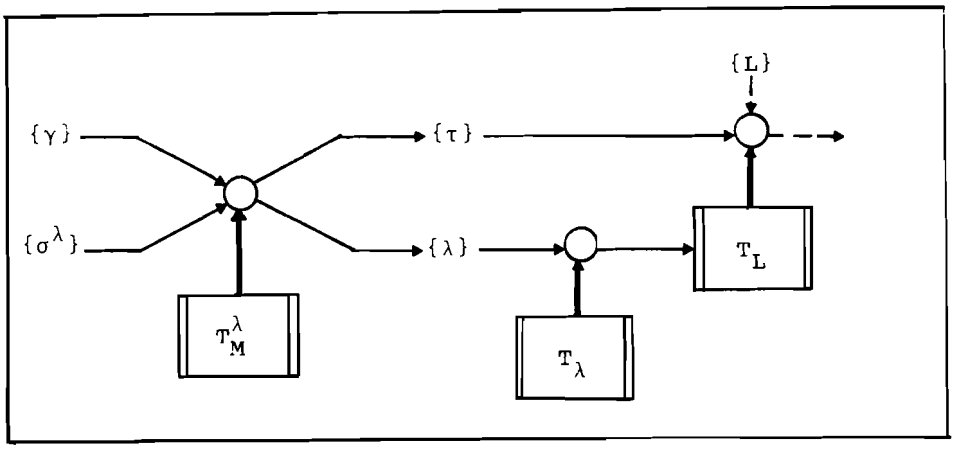


Figure 1.

The scheme above is implemented by the author on the computer RC-4000 [5]. In this implementation, the metasyntactic language γ is a machine-oriented version of the Backus-Naur Form (BNF) [9]; the metasemantic language σ^{λ} is an extension of the language ALGOL-6 [4]; the intermediate language λ is ALGOL-6; the translator T^{λ}_{M} is written on ALGOL-6 and FP-language [7]; and, for compiler T_{λ}, the highly efficient Regnecentralen's ALGOL-6 compiler is used. Beginning with a translator T^{λ}_{M} written by hand, the system was boot-strapped onto a more efficient and powerful one.

3. TGS-4000 Metalanguage

The metalanguage for defining languages in the system TGS-4000 will be described by means of a modified Backus notation. The new metalanguage element introduced is:

$$\left[\begin{array}{l} \langle \text{ string 1 } \rangle \\ \quad \cdot \ \cdot \ \cdot \\ \langle \text{ string n } \rangle \end{array}\right]_{a}^{b} .$$

The meaning is that any sequence of these strings may appear at this place in the construction; a and b give the minimum and the maximum number of strings in the sequence. Where either index is represented by a variable, the domain of the variable is all metaexpressions. So,

< language definition > :: =

< any sequence not containing @ >

< syntax definition >

< semantics definition > .

The metalanguage is a concatenation of two languages--metasyntactic and metasemantics languages. This permits the independent use of the two languages when there is a need to change only the syntax or the semantics for a given language.

3.1 Metasyntactic Language

In order to make BNF sentences suitable for input to a computer program, their form is slightly modified.

3.1.1 Syntax

<syntax definition> ::= @ syntax: < list of rules >

< list of rules > ::= $\left[\text{< rule >} \right]_{1}^{\text{nrules}}$

< rule > ::= < left-hand part > = < right-hand part >.

< left-hand part > ::= <identifier >

< right-hand part > ::= < part of right-hand > !
< right-hand part > ! < part of right-hand >

< part of right-hand > ::= < element > !
<part of right-hand> <element>

<element> ::= <identifier> !
<basic symbol> !
<unsigned number>

<basic symbol> ::= <string>

3.1.2. Semantics

The basic symbols (=) and (!) stand, respectively, for the metasymbols :: = and 1 of the strict BNF; < identifier >, < string >, < unsigned number >, and < basic symbol > have their usual meaning; the first three are considered here as terminal symbols. The characters SP (space) and NL (new line) may be inserted freely everywhere except inside terminal symbols. Text between two symbols is ignored.

