A LANGUAGE BASED PROBLEM-SOLVER

S. Ramani
Computer Group
Tata Institute of Fundamental Research,
Bombay 5, India

Abstract

Language learning and language use play central roles in problem-solving. It is argued here that a rigidly built-in language will not serve the needs of problem-solving. To use language as significantly as in human problem-solving it is necessary to design language processors specially suited to the task. The procesBoi should use language as a medium for describing situations so that their similarities are recognized. Recognition of such similarity would enable the system to respond to new situations with forms of response known to be appropriate in similar, familiar situations.

The design and implementation of a problem-solving system based on this principle are described. This system exhibits some capability to learn and use a language, and to solve problems.

1• Introduction

The state of the art in problem-solving, constituting the background to the work reported here, is based on a number of concepts and techniques, including the following:

- use of heuristics, in addition to algorithms, in programs (1,2);
- provision for pursuing a hierarchically organized set of goals and subgoals, by the use of recursive routines and push-down stacks (3);
- use of a scheme of representation for mapping problem-situations into data structures (4);
- uBe of an input language resembling a natural language (5);
- use of sets of strings in a language as representations, to provide for generality (6).

The use of an adequate language for input and output, and for the internal representation of relevant information as well, is very attractive as a possible solution to the problem of generality. Similarly, one is also attracted by the possibility of realizing a system which will accept directions (complete or partial) for solving problems in some language resembling a natural language (7). Obviously, such a system will be open-ended in a very significant sense of the term,

We start with these concepts and argue further that a general problem-solver should possess a general language-using capability. This capability should uniformly provide for input/output communication, for the internal representation of the totality of problem situations constituting the problem world, and for the communication and representation of problem-solving techniques, including algorithms and heuristics.

The approach reported here is further characterized by three features:

(a) the requirement that the system be open-ended, in regard to the language used; it should be possible, at any stage, to add incrementally to the repertoire of the system,

(b) the requirement that the system be similarly open-ended in regard to the class of questions answered, or the class of problems solved, and

(c) the requirement that the capability of the system be enhanced in a simple and natural manner. Specifically, it is required that the system should enhance its capabilities automatically when a worked-out example is supplied.

The first two requirements raise major questions regarding the nature of the language processor to be used. The following sections describe these questions and provide an answer for them, based on the concept of analogy directed behaviour. This concept is then extended to provide a basis for meeting the third requirement cited above - that the problem-solving should itself be example driven. It is argued that such a problem-solver will cope with an open-ended problem world, in a sense, creatively.

2. The Basic Issues

One needs a theory of language to design any language-using system, a theory which covers syntax, semantics and pragmatics.

Designers have been quick to borrow from linguistics whatever has been available in the way of syntax and it has been suggested that a formal grammar of a language, such as a transformational generative grammar, could be used in conjunction with a syntax-directed parser to perform an analysis of natural language sentences (8, 9).

Another frequently suggested possibility is the use of a semantic processor which will map sentences of a natural language into strings in a formal language (say, a first order predicate calculus) using information obtained from syntax analysis, whenever necessary (10).

There are difficulties in adopting syntax directed parsing methods for use in a problem-solver of the type visualized in the previous section. For example, it is algorithmically impossible to produce a generative grammar for every given set of sentences (see(II) for a formal analysis). Making incremental additions to a generative grammar to extend the language is an equally demanding task, sharing many features with the task of program derelopment and debugging.

These limitations on automatic acquisition are severe in view of the enormous effort required to create grammars manually. Machine-usable grammars for chunks of natural language large enough to be useful are not yet available.

A similar, if not worse, situation prevails in the area of semantics of useful chunks of natural language. The mapping of an input sentence into a formula in some logical calculus requires an algorithm. There is no uniform procedure for arriving at such an algorithm which maps sentences in a given language into formulas in a chosen logical calculus. Any such algorithm will require considerable modification and debugging when the set of sentences it has to handle is enlarged in any significant manner.

3. The Case for Analogy

Is there any alternative, then, to the design of a language processor? Is it possible to avoid syntax-directed analysis, followed by a mapping of the input sentences into strings in a logical calculus ?

The alternative explored in the work reported here is based on the notion that human language behaviour is analogy directed. One's response to a verbal stimulus is generally determined by the similarity it has to a familiar stimulus. The response to the new stimulus resembles the response to the familiar one in certain ways. One is often taught the answer to a new kind of stimulus by being given an example.

To model this type of behaviour, one has to look for a programmable process which will carry out this analogizing and answer new questions on the basis of a set of question-answer pairs used as examples. Each question-answer pair that is supplied as an example would be accompanied by a set of sentences (the context). The processor would obviously use a store for factual information (say, in the form of sentences) which describes the class of situations about which it can be questioned. This store forms a kind of data-base. When presented with a new question, the processor creates a reply to it by recognizing its similarity to a question given earlier as an example. From the example, the processor would identify the form of the informative sentences relevant to the question. Then the processor would identify such sentences in the data-base and create a reply.

Proceeding further, one could add to the system being visualized to enable it to handle inputs other than simple questions. Some inputs may be informative sentences which have to be analyzed in order to extract the information they bring to the system. The extracted information is then incorporated into the data base.

Finally, one could imbed a language processor of this sort into a problem-solver.

