MEMORANDUM
RM-3285-1-PR
FEBRUARY 1963

LEARNING, GENERALITY AND PROBLEM-SOLVING

Allen Newell

PREPARED FOR:
UNITED STATES AIR FORCE PROJECT RAND

The RAND Corporation
SANTA MONICA • CALIFORNIA

MEMORANDUM
RM-3285-1-PR
FEBRUARY 1963

LEARNING, GENERALITY AND PROBLEM-SOLVING

Allen Newell

This research is sponsored by the United States Air Force under Project RAND—contract No. AF 49(638)-700 monitored by the Directorate of Development Planning, Deputy Chief of Staff, Research and Development, Hq USAF. Views or conclusions contained in this Memorandum should not be interpreted as representing the official opinion or policy of the United States Air Force. Permission to quote from or reproduce portions of this Memorandum must be obtained from The RAND Corporation.

The RAND Corporation
1700 MAIN ST. • SANTA MONICA • CALIFORNIA

PREFACE

This Memorandum is a discussion of the concept of learning in the field of artificial intelligence and its intimate relationship to other concepts such as generality and problem-solving. It is part of a continuing RAND research effort in these areas.

The long range goals of artificial intelligence imply the ability for programs to be truly general purpose, in the sense of being able to acquire from their environment the information necessary to develop successfully in ways not envisioned in detail by their designers. The analysis given here shows that learning is generally viewed as the means to accomplish this. These issues are fundamental not only to the field of pure artificial intelligence, but to the whole attempt to develop more sophisticated information processing, such as in Command and Control systems.

To illustrate some of the questions brought up in the paper, a discussion is given of some very recent work done at RAND on a learning scheme for the General Problem-Solver (GPS). More details and specifications of this scheme will be given in a subsequent RAND Memorandum.

This revision of RM-3285-PR incorporates several changes and additions designed to present a more straightforward approach to the subject matter.

The work formed the basis of an invited presentation by the author at the International Federation for Information Processing Congress, 1962, at Munich, Germany on August 29, 1962.

SUMMARY

Learning plays a peculiar and subtle role in the field of artificial intelligence. Emphasis is placed on it in a way that seems to indicate it is a special concern, different from the other topics--problem-solving, pattern recognition, etc.--that also receive attention in the field.

An analysis of learning is presented to support the argument that the usual paradigm of learning (i.e., if the performance of a program on a task changes over time, then learning has occurred) does not get to the heart of the matter. This is done by showing that problem-solving programs are also learning programs by these standard criteria. A little more analysis supports the contention that the key features are the uses of the past for general utility and the use of genuine induction and generalization. Further probing reveals that we are really concerned with generality--with producing a machine general enough to transcend the vision of its designers. Learning is viewed as the major means to achieving this end.

Turning to the question of how to use experience, a review is made of the various techniques and ideas which have been used. It is argued that the standard schemes of repetition with the modification of statistical weights is not the most fruitful way to proceed. Rather, attention

is directed to the problem of representation, to the work on programs that construct programs, to the feature constructing pattern recognition programs, and to the programs for interpreting natural language.

To provide a concrete illustration of alternative ways to explore in developing learning programs, a modification of GPS is discussed that learns its own differences from an examination of the operators it is given with which it manipulates the objects of the problem. This scheme makes use of EPAM, a program for simulating the way humans do rote memory. Although this scheme does not involve either repetition, the piling up of statistical experience, or the use of success or failure data, it is clearly a learning program in its use of past experience for an indefinite future and its use of genuine techniques of generalization.

CONTENTS

I. INTRODUCTION*

In the field of artificial intelligence we are engaged in constructing mechanisms that reproduce the information processing we see in man. Sometimes we wish only to match or surpass his external performance; sometimes we are at pains to simulate his behavior in detail. For this memorandum the distinction is of little importance. For many sufficiently-limited, symbolic tasks, such as multiplication, we both understand the task and can greatly exceed man's performance; interest fades from these. We are more concerned with performances that seem highly sophisticated and elaborate, where we understand completely neither the task nor how to accomplish it. Yet we demand a certain clarity: dreaming, reverie, and wit still lie outside the operational boundaries of artificial intelligence in 1962.