3.2 Metasemantics Language

3.2.1 Syntax

< semantics definition > :: =

@ environment: < environment description >

@ translator: < translator description >

@ interpreter: < interpreter description >

< environment description > ::=

$$\left[\text{< block head >} \left[\begin{matrix} \text{< statement >} \\ \text{< empty >} \end{matrix} \right] \begin{matrix} \infty \\ 0 \end{matrix} \right] \begin{matrix} N \\ 0 \end{matrix}$$

$$\text{< translator description >} :: = \left[\text{< statement >} \right] \begin{matrix} Nrules \\ 1 \end{matrix}$$

$$\text{< interpreter description >} :: = \left[\text{< compound tail >} \right] \begin{matrix} N \\ 0 \end{matrix}$$

Here Nrules is the number of syntax rules in the syntax definition. The definitions above must be considered in the context of the description of the language ALGOL-6.

3.2.2 Semantics

The environment description contains declarations of simple variables, array variables, switches, procedures, etc., which are used in the translation and/or interpretation phase. It may also contain statements describing some initial computations, preprocessing, communications with the operating system, and other processes.

Every statement in the translator description corresponds to one rule only in the syntax definition. It represents the step, which is executed in the parsing phase, if the corresponding syntactical unit is recognized. This step may be an immediate interpretation (one-pass translator) or a translation into an internal language (multipass translator).

The interpreter description may contain statements defining next passes of the translator and the termination of the translation.

Quantities, declared in the environment description, may be used in the corresponding blocks. In all phases of the translator special TGS-4000 quantities, standard identifiers in ALGOL-6 language, library procedures, and the RC-4000 list-processing system [1] may be used.

4. Nucleus of the TGS-4000 Generator

The translator T_M^{λ} proceeds as follows:

a) translates $\{\gamma\}$ into machine-oriented form:

- constructs the terminal Vt and nonterminal Vn vocabulary of the language L;

- codes the syntax rules for the language L into table t;

b) computes the precedence matrix M for the language L, containing the relations <. , ≐ , .> and ∴ [2];

c) if the grammar of L is not in precedence form, then transforms t to precedence form [8] and continues from step [b);

d) constructs an inverted indexed array I with pointers to t;

e) transfers the arrays t, I, Vt, and M (the set $\{\tau\}$ in an indexed sequential file, which is common for all translators, generated by TGS-4000 system. The set is saved under key equal to the name of the defined language, so an old version is deleted;

f) translates $\{\sigma^{\lambda}\}$ into $\{\lambda\}$.

The generated syntax-driven translator $\{\lambda\}$ has the following structure:

< operating environment description >

< environment description >

< parsing phase >

< interpreter description >

< terminating phase > ,

where,

< operating environment description > :: = $\left[\text{< block head >} \right]_3^3$

< terminating phase > :: = $\left[\text{< compound tail >} \right]_3^3$.

The words used for metalinguistic variables not yet defined in the above definitions describe their nature. First-order approximation of the algorithm, implemented in the < parsing phase > is as follows:

```
s(0): = '@'; i: = 0; k: = 1;
while p(k) ≠ '@' do
  begin i: = j: = i + 1; s(i): = p(k); k: = k + 1;
    while s(i) .> p(k) do
      begin while s(j-1) ≐ s(j) do j:=j-1;
        s(j): = leftpart (s(j),...,s(i), rule_no);
        case rule_no of
          begin <translator description> end case;
     end .>;
  end of the parsing phase  .
```

Here p(1),...,p(n) is the original sentence; k is the index of the last symbol scanned; s(1),...,s(i) is stack; s(j),...,s(i) is the reducible substring; ! @ is initializing and terminating the process; to any symbol S ε Vt U Vn the relations @ <. S & S .> @ and p(n + 1) = '@' are true.

The function leftpart reduces the metaexpression s(j),...s(i) to the corresponding metavariable and returns the serial number (rule_no) of the syntactical rule

U :: = s(j),...,s(i) .

In fact, the processes of lexicographical analysis and parsing are performed alternatively. The lexical procedure reads a lexicographical unit and delivers a value for it in s(k). In the case of identifiers, strings, and unsigned numbers, it stores a code in stack named data, which is parallel to the stack s.

5. Conclusion

The system TGS-4000 is now being used in the design and implementation of special purpose languages with context-free grammars. The generator allows both interactive and batch processing of language definitions defining both interactive and batch-processing languages.