The following sections show how these tasks are tackled. But, it is useful here to summarize the motivations for investigating analogy as a principle directing natural language behaviour:

(a) Analogy appears attractive as one technique by which a language may be acquired. An analogy directed processor can be made to tackle a new class of questions or statements by supplying it with an example. The process is incremental.

(b) An analogy directed processor is not limited to any specific language. Depending upon the examples supplied, it tackles different languages.

(c) Recognizing a familiar structure underlying a new situation, and responding to it accordingly, appears to be an essential component of all intelligent behaviour. A language processor used by a problem-solver will provide for such recognition at each step of problem-solving. For instance, a simple question that arises at some step of problem-solving will be recognized by its familiar structure, even if it is new in certain respects, and be answered appropriately. In other words, a language processor which can handle new sentences by analogy provides the basis for a problem-solver which can handle new problems by analogy.

A detailed treatment of the role of language learning in intelligent behavior may be found in (12). In this paper, Narasimhan argues that the use of a rigidly determined language denies a problem-solver the ability to cope with new classes of situations, (in McCarthy's words (7) a problem-solver using such a limited language cannot be 'told' about the new classes of situations, and therefore, it cannot also learn their relevant features by itself).

(d) The approach prescribes a simpler structure for a language processor. Seemingly arbitrary compartmentalization of the process is avoided. Consider, for example, the syntax directed processors which fail to use semantic or contextual clues in syntactic disambiguation. Analogy directed processors are not divided into such non-communicating sub-systems.

A detailed critique of language behavior models incorporating generative grammars has been written by Hockett (13). He also presents arguments for recognizing the important role of analogy in human language behavior.

4. The Proposed Model for a Language Processor

The considerations described in the previous sections lead to a model for a language processor having the following features:

(a) The processor deals with situations, a situation being a stimulus (or input) in its context.

The stimulus is a string of symbols (e.g., a question in a natural language, the words being treated as symbols here). The context is a set of strings of symbols (e.g., a set of sentences describing some aspect of the world that is being questioned).

(b) The response of the processor to any situation is based on pattern setting examples (or paradigms) given to it earlier.

A paradigm consists of a context (a set ol strings of symbols), an input (a string of symbols) and a response (or output). The response is a sequence of strings of symbols (e.g., a sequence of sentences in a natural language).

(c) Paradigms are stored in a form that highlights their structure, de-emphasizing the particulars of the situation involved. A paradigm stored in this form, called a schema, can be used by the processor to create responses to new situations by a process of extrapolation.

(d) The processor has access to a store for sentential strings, constituting a data-base (see Fig. l).