Among the functions that do concern us, learning plays a peculiarly crucial and subtle role. For instance, the following typical questions are asked with great regularity by sophisticated visitors:

> "I understand that your program plays chess, but does it <u>learn</u> to play better?"
> "No."
> "Oh," (with an intonation of disappointment).

---

*I am much indebted to my colleague H. A. Simon for discussions on the issues raised in this paper.

> "Will your chess program repeat itself exactly if it is played against in the same way?"
> "Yes."
> "I see," (with a hint of satisfaction: the program has failed some crucial test).

Throughout the field of artificial intelligence emphasis is placed on learning. Pattern recognition programs continually underplay the performance of recognition, emphasizing instead the ability to learn to recognize new patterns. When Samuel's checker player[1] is cited in discussions of problem-solving, its learning is usually stressed as a crucial feature.

Why does learning play this crucial role? What is its special power? I would like to explore these questions and their implications for progress in artificial intelligence.

II. THE PROBLEM OF LEARNING

Learning is a broad, but well-established, concept that stands for a cluster of notions, many of which are implicit. Discussions of learning, such as the following, are not attempts at precision; they constitute hypotheses about what are the important features of the existing concept.

If we observe that a performance of X is a function of its experience in the past, then we say that X learned from its experience. This corresponds generally to the man-in-the-street's notion of learning. There are so many things wrong with it that the psychologists long ago elaborated it to the paradigm of Figure 1. At time T we observe the performance of X on a specific task; if at T' its performance on the same task has changed, it has learned something from its previous performance and intervening experience. The task may be repeated over and over again in order to study the course of learning. With a few caveats, this paradigm has sufficed in the psychological laboratory.

The paradigm is also much used in the study of artificial intelligence. It has even been elaborated slightly, since we can determine the internal structure of our learning machines precisely. The machine consists of a performance program for doing the task and a learning program that