In the Appendix, we show, through a simple example, the utility of TGS-4000 in producing an interpreter for a language subset of ALGOL-60, and an on-line execution of the produced interpreter. The translator generator is used also to implement FORMAL--an interactive language designed by the author for doing formal algebraic manipulations. This language has a syntax similar to the syntax of BASIC [6], 52 basic symbols and 36 metavariables in its vocabulary, 90 syntax rules, and semantic definitions containing about 100,000 characters. TGS-4000 generates the corresponding translator in three minutes on an RC-4000 computer, running with 32 K core storage and software-implemented virtual storage.

The using of TGS-4000 metalanguage permits an approach more synthetic for the translator's writing, makes possible the creation of developing languages, tailor-made to the up-to-date or future needs of the users.

References

[1] Alexandrov, D. List-Processing System for RC-4000 Computer. Proc. of the 7th May Conference, Sofia, Vol. 1, 1974.

[2] Colmerauer, A. "Total Precedence Relations." Journal ACM, 17, 14-30, 1970.

[3] Feldman, J., and D. Gries. "Translator Writing Systems." Comm. ACM, 11, No.2, 1968.

[4] Hansen, H., ed. ALGOL-6 User's Manual. RCSL: 31-d322, A/S Regnecentralen. Copenhagen, 1974.

[5] Hansen, P. RC-4000 Reference Manual. RCSL: 51-D1, A/S Regnecentralen. Copenhagen, 1969.

[6] Kemeny, J. and T. Kurtz. BASIC Programming. New York, Wiley, 1971.

[7] Lauesen, S., ed. File Processor User's Manual. A/S Regnecentralen. Copenhagen, 1969.

[8] Learner, A., and A. Lim. "A Note on Transforming Context-Free Grammars to Wirth-Weber Precedence Form." Computer Journal, 13, 142-144, 1970.

[9] Naur, P., ed. Revised Report on the Algorithmic Language ALGOL-60. Comm. ACM, 6, 1-17, 1963.

Appendix

Contents of File "Simpledef"

Simple Precedence Structure Programing Language "Simple"

The meaning of the language is explained in terms of an array of variables, called vs (value stack), which has to be understood as being associated with the array s - stack, used in the parsing algorithm. A second set of variables is called ns (name stack). It serves to represent a second value of certain symbols, which can be considered as a name.

Formal definition of the language "simple"

@ syntax:

```
program = block '.'  .
 block = 'begin' body 'end'  .
 body = body1  .
 body1 = declar ';' body1 ! statm_list  .
  declar = 'real' 'identifier'  .
  statm_list = statement ! statm_list ';' statement  .
  statement = variable ':=' expr !
              block  .
   variable = 'identifier'
   expr = expr '+' term ! 'permitted arithmetic expressions'
          expr '-' term !
   term = term '*' factor ! term '/' factor ! factor  .
   factor = variable ! '('expr')' ! 'number'  .
```

@ environment:

```
 begin array vs(1:100);
   integer array ns(1:100);
  write(out,<:'simple' ready <10>: :> ;
  setposition(out,0,0);
```

@ translator:

```
11:  go to exit; 12: ; 13: ; 14: ;
15:  ;
16:  begin ns(j): = data(i); vs(j): = 0 end declaration;
17:  ; 18: ;
19:  begin j1: = vs(j); vs(j1): = vs(i);
      write (out, string dictionary(ns(j1)),<: = :>,vs(i),
        <:<10>::>);
      setpostion (out,0,0);
     end assignment statement;
110:  ;
```

```
111:  begin for i 1:  = j - 1 step -1 until 1 do
         if ns(i1) = data(j) then go_to founded;
         error(<:undeclared variable:>, exit);
         fpunded:  vs(j): = i1;
         end variable;
112:  expr:  vs(j): = vs(j) + vs(i); vs(j): = vs(j) - vs(i);
             vs(j): = -vs(i);
116:  term:  vs(j): = vs(j)*vs(i); vs(j): = vs(j)/vs(i);
119:  factor:  vs(j): = vs(vs(j));  vs(j): = vs(j+1);
               vs(j): = number;

               @ interpreter:

exit:  printtime (<:end of run  .:>);
end  ;

     comment end of file simpledef:
```