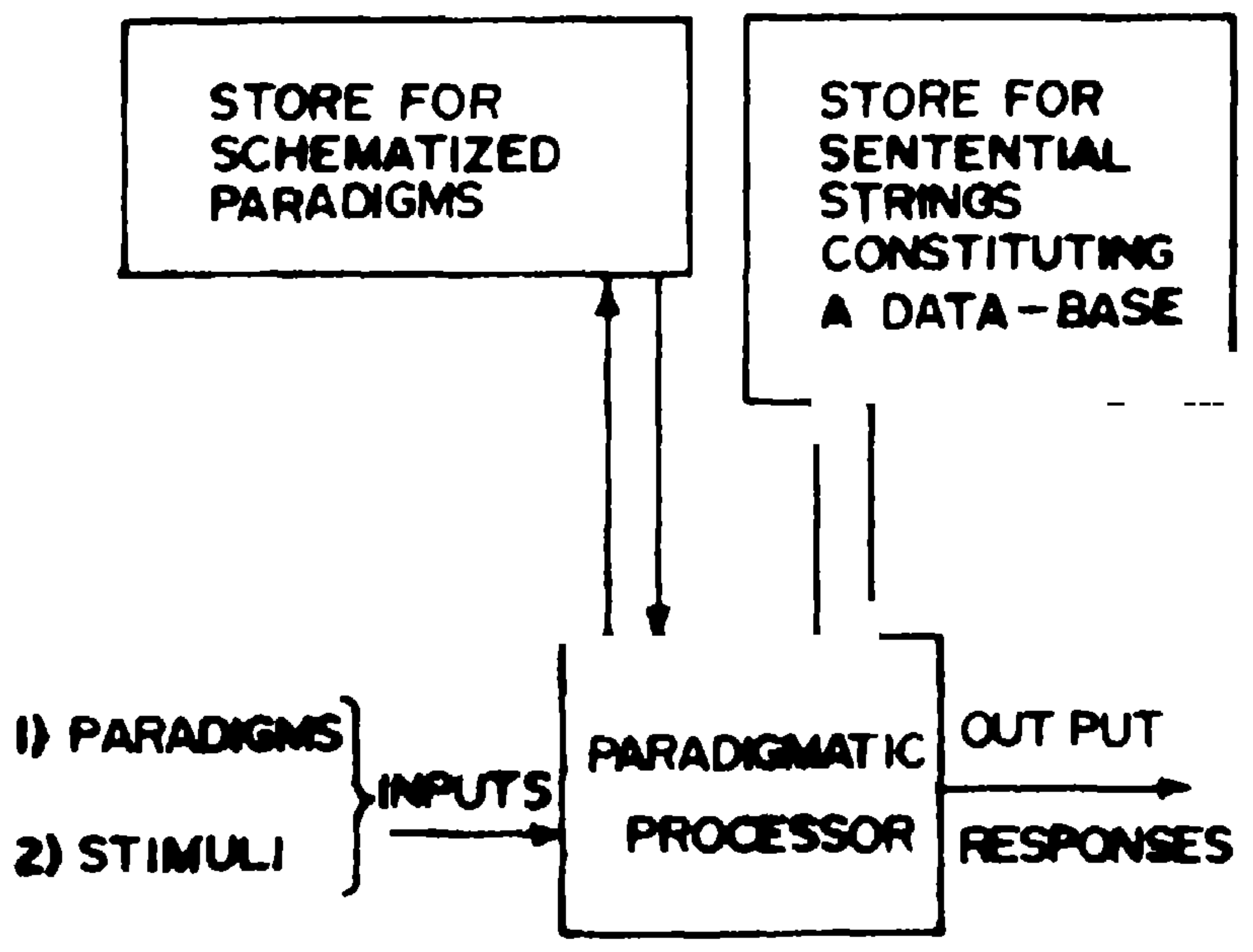

STORE FOR SCHEMATIZED PARADIGMS
STORE FOR SENTENTIAL STRINGS CONSTITUTING A DATA-BASE
1) PARADIGMS
2) STIMULI
INPUTS
PARADIGMATIC PROCESSOR
OUT PUT
RESPONSES

FIG. 1. THE ELEMENTS OF A PARADIGMATIC PROCESSOR.

Inputs to the processor are of two kinds. The first kind of inputs are paradigms labelled as such. The other kind of inputs are sentential strings (e.g., questions, commands or informational sentences in a natural language The processor schematizes and stores the paradigms for later use. The processor responds to sentential stimuli by extrapolating from an appropriate schematized paradigm, if such a paradigm is available. In any case, the sentential input is added to the data-base, for possible functioning as a contextual string for later inputs.

(e) The suitability of a (schematized) paradigm for determining the creation of a response to a given stimulus is based on two factors:

i) structural similarity between the current stimulus and the stimulus component (or input component) of the paradigm,

ii) the presence or absence in the data-base of a set of strings having specified structures (the processor obtains the specification for these structures by extrapolating from the schematized paradigm being tested for suitability).

(f) Assume a suitable schema is found for determining the response in a given situation. The processor computes a response having a structure determined by the schema. Elements of the response (e.g.. words, numbers and phrases) may be obtained by the processor from the sentential store and incorporated into the structure as specified by the schema.

Obviously, the processes referred to in paragraphs (c), (d), (e) and (f) above have to be described in detail. This is done in the following sections. Before proceeding further, it is, however, necessary to describe an important feature of the processor which provides for a problem-solving capability. The same facility enables the processor to respond to a stimulus which has a complex structure, with phrases nested inside other phrases at several levels,

(g) The extrapolation procedure referred to in paragraph (f) determines a series of strings called 'the response'. In the case of certain classes of situations, the response may simply be printed out (e. g. , a simple question is given a factual answer, without involving nested computation).

In the general case, however, the extrapolated 'response' may function as a sequence of internal stimuli. In other words, in response to a given stimulus, the processor presents itself with a sequence of internal stimuli. The processor has appropriate provision to halt the main sequence of computation at one level to perform necessary subcomputations at a lower level, as necessitated by the internal stimuli. The processor can, in this manner, respond to internal stimuli precisely as it rerponds to an external one, choosing suitable schema to determine the response. Generally, the nesting extends only a finite number of levels and the process ultimately terminates.

The processor prints out each string in 'the response' at each level when it arises as an internal stimulus. Many such strings do not require any further processing, as the available schemata do not provide for it. Any such string just gets printed out and the processor moves on to the next internal stimulus. The complete print-out is the true record of responses at all the levels.

The machinery for nested computation of responses to a hierarchically organized set of stimuli is basically the GPS (3) machinery, involving push-down stacks, etc.

5. <u>The Nature of the Paradigms Used</u>

A simple paradigm is:

Context:	The weight of the ship is 2000 tons
Input :	What is the weight of the ship?
Output :	2000 tons

Given the new situation,

Context :	The height of the boy is four feet
Input :	What is the height of the boy?

Extrapolation at one level yields the answer: "Four feet".

On the other hand, a more complex paradigm would be something like this:

Context:	The density of lead is 8 gm per cc
	The volume of a sphere is 0.75 x PI x radius **3
	The radius of a sphere is 0. 5 x the diameter
Input :	Find the weight of a lead sphere 6 cm in diameter
Output :	1. Find the volume of the sphere

2. Multiply the volume by the density
3. That is the weight of the sphere

Obviously, the three strings in the 'output' are internal stimuli which trigger off internal activity to create the final output.

The choice of these forms for paradigms reflects certain beliefs regarding human verbal behavior. One belief is that the presenting of a paradigmatic example of the desired behavior is most valuable when it is clearly separated from the preceding and succeeding activity. The paradigm should be well segmented from the background activity and be labelled as such, Secondly, when question-answering or problem-solving is taught, it is very useful, if not imperative, to draw attention to the relevant facts - the context.