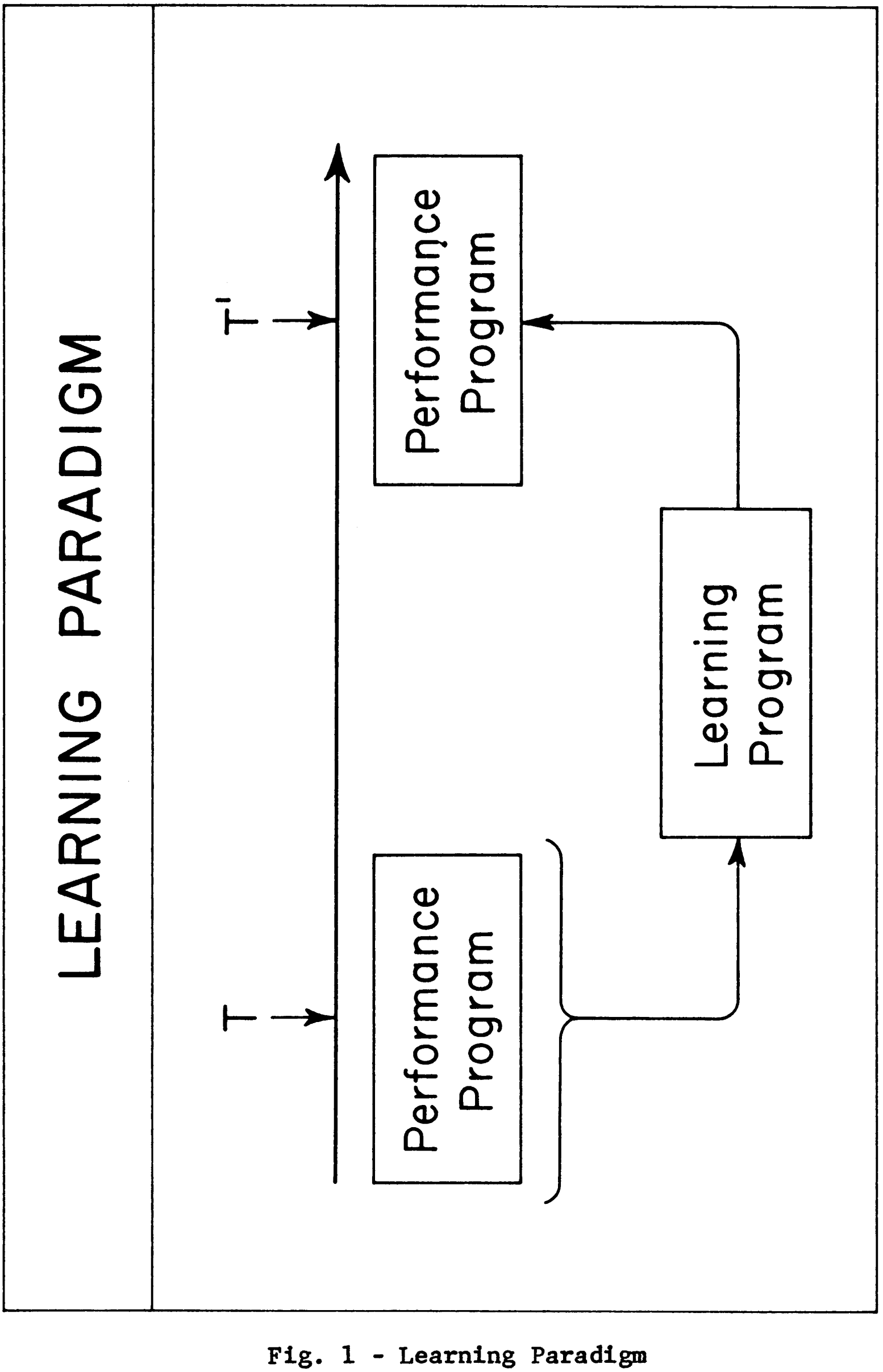

Fig. 1 - Learning Paradigm

modifies the performance program as a function of its behavior on the task.

Does this paradigm express the aspects of learning that are really important to us? Let me test it against existing problem-solving programs. By common consent these are not learning programs--witness the dialogues just quoted on the chess program. Yet they fit the paradigm exactly. Figure 2 shows the top-level flow diagram of LT, one of the early theorem-proving programs.(2) The executive routine is performed repeatedly in the same situation. On first performance it usually does not solve the problem and on some later performance it does. Its eventual success--that is, its change in performance--is due to the experience accumulated from the intervening trials. Even LT's internal structure corresponds to the structure of a learning machine. The top box with the subproblem tree corresponds to the performance program. This includes the Substitution Method, which is the only method in LT that can actually solve problems. The second box in the diagram corresponds to the learning program. It contains the Detachment and Chaining Methods, which create new subproblems and add them to the subproblem tree--that is, modify the performance program. This part is called into play only after the performance program fails. Learning occurs only on failure. And the modification is adaptive, since by construction it generates