Generating and Execution of Translator for the Language "Simple"

```
*  comment lines are prefixed by the character * .
*  Generate an interpreter for the language "simple"!
*  Its definition is in file "simpledef".  No listing!
     simple = tgs4000 simpledef list.no

tgs4000 begin.
   Number of basic symbols = 15
   Number of meta .symbols = 11 + 2
   Syntax-tables for the language *simple* are saved
     in file *ltables*.
   OK, translator for *simple* language is ready.
tgs4000 end.

*  Call the translator for the language "simple" :

*  User's lines are prefixed by a colon.
   simple  ;
"simple" ready
:  begin real a; real b; a:=5;
a  = 5.000
:    a:=b;
a  = 0.000
:   a:=1/a+ 0.001;
a  = 0.201
:     begin real a;  b:=a+314'-2 - a*a
b  = 3.140
:  end  ; a:=5*10 + a + b
a  = 53.341
:  end  .

end of run  . CPU-time used: 0.12 sec.
```

Industrial Manipulators and Robots

Nicolay D. Naplatanoff

1. Introduction

The increasing want of highly qualified workers and the tendency toward a sharp rise in the productivity of labor of the separate worker, intensifies the interest for the use of industrial manipulators and robots. These objects of investigation are quite suitable for the application of the question-answering systems (QASs).

It is our opinion that high-quality dialogue systems of the type in question can be created, but there are also a number of problems concerning their efficient use that need to be clarified.

In this connection, it is necessary to remember that it is not possible to apply the modern means of automation to every "old" technology. That is why the development of "objects," which will require in the process of their development a highly organized control system with artificial intelligence, is quite an urgent and expedient problem.

In this paper we briefly consider problems connected with the creation of industrial manipulators with a simplified mechanical control system on the basis of fluids.

At the Institute of Engineering Cybernetics (IEC) at the Bulgarian Academy of Sciences, the first steps in the synthesis and realization of two cycle manipulators have been undertaken.

2. Manipulators of the Type "Faloma-IEC-01"

The manipulator carries out a cycle of the following motions: vertical upwards, horizontal forwards, supply of one detail, rotation to 90^{o}, horizontal backwards, vertical downwards, releasing of the detail, return to the initial position.

The logical block-scheme provides the following regimes:

a) complete automatic cycle with the capabilities to change the duration of the separate motion,

b) manual control.

The control is realized as a combinational logic network with a program device. The program is recorded on a data tape and is read by a pneumatic reading device. The current impulse is fed by a pneumatic current generator with frequency 2Hz. The logic block is realized by pneumatic membrane elements of the type "Dreloba" (GDR).

The manipulator scheme investigates the turn to fluidic elements of the type "Faloma," designed by the Institute of Engineering Cybernetics.

3. Manipulators of the Type "Faloma-IES-05"

The manipulator has five grippers which act simultaneously. The cycle consists of the following motions: horizontal forwards, catching of five details simultaneously, rotation to 45^{o}, horizontal to the right, slackening of the details, horizontal backwards, horizontal to the left.

The working regimes are three:

a) single automatic cycle,

b) continuous automatic cycle return, and

c) manual control.

The following block systems are stipulated:

a) stop--stopping at closed position

b) zero--automatic return to the initial position.

The control system is a synchronous automaton with a period frequency 0,5Hz and consists of the following main blocks:

a) supplying,

b) a generating device for time impulses,

c) memory block,

d) amplifier, and

e) servomechanism.

The system is organized by fluidic logic modules of the type "Faloma" designed at the IEC. The total number of elements is 149. Forty-nine of them are pneumatic, fluidic, logic elements, while the other 100 are interfaces and connective elements.

4. Development Tendencies

The electronics give great possibilities for the realization of all main blocks with the exception of servo-organs.

However, the using of the pneumatic and hydraulic servo-mechanisms makes imperative the introducing of a second kind of energy in the manipulator system and requires the necessary electrofluidic transformers and amplifiers.

This complication is the reason for designing entirely fluidic systems using fluidic logic modules and interfaces.

A comparative evaluation is represented in the table given below.

Table 1.