6. The Paradigmatic Processor

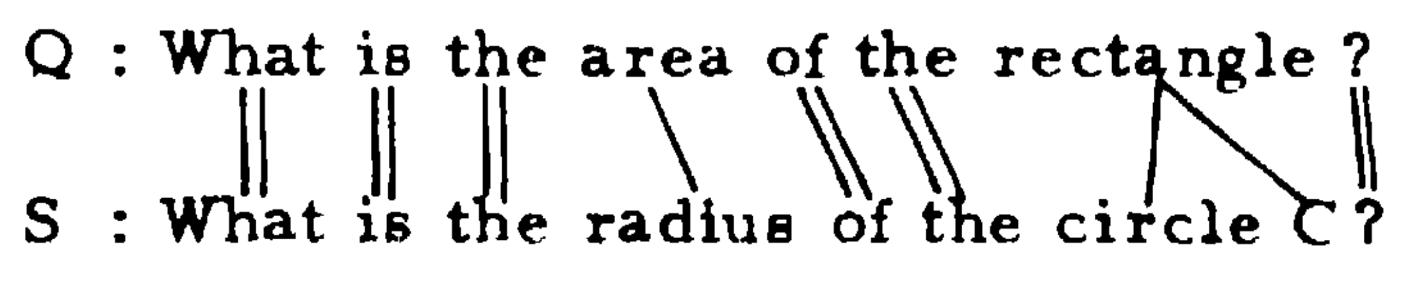
6. 1 Schematization of Paradigms

Consider a paradigm consisting of strings of words and symbols, these strings being statements, questions or commands occurring in the paradigm. The paradigm is divided into a contextual component, consisting of the strings C1, C2, C3, ..., C_n, an 'input' component consisting of one string Q and an 'output' component consisting of the strings A1, A2, A3, ...,A_n.

Context:		Input:	Output:
	C_1	Q	A_1
	C_2		A_2
	C_3		A_3
	.		.
	.		.
	.		.
	C_n		A_m

Let the ordered set of strings $C_1, C_2, C_3, \ldots, C_n$ be named $\bar{C}$ and the ordered set of strings $A_1, A_2, A_3, \ldots, A_m$ be named $\bar{A}$. A paradigm p is then a triple $[\bar{C}, Q, \bar{A}]$. Given a set of paradigms of this form, how does one extrapolate from them to compute the appropriate response to a new stimulus?

As mentioned earlier, paradigms supplied to the processor are 'schematized' and stored. The schematization serves essentially to assign a structure to the paradigm, facilitating extrapolation. Any input stimulus S, a string of words and symbols, triggers off a search activity in the processor. The search is for the 'schema' of a paradigm p [C,Q,A] which has an input component, Q, maximally resembling S. By a linear mapping of symbols of S onto matching symbols in Q, substrings of S are placed in one-to-one correspondence with substrings of Q (illustrated below).


Q : What is the area of the rectangle ?
S : What is the radius of the circle C ?

Substrings such as 'the weight of the ship[1], che radius of the sphere', etc., are treated as integral units whenever necessary. This follows the conception of a sentence as a string of units in specified order, some units being words or single symbols, while others are phrases defined by sets of recursively applicable rules (Bobrow(5) and Weizenbaum (14) describe approaches which suggested this development. Leavenworth (15) and Woods (16) are also relevant papers).

Consider a paradigm p [C,Q,A] . To assign a structure to this paradigm, we look for maximal substrings which occur in one or more places in one Bet of Btrings as well as in one or more places in another set of strings. For example, the maximal substrings common to C and Q below are (the cost of, is, 4, pencils)

C : The cost of one pen is 4 rupees
The cost of two pencils is 1 rupee
Q : What is the cost of 4 pens and 3 pencils

Let S_{cq} be the set of maximal substrings common to Q on one hand and to C on the other. S_{aq} is the set similarly defined with respect to "K and Q. The set of maximal substrings, each one of which occurs somewhere in A as well as somewhere in C, is called S_{ca} • We define two new sets

$$S_q = S_{cq} \cup S_{aq} \text{ and } S_v = S_{ca} - S_q .$$

The schematization of the paradigm p [C, Q. A] is performed by substituting uniformly for each substring in the set S a corresponding formal variable, at each of its occurrences in p. This requires the generation of a set of distinct formal variables Sf , such that $|Sf| = |S_v|$ • The schematization gives us the schema K $[S_q, S_{cf}, C, Q, A']$ where C' is obtained from C and A' from A by the substitution process described above. S_{cf} is a new set of variables obtained by substituting in Sca every occurrence of a variable in S_v by the corresponding formal variable from Sf . This process is illustrated by the following example:

Paradigm

71 : G is 31. 5 feet per second squared
The ball is thrown vertically upward at 31 feet per second
Q : Find how high the ball will go
A : V = 31
G= 31.5
Find (V ** 2) / (2 * G)
That gives the height reached by the ball

S_q = { the ball }

S_{ca}= { 31, G, 31.5, the ball }

S_v = { 31. G, 31.5 }

S_f = { 00001, 00002, 00003 }

S_{cf} = " 00001, 00002, 00003, the ball }

Schema

00002 is 00003 feet per second squared
The ball is thrown vertically upward at 00001 feet per second
Q Find how high the ball will go
V= 00001
00002 = 00003
Find (V ** 2) / (2 * 00002)
That gives the height reached by the ball

6. 2 Extrapolation from Schemata

Given a stimulus, S, one may compare it with the input component of a schema K [Sq , S_{cf} , C^1 , Q, A'] . The comparison performed by the processor consists of two steps: (a) linear mapping of S onto Q, and (b) evaluation of the match. Consider two strings of words and symbols

$$Y_1 \quad Y_2 \quad Y_3 \quad \cdots \quad Y_n$$
$$Z_1 \quad Z_2 \quad Z_3 \quad \cdots \quad Z_m$$

The linear mapping of Y1 *Y2* Y3 ... Y_n onto $Z_1 Z_2$ Z_3 ...Z_m is performed as follows:

1. The mapping should proceed from Y_1 to Y_n.
2. If it is otherwise permissible to map Yj onto either *Zj* or Zk, j < k, Yi should be mapped onto Zj .
3. If Yj is mapped onto Z;, no mapping of the form Yk - *Z1* is permitted if k> i while 1 < j.
4. Pursuant to the conditions stated above, Yj may be mapped onto *Z:* if Yj and Z J are identical.
5. Pursuant to the first three conditions, Y^ may be mapped onto Z: irrespective of their being identical, if they are both numbers, neither being a formal variable.