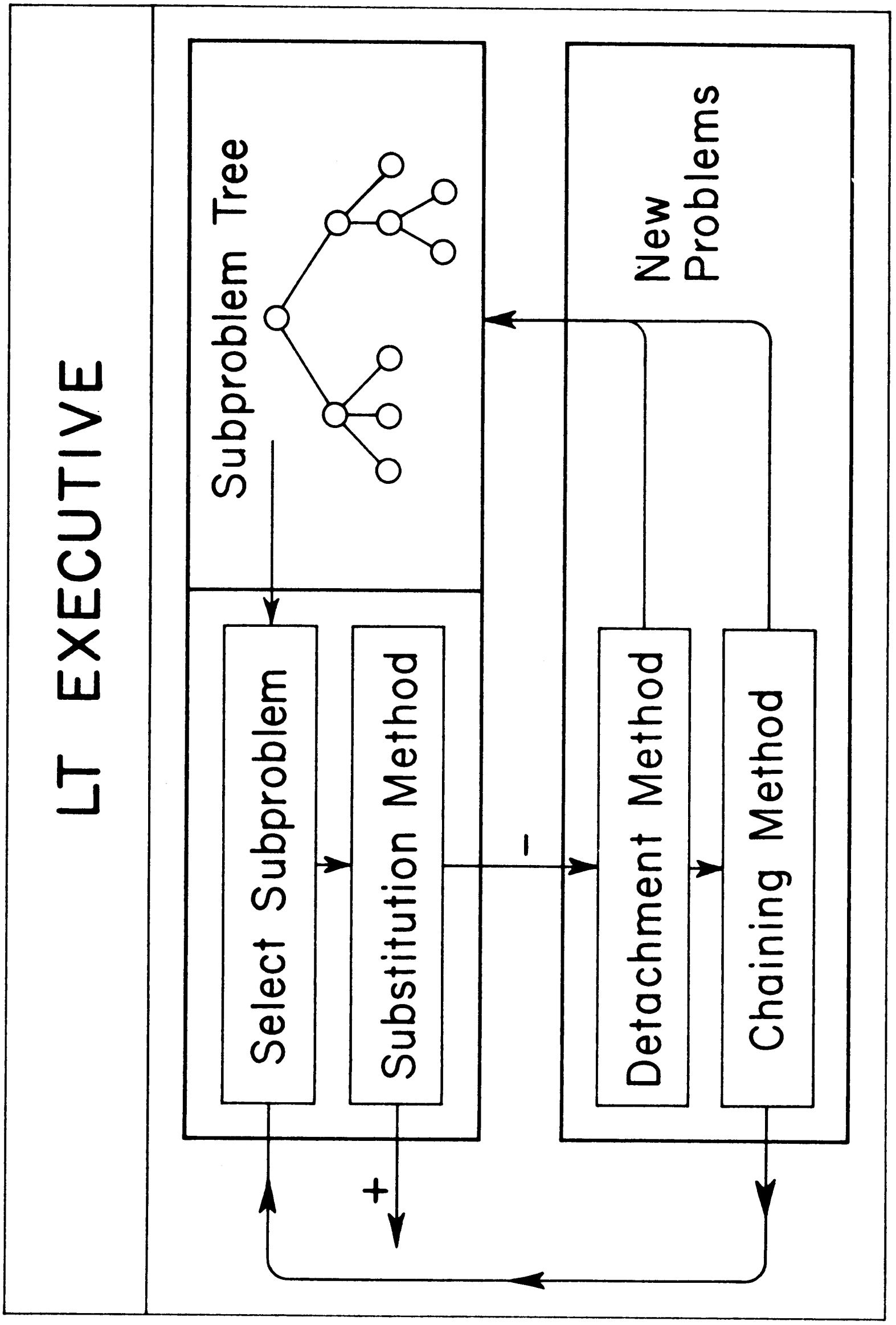

Fig. 2 - LT Executive

new subproblems that increase the probability of success on later trials.

This example is not a trick. LT can in fact be said to learn the solution. However, LT is still not usually considered a learning program, nor do I consider it one. Important elements exist in our concept of learning besides those given in the standard paradigm.

The most important way the problem-solvers seem deficient as learning machines is in the use they make of their experiences. All the information is accumulated to obtain a single result: to prove the original theorem. Once this is done, the experiences have no more value. In contrast, the devices we do call learning machines use their experience to prepare for a whole range of future situations. Pattern recognizers use the learning trials to extract information from which innumerable varying exemplars can be recognized. Samuel's checker player improves its function for evaluating positions, which is used in all future play. Significant learning seems to imply obtaining something of general utility from the experience.

We also seem to require a learning program to use genuine generalization and induction in processing its experiences. The problem-solvers are again deficient on this score, since what is learned from each new experience--from each new subproblem--is primarily that it is <u>not</u> a solution. In contrast, some basic postulate of generalization

is a prominent part of all current learning machines. Often this is simply, "what has worked in the past will work in the future," but it always exists.