Number	Type of the Manipulator (Robot)	Character--"nature" of the blocks
1.	Manipulator with a cycle of action (nonintelligent robot or industrial robot)	All blocks are pneumatic
2.	Manipulators with cyclic action and possibilities for readjustment at the time of their action (intelligent robot)	Input block-computer Transforming block-electro-pneumatic Servoblock-pneumatic
3.	Robots with analogous action with an adaptive behavior; recognizing, searching, with space action (superintelligent robot)	---

These tendencies determine for us an interesting and necessary scientific trend, namely: synthesis of "intelligent" systems of manipulator and robot control, including also dialogue systems of the type "questions-answers" in other modifiable "question-answering" systems.

In this direction, we look forward to coworking with IIASA on the examined problems.

APPENDIX 1

Some Comments on AI Research Coinformation

D. Dubrovsky

1. Introduction

The participants of the workshop would like to point out that QAS development is only one part of AI research being conducted in the different countries.

There are important and interesting results also in AI research areas connected with psychology, linguistics, neurocybernetics, automatic programing, robotics, symbol manipulation, pedagogics, etc. (For example, see the following section.) It is very desirable that the different forms of AI research being conducted by IIASA member countries should be coinformed to some extent by IIASA research policy.

Coinformation in the AI field could become a framework for the system analysis of AI research as a complex "R & D system." In that case, IISA could be a point of linkage between national research groups cooperating on different topics of an AI field, while the work could be done by scientists of national organizations who were interested in it.

Now, the workshop initiates that work and the participants hope that there will be fruitful results in that direction in the near future.

2. Appendix to AI Research Coinformation

2.1 On Social Prelevant Applications of AI

QAS's are an application of AI with clear general relevance for social systems as far as storage and retrieval operations in large data bases are concerned.

There is a great tendency to consider that QAS and robotics are the only two socially relevant applications of AI. It is important to refute this opinion.

One of the source fields of semantic information processing was the Computer Aided Instruction (CAI) Project from ARPA at Bolt, Beranek and Newman (BBN) in Cambridge, Massachusetts. A

second pedagogical application of AI was the research work of Professor Seymour Pappert, AI-Laboratory, MIT, Cambridge, on "Teaching Children to be Mathematicians and not Teaching Them about Mathematics." A third application on pedagogics was suggested by Professor Quillian (at both BBN and AI-Lab, MIT) in his paper, "The Computerized Piaget or Psychology from the Point of View of a Radical Computerist." In this paper, the natural mathematical ability and performance of little children (pattern recognition, generative grammar) is studied in order to teach children at school another kind of mathematics even before they begin to learn to read. The work of Messrs. Chomsky, Linguistics Department, MIT, Cambridge, on formal models of language acquisition by children is also related to this field.

A further application in this sense is to simulate the cognitive development stages of children's brains in order to provide children from underdeveloped countries or from underprivileged strata with an early compensatory training in mathematics and linguistics, and thereby reduce the cognitive deprivation effects on large parts of humanity.

A project in this area would allow IIASA to obtain financial support from UNESCO in Paris and other UN agencies involved in a world-wide effort to combat illiteracy and cognitive deprivation among more than half of the world population. A theoretical project at the Paedagogische Hochschule in West Berlin is actually engaged in this subject, (conducted by Prof. Alex Baumgartner and myself) with special regard to a mathematical curriculum for preelementary schools. In this project, the simulation of affective development stages of children (as described by Rene Spitz) is linked interactively with the simulation of cognitive development stages (as described by Piaget and others).

An experimental project is being conducted at an elementary special school for children of Spanish immigrant workers at Frankfurt-Main, FRG, together with psychologist Professor Leber, Department of Special Paedagogics (for deviate behavior) at Frankfurt University. The results of this project could be utilized for a project proposal to be sent to the involved UN agencies.

An important aspect of social behavior between nations and other human collectives is the problem of peace and conflict. Various important institutes in the world are engaged in empirical data collecting and the interpretation of such problems as the arms race, racial or national hatred, social struggle, counter-insurgency, military-industrial complexes, raw material embargoes, etc.

The Max-Planck-Institute at Starnberg, FRG, the Nobel Institute of Peace Research at Stockholm, the Governmental US Peace Research Agency, the Institute of Strategic Studies in the UK are among others involved in these problems.