The mapping will not be complete after the exhaustive application of the five rules listed above. Considering the example referred to in Section 6. 1, the part of the mapping performed according to these rules is shown here:

```
Q : What is the area of the rectangle ?
     ||   ||  ||         \\  \\         ||
S : What is the radius of the circle C ?
```

However, it is possible at this stage to estimate the degree to which Q resembles S. A useful index of the match is found as follows:

1. Assign to every element in Q, but not in SQ , the weight 1.
2. Assign to every element in S_q the weight 0.25.
3. Make a weighted count of all elements of S successfully mapped onto some elements of Q at this stage. Let this be R1 .
4. Let a weighted count of all elements in Q be R_2.
5. R_1 / R_2 is the match index.

By using this match criterion, the schemata most suitable to the task at hand are selected and tried one by one for applicability. For instance, if the task is to create a response to

'Find how high the stone will go'

the paradigm listed in Section 6.1 could be selected.

How does one use the paradigm in the new situation? The linear mapping of S onto Q creates a one-to-one correspondence between elements which occur in S with their substitute elements in Q. The mapping can now be completed by an interpolation process (see (17) for a flow-chart of this process, as well as for details in general. In certain places, however, (17) differs in formulation).

On completion of the mapping, a one-to-one correspondence can be set up between strings in S_q which occur in Q and their substitutes, S' , which occur in S. For instance, in the above example, the single member of S_q , 'the ball^1, can be put in correspondence with 'the stone' in S. Uniformly substituting members of S_q occurring in C of the schema by corresponding members of Sq' , a modified context C" is created. In the above example, T" consists of two strings:

00002 is 00003 feet per second squared

The stone is thrown vertically upward at 00001 feet per second squared

The modified context C" shows the structure of sentences to be identified in the sentential store, if the chosen schema is to be useful in creating a response. The processor searches this store to locate sentences which match the strings of C " , using the linear mapping technique described above. If matching sentences are not found, the processor attempts to use another paradigm. If matching sentences are found, the mapping technique described identifies substrings which occur in the place of the formal variables Sf. A one-to-one correspondence is created between elements of Sf and their substitutes, which constitute the set

At this stage, the processor creates th-e response by uniformly substituting elements of Sf occurring in X' by the corresponding elements of the set S_u.

If no suitable schema is found for a given stimulus S, it is treated as informative input to be directly sent to the sentential store. On the other hand, if a schema is found, the computed response is output string by string. After each string is output in this manner, it is re-presented to the processor, as an internal stimulus. For instance, if the computed response has the string 'V*41 ' in it, it will be re-presented as an internal stimulus. Failing to find a suitable schematized paradigm to guide the response to this new stimulus, the processor will file away 'V*41' in the sentential store for later use.

If the command

•Find (V ** 2) / (2 * G)^1

becomes an internal stimulus, the processor could find a suitable schema to guide it in performing this sub-task.

7. Design and Implementation of the Problem-solver

The simple examples presented in Section 6 show how the paradigmatic creation of a direct, simple response is possible at a very lew level of operation. A stimulus of any complexity results in internal stimuli, and a whole hierarchy of schemata is brought into play. This is true not only in numerical problem-solving, but also in the interpretation of complex sentences, which is a kind of 'problem-solving' in itself.

Fig. 2 shows the problem-solver which consists of the rudimentary processor shown in Fig. 1 along with push-down stores necessary