III. THE PROBLEM OF GENERALITY

These added factors--general utility and induction--which imply significant learning, as opposed to learning that simply satisfies the paradigm, reveal a deep concern in artificial intelligence to construct a machine general enough to transcend the vision of its designer.* Learning is viewed as the major means for achieving this. Man has the ability to get along on his own and to define for himself the terms on which he will enter into dependencies with parts of his environment. In artificial intelligence we face the prospect of externally special-purpose machines, brilliant within a narrow range, but always encased within an artificial universe bounded by the limited vision of its designers.

Generality, then, is one of our major goals: the ability to cope with the range and diversity of the real world. Theoretically, we do not care how we get such a machine. We would accept a perfectly constructed general problem-solver. Such a machine would not need to learn at all! That we reject this possibility--as by common consent I believe we do--expresses the conviction that the real world is too diverse, too full of details, and too complex

---

*This same concern is expressed, although improperly, in the question of whether a machine can outperform its designer. This question was laid to rest long ago, both trivially in tasks like multiplication and non-trivially in tasks like checkers.

to permit any pure "performance" device to achieve perfection. Continuous learning is the price of generality.

When learning is emphasized as the basic solution to the problems of generality, we must include all the ways experience might be processed to prepare for future action. The world itself determines what regularities exist over time and place to be exploited by learning; their nature determines in large measure the kind of processing that must be used.

However, current learning machines utilize an extremely narrow range of mechanisms. The central ideas are well known: repetition, simple reinforcement, statistically determined weights. They derive in many ways from taking seriously the paradigm of Figure 1, as realized in animal experimentation. The range of relevant processes is clearly broader than this, however difficult it may be to visualize the possibilities. Our horizons need expanding on the varieties of mechanisms that are relevant and requisite to achieving generality.

IV. REPRESENTATIONS

All learning requires an internal representation of the experiences to be made available for later use. This may be as simple as a recorded fact, as obscure as the connectivity of a network, or as elaborate as a detailed map, but it must exist. The representation forms a crucial bridge, not just between the moment of gathering and the moment of using information, but between descriptions of the environment and determinations of action.(3) The representation is pulled in two directions. Processes must translate from the raw experience to the representation; to simplify this translation, the representation should be simple in terms of the environment. But processes must also translate from the representation to action; to simplify this translation, the representation should be simple in terms of the action principles of the machine. Complexity of translation can be exchanged between the encoding and decoding processes by an appropriate choice of representation.* In all events, a certain gap must be bridged to get from the environment to its implications for action. The representation remains the filter through which all information must pass.

Representations of experience are crucial in another way. They form not only limits to what can be accomplished

---

*There is a nice discussion of this in Lindsay.(4)

by the machine, but limits to what can be envisioned by the designer. New forms of learning require the invention of representations that can express new potentialities--that make variable what was previously fixed structure. These extensions are rarely suggested by existing structures; they constitute true inventions. For example, LT never learned new methods because we could not invent a space of possible methods expressed in a language that LT could manipulate. A consequence of this paucity of good representations is that existing learning programs tend to cluster around the few existing representations. Here, above all, we need our horizons extended.

Most of the representations currently in use are extremely close to the action scheme of their machines. This is quite natural, since the easiest way to get a learning scheme is to generalize some feature of the way an existing machine performs. The problem of translation from the representation to action is thereby solved, since the representation is already in terms directly understood by the performance program. However, translation of the raw experience into the representation must still occur. Our excessive tendency to describe an environment solely by the dichotomy, "succeed" or "fail," may partly reflect the difficulties of this translation.

Many learning programs use some collection of numerical parameters of the action scheme as a representation, which

are then optimized by experience. From our viewpoint here, these are the least interesting kinds of learning precisely because they never lead beyond themselves--which does not deny their occasional effectiveness. The variety of the world is far richer than can be represented by a fixed set of numbers controlling a machine of fixed structure.

More interesting are representations using extremely general languages of action. Much of the work in learning machines has used networks of linear threshold elements, the set of threshold values serving as the representation of experience. If I dismiss this work somewhat summarily, it is only because this representation still seems too limiting to express, without great organizational innovation, the kind of detail and complexity, both of