Professor Hiel de Soola Pool and Professor Nazli Chouckri both from MIT, Cambridge, and Professor Carl W. Deutsch, Harvard

University, Cambridge, were engaged for some years in formalizing nongaming approaches for a computer simulation of international conflicts. Artificial intelligence approaches for the representation of the subjective attutdes of politicians and governmental and nongovernmental organizations in international relations were developed among others by Professor Matthew Bonham, American University, Washington; Professor Howard Alker, MIT, Cambridge; and myself, at the Frankfurt University. I have made an arrangement with Professor Bonham to link our simulation algorithms to a complex system in order to simulate historical processes as sequences of discrete political events synchronized with socio-economic semicontinuous processes.

Dr. Raul Espejo, IIASA Project on Large Organizations, worked together with Stafford Beer (see his book: Designing Freedom), on the theory of Maturana (Urbana) and Varela (Stanford) on self-repairing (autopoietic) automatons, applicable to the behavior of social systems.

This could be improved as the beginning of an IIASA project on Peace and Conflict Simulation Modeling with the financial support of some important international or national agencies and foundations. It is presumable that such a project could provide for IIASA an international resonance as the project "Limits to Growth" provided for the Club of Rome, and have a broad public relations effect for the Laxenburg organization.

APPENDIX 2

A Word of Caution

S.D. Isard

Abstract

In planning a natural language information retrieval system, it is tempting to think in terms of two relatively independent components: one which knows about language and is able to translate sentences, regardless of their content, into some internal semantic representation, and another which can manipulate semantic representations in order to produce answers to questions. However, I feel that one must give serious consideration to the fact that in successful artificial intelligence projects of the past few years, not only have language and reasoning components been blended together, but the design of the whole system has been heavily influenced by the particular subject matter with which it is intended to deal. The main reason for this would appear to be that knowledge of language and reasoning alone are not sufficient to determine either (a) what constitutes a useful answer to a given question, or (b) how to look for it in a sensible way.

Another important point is that neither the first order, predicate calculus nor the other forms of semantic representation so far proposed offer truly satisfactory ways of representing the information conveyed in natural language by tense, aspect, subjunctive mood, modal verbs, intentional adjectives, and a variety of other sorts of words and constructions. Each artificial intelligence system tends to cope with such phenomena in a fashion appropriate to the particular questions with which it is designed to deal, or to otherwise ignore them whenever possible.

"Why" and "how" questions constitute an especially tricky area where neither predicate calculus nor semantic nets offer any straightforward hints on how to proceed, and where the problem of what will serve as an answer is dependent both on the subject matter under discussion and the state of knowledge of the questioner. It is interesting to compare the ways in which the programs of Winograd, Stansfield, and Scragg adopt different tactics appropriate to their different settings in trying to answer such questions.

The moral, I think, is that one will inevitably fail to capture the full subtlety of natural language, and that practical success depends largely on making the right compromises. In

particular, the nature of these compromises should be determined primarily by the sort of information to be stored and the kinds of questions you want to ask and answers you want to receive. If, for instance, you are dealing with a static body of facts, such as a table of atomic weights, you may be able to largely ignore the semantics of tenses, aspects, and modal verbs; but if you want to discuss the effects of various processes on a situation which change over time, such as the world economy, then you cannot. It is these considerations which should determine the choice of semantic representation and the way in which natural language sentences are translated into it.

Conference on Artificial Intelligence: Question-Answering Systems

AGENDA

June 23-25, 1975

Wodak Conference Room
Schloss Laxenburg

Chairman: Professor F. Klix

Monday, June 23

9:30	QAS Purposes and General Structure	
	Introduction to the Conference	F. Klix
	DILOS - Dialog System for Information Retrieval, Computation and Logical Inference	D. Pospelov
10:45	Some Comments on Efficient Question-Answering Systems	H. Nishino
	Language Analysis and Language Representation for QAS	
	Partitioned Semantic Networks for Question-Answering Systems	G. Hendrix
14:00	The Choice of Semantic Representation in a QAS	J. Simon
	Analysis of Japanese Sentences by Using Semantic and Contextual Information	M. Nagao