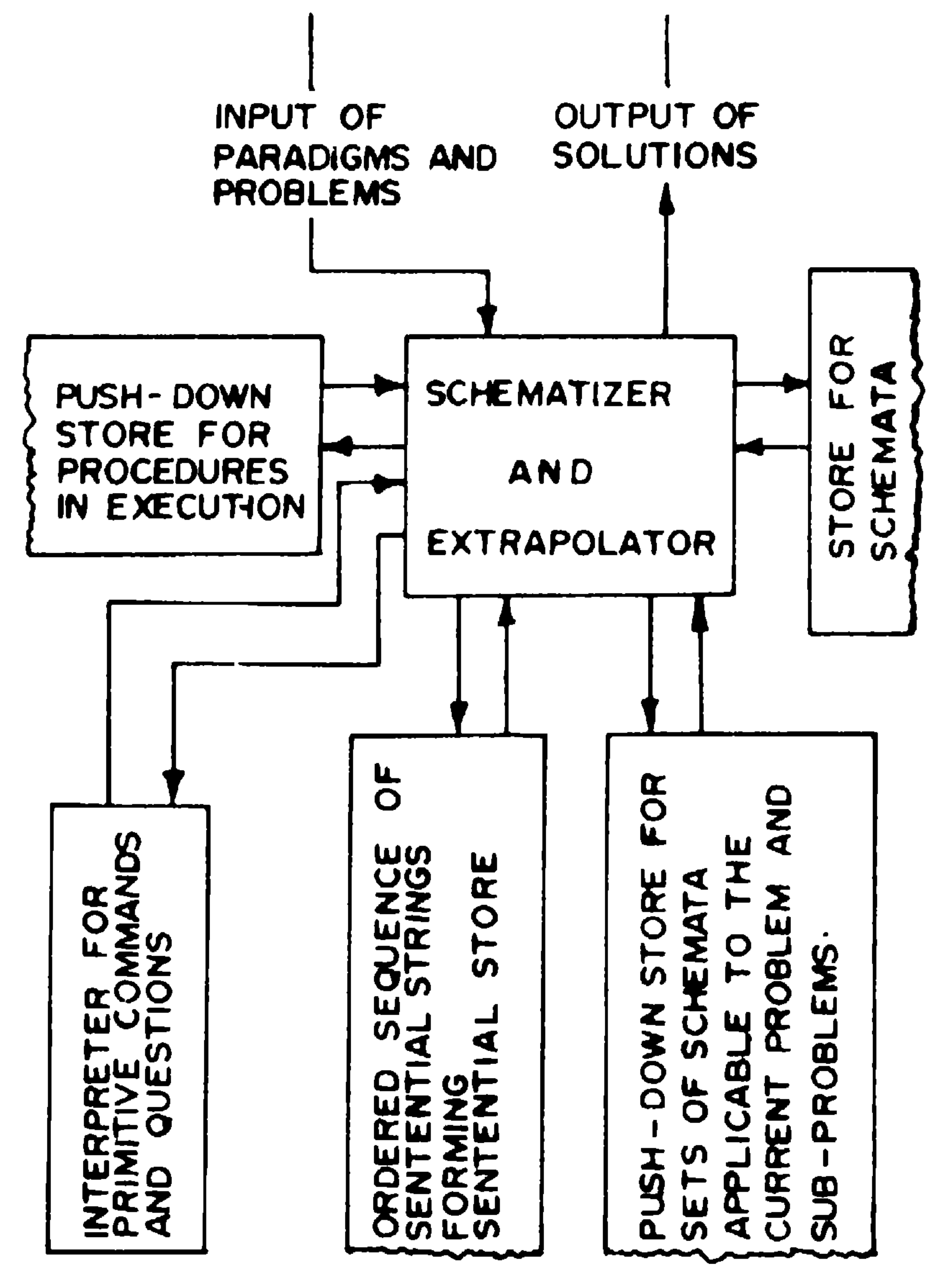

INPUT OF PARADIGMS AND PROBLEMS
OUTPUT OF SOLUTIONS
PUSH-DOWN STORE FOR PROCEDURES IN EXECUTION
SCHEMATIZER AND EXTRAPOLATOR
STORE FOR SCHEMATA
INTERPRETER FOR PRIMITIVE COMMANDS AND QUESTIONS
ORDERED SEQUENCE OF SENTENTIAL STRINGS FORMING SENTENTIAL STORE
PUSH-DOWN STORE FOR SETS OF SCHEMATA APPLICABLE TO THE CURRENT PROBLEM AND SUB-PROBLEMS.

FIG. 2. SCHEMATIC OF PROBLEM-SOLVER.

for the recursive application of schemata. The interpreter is a simple speedup device which responds to frequent forms of internal stimuli. Though whatever it does can, in principle, be performed in the paradigmatic mode, it increases efficiency and reduces storage requiremerts by interpreting a set of simple stimuli very economically. Some of the forms it handles, using a finite state grammar, are*

$$\textbf{Find} \;—\; \left\{ \begin{array}{c} + \\ - \\ * \\ / \\ ** \end{array} \right\} \;—$$

$$\textbf{Let} \;—\; \text{be} \;—$$

$$\text{Goto} \; \left(\left\{ \begin{array}{l} \text{step no.} \\ \text{step} \end{array} \right\} \right) \;—$$

$$\text{Stop}$$

$$\textbf{If} \;—\; \textbf{is} \left\{ \begin{array}{l} \textbf{greater than} \\ \textbf{less than} \\ \textbf{equal to} \end{array} \right\} —\;, \textbf{goto} \left(\left\{ \begin{array}{l} \textbf{step no.} \\ \textbf{step} \end{array} \right\} \right) —$$

$$\textbf{What is} \;——\; ?$$

$$\textbf{Find} \left\{ \begin{array}{l} \textbf{Sine} \\ \textbf{Cosine} \\ \textbf{Log} \\ \textbf{Square} \\ \textbf{Cube} \\ \textbf{Square root} \\ \textbf{Cube root} \end{array} \right\} \;(\text{ of })\; —$$

Step numbers may be incorporated in the strings constituting a paradigmatic response. The 'goto' statements of the interpreter's language essentially allow jumping and looping within a sequence of internal stimuli essentially constituting a program.

The stimulus 'what is A' would be responded to by locating a 'Sentence' of the form 'A = -- ' in the sentential store and keeping the value ready for further use. The interpreter provides for the use of 'that', 'the result', 'it' and 'result[1] to refer to any such quantity arising from the preceding operation. The interpreter used incorporates many sophistications, and descriptions of these may be found in (17). For instance, the terms in the interpreted statements could be numbers such as 3.61 or 4, or a string of words such as 'the weight of the stone'. In the last case, the numerical value associated with the string would be searched for in the sentential store. Units may follow terms.