Tuesday, June 24

9:00	Parsing in QAS	W. Paxton
	Input Processing in a German Language Question-Answering System	E. Lehmann

10:45	Representing and Deducing Information	
	A Formal Framework for a Unitary Approach to the Theory of Problem Solving	M. Somalvico
	Logic and Interpreters	E. Pagello
13:30	Learning Procedures and Simulation Aspects	
	Artificial Learning Systems and QAS	A. Andrew
	A Computer Interview Procedure Which Reconstructs Generative Semantic Structures of Human Beings Using Modal Sets	S. Klaczko-Ryndziun
15:45	Information Retrieval	
	Cognitive Information Retrieval by Goal-Oriented Languages	G. Gini
	Statistical and Information Theoretical Approach to the Retrieval Systems	A. Benczur

Wednesday, June 25

9:00	Implementation of QAS and Programming Languages	
	An Experimental Environment for the Implementation of Question-Answering Systems	G. Nees
	PLATON - A New Programming Language for Natural Language Analysis	M. Nagao
10:45	The TGS-4000 Translator - Generator System	D. Alexandrov
13:00	Special Application Aspects	
	Some Problems of Industrial Manipulators	N. Naplatanoff

LIST OF PARTICIPANTS

Chairman

Prof. Dr. F. Klix
Academy of Sciences of the GDR
Zentralinstitut für Kybernetik und Informationsprozesse
Rudower Chaussee 5
1199 Berlin

Austria

W. Rhomberg
Technische Hochschule Wien
Siebenbrunnenplatz 6/9
1050 Wien

E. Stöhr
Hochschule für Bildungswissenschaften
Wörthersee Süduferstrasse
Klagenfurt

Bulgaria

Nicolay D. Naplatanoff
Institute of Engineering Cybernetics at the Bulgarian Academy of Sciences
"Geo Milev" Block IV
Sofia 13

Federal Republic of Germany

Dr. Georg Nees
Siemens A.G.
E-549
Freyeslebenstrasse
852 Erlangen

Salomon Klaczko-Ryndziun
Seminar für Kybernetick
Swiss Federal Institute of Technology - ETH
CH-8006 Zurich, Switzerland

France

Prof. J.C. Simon
Universite de Paris
Institut de Programmation
Place Jussieu
75005 Paris

German Democratic Republic

Prof. Dr. F. Klix (Chairman)

Hungary

Dr. Jozsef Pergel
Computer and Automation Institute
Hungarian Academy of Sciences
XI Kende utca. 13-17
1502 Budapest

Andras Benczur
(address as above)

L. Kiraly
(address as above)

Italy

Prof. Marco Somalvico
Milan Polytechnic
Artificial Intelligence Project
MP-AI Project
Politecnico di Milano
Piazza Leonardo da Vinci 32
20133 Milano

Dr. Giuseppina Gini
Milan Polytechnic
Artificial Intelligence
Project
MP-AI Project
Politecnico di Milano
Piazza Leonardo da Vinci 32
20133 Milano

Dr. Maria Gini
(address as above)

Japan

Prof. Makoto Nagao
Kyoto University
Department of Electrical
Engineering
Kyoto

Dr. Hiroji Nishino
Electrotechnical Laboratory
2-6-1, Nagato-cho
Chiyoda-ku
Tokyo

Poland

Dr. Jozef Maronski
Institute for Organization
Management and Control Science
K.R.N. 55
00-818 Warsaw

United Kingdom

Dr. A.M. Andrew
University of Reading
Department of Engineering
and Cybernetics
Whiteknights
Reading

Stephen D. Isard
The University of Sussex
Laboratory of Experimental
Psychology
Brighton

U.S.A.

Gary G. Hendrix
Stanford Research Institute
Artificial Intelligence Center
333 Ravenswood
Menlo Prk, California 94025

William H. Paxton
(address as above)

U.S.S.R.

V.M. Briabrin
Computing Center of the
Academy of Sciences
C/-State Committee for
Science and Technology
11 Gorky Street
Moscow

D.A. Pospelov
(address as above)

Dr. G. Fomin
(address as above)

Dr. Kuzin
(address as above)

Dr. D. Dubrovsky
(address as above)