*Following standard linguistic convention, alternatives appear within braces, while optional fragments appear within brackets.

The sentential store and the store for schemata along with necessary indexes are accommodated in nearly 500,000 words of a disc-file memory. A description of indexing and retrieval techniques specially developed for this work may be found in (18).

A number of problems solved by this system, along with the guiding paradigms may be found in (17). An example is listed here, (deletions, employed to save space here, are marked by '. . . ').

```
   PROBLEM SOLVED
C - 5 GRAM S OF SAND ARE AT 80 DEGREE S
        CENTIGRADE
    60 GRAM S OF KEROSENE ARE AT 10
        DEGREE S CENTIGRADE
    THE SPECIFIC HEAT OF SAND IS 0. 2
    THE SPECIFIC HEAT OF KEROSENE IS
        0.4
    THE SAND AND THE KEROSENE ARE
        BROUGHT TOGETHER

Q -  WHAT IS THE FINAL TEMPERATURE
        ATTAINED

Final
Output -               . . .
    FIND THE HEAT IN THE KEROSENE ,
    MULTIPLY 60 BY 0.4
    MULTIPLY THE RESULT BY 10
    THAT IS THE HEAT IN THE KEROSENE
    THE HEAT IN THE KEROSENE= 240. 000
    FIND THE HEAT IN THE SAND
                 • • •
    THE HEAT IN THE SAND = 80. 000
    ADD IT TO THE HEAT IN THE KEROSENE
    THAT IS THE TOTAL HEAT
    THE TOTAL HEAT = 320.000
    FIND THE THERMAL CAPACITY OF THE
        KEROSENE
    MULTIPLY 60 BY 0.4
    THAT IS THE THERMAL CAPACITY OF
        KEROSENE
    THE THERMAL CAPACITY OF KEROSE-
        NE = 24.000
    IGNORE THE TEMPERATURE - 10 DEG-
        REE S CENTIGRADE
    FIND THE THERMAL CAPACITY OF THE
```

```
        SAND


     IGNORE THE TEMPERATURE - 80 DEGRE-
          E S CENTIGRADE
     ADD IT TO THE THERMAL CAPACITY OF
          THE KEROSENE
     THAT IS THE TOTAL THERMAL CAPAC-
          ITY
     THE TOTAL THERMAL CAPACITY *
          25.000
     DIVIDE THE TOTAL HEAT BY THE TOT-
          AL THERMAL CAPACITY
     THE RESULT IS THE ANSWER, IN DEG-
          REE S CENTIGRADE
     THE ANSWER, IN DEGREE S CENTIGRA-
          DE = 12,800
     PARADIGMS USED
C -  75 GRAM S OF CLAY ARE AT 20 DEGREE
          S CENTIGRADE
     THE SPECIFIC HEAT OF CLAY IS 0. 5
Q -  FIND THE HEAT IN THE CLAY

A -  1. MULTIPLY 75 BY 0.5
     2. MULTIPLY THE RESULT BY 20
     3. THAT IS THE HEAT IN THE CLAY

C -  75 GRAM S OF METAL ARE AT 20 DEG-
          REE S CENTIGRADE
     THE SPECIFIC HEAT OF METAL IS 0. 5

Q -  FIND THE THERMAL CAPACITY OF THE
          METAL

A -  1. MULTIPLY 75 BY 0.5
     2. THAT IS THE THERMAL CAPACITY
          OF THE METAL
     3. IGNORE THE TEMPERATURE - 20
          DEGREE S CENTIGRADE

C -  THE IRON FILING S AND WATER ARE
          BROUGHT TOGETHER

Q -  WHAT IS THE FINAL TEMPERATURE
          ATTAINED

A -  1. FIND THE HEAT IN THE WATER
     2. FIND THE HEAT IN THE IRON FILINGS
     3. ADD IT TO THE HEAT IN THE WATER
     4. THAT IS THE TOTAL HEAT
     5. FIND THE THERMAL CAPACITY OF
          THE WATER
     6. FIND THE THERMAL CAPACITY OF
          THE IRON FILING S
     7. ADD IT TO THE THERMAL CAPACITY
          OF THE WATER
     8. THAT IS THE TOTAL THERMAL
          CAPACITY
     9. DIVIDE THE TOTAL HEAT BY THE
          TOTAL THERMAL CAPACITY
    10. THAT IS THE ANSWER, IN DEGREE S
          CENTIGRADE
```

8. Discussion

A number of questions can be raised about this approach. Some of these questions are answered here and certain lines for possible developments in consonance with this approach are discussed.

8. 1 Nature of the Language Provided for

Does the system at least provide for the use of phrase structure languages?

It will be apparent that we are not working with a slot-and-filler model, where the fillers are single words. Since a variable could be any string of words and symbols, there exists the possibility that such fillers themselves will have a complex structure. Fillers usually get transferred from the stimulus to the internal responses. Incorporation of a filler with a complex structure modifies the internal response and increases the processing it will go through. But eventually, any necessary breaking up of phrases does get carried out.

Further, it can be seen that there is provision for handling surface transformations. If the system can handle a question of the form 'Does N divide M?' , we can provide a new paradigm that will transform an equivalent question to this form:

```
C  :   -
Q  :  Is M divisible by N?
A  :  Does N divide M?
```

8. 2. Completeness and Contradictions

It is reasonable to ask of a problem-solving system if it guarantees the solution of any given problem belonging to some class of problems. Can it give answers which contradict each other? Can the problem-solver go into an endless loop in response to any given problem?

The system described here can execute an algorithmic process and in that trivial sense, it can guarantee solutions to any problem in a class which is algorithmically solvable. Operating in areas which cannot be wrapped up in formal systems, it can give contradictory solutions. It can go into an endless loop unless the paradigms provided indicate methods of avoiding such a possibility.

8. 3 Possible Developments

Useful minor additions would include a random number generator accessible in the interpreter's language and a name generator for creating arbitrary names for local variables. 'Let that be X[1], 'Let that be Y' etc. are sufficient for handling small problems, but one needs a name generator for larger problems.

The system presently handles words as units and cannot separate a word into its root and word-ending. A provision to handle this would be useful. It would also be useful to have a good evaluator for arithmetic expressions, instead of depending on the crude programming language of the interpreter. A very sophisticated addition would be a provision which will enable the system to write a few lines in a programming language such as FORTRAN and have them executed.

It would be very useful to provide for on-line interaction between the system and a communicant. This ought to be implemented easily by incorporating two new primitives in the interpreter's repertoire, for on-line input and output.

Typical symbol manipulation by a student (for example, that involved in symbolic integration or algebraic factorization) is often performed using an external aid such as a blackboard, or pencil and paper. It would be useful to provide a two dimensional array on which the system could write, sense, and manipulate alphanumeric expressions. With a number of manipulation primitives built into the interpreter's repertoire, this array could be used as an internal blackboard.

Such a 'blackboard' will also enable the writing, examination and 'editorial' manipulation *oi* long sentences. This might, to some extent, bridge the difference in complexity between the sentences used by the system and those found in written human communication.

Many of the facilities mentioned here are available in standard programming languages, but it would be interesting to see them used by a paradigmatic problem-solver.

A more basic advance would be the development of paradigms which would lead the system to use trial and error as a learning device in solving a variety of problems. Trial episodes which end in success should be automatically culled out as useful paradigms for schematization and storing. This will involve considerable addition to the system described.

Acknowledgment

This work owes much by way of guidance to Professor R. Narasimhan, and without his collaboration it would not have been possibles

Professor J. R. Isaac provided constant encouragement and his critical interest put many ideas into perspective.

Comments of the referees on the original draft are gratefully acknowledged.

References

1. Newell, A,, Shaw, J. C. , and Simon, H. A., Empirical Explorations of the Logic Theory Machine Proc. WJCC, 1957
2. Minsky, M., A Selected Descriptor-Indexed Bibliography to the Literature on Artificial Intelligence, IRE Transactions on Human Factors in Electronics, March 61
3. GPS, A Program that Simulates Human Thought, in 'Computers and Thought', Feigenbaum, E. and Feldman, J., Eds., McGraw-Hill, 1963
4. Newell, A., Shaw, J.C., and Simon, H.A., A Variety of Intelligent Learning in a General Problem Solver, Self-Organizing Systems (Yovitts, M., and Cameron, S. Eds.) Pergamon, 1960
5. Bobrow, D. G., 'Natural Language Input for a Computer Problem-Solving Syst em'$_t$ Ph. D. Thesis. M.I.T., 1964
6. Newell, A., The Search for Generality, Proceedings of the IFIP Congress 65, Spartan Books, 1965
7. McCarthy, J., Programs with Common Sense, Proceedings of the Symposium on the Mechanization of Thought Processes (Blake, D. V.,

and Uttley, A.M., Eds.), H.M. Stationary Office, 1959

8. Thompson, F. B., English for the Computer, Proc. FJCC, 1966
9. Zwicky, A.M., Friedman, J., Hall, B.C., and Walker, D. E., The Mitre Syntactic Analysis Procedure for Transformational Grammars, AFIPS Conference Proceedings, Vol. 27, Part 1, 1965
10. Coles, L.S., An On-line Question Answering System with Natural Language and Pictorial Input, Proceedings of ACM, Spartan Press, 1968
11. See the Chapters by Chomsky, N. in Handbook of Mathematical Psychology, Luce, R.D., et al, Eds., John Wiley, 1963
12. Narasimhan, R., Intelligence and Artificial Intelligence, Computer Studies in the Humanities and Verbal Behavior, Vol.11, Nr. 1, March 1969
13. Hockett, C.F., The State of the Art, Mouton, 1968.
1^ Weizenbaum J., Eliza - A Computer Program for the Study of Natural Language Communication Between Man and Machine, CACM, Vol. 9, No. 1, Jan. 1966.
15. Leavenworth, B. M., Syntax Macros and Extended Translation, CACM, Vol. 9, No. 11, Nov. 1966
16. Woods, W.A., Transition Network Grammare for Natural Language Analysis, CACM, Vol. 13, No. 10, Oct. 70
17. Ramani, S., Language Based Problem-Solving, Ph.D. Thesis, Computer Group, Tata Institute of Fundamental Research, 1969
18. Ramani, S., Indexing and Retrieval of Sentences in a Natural Language Data-Base, Tech. Report No. 89, Computer Group, Tata Institute of Fundamental Research, Jan. 1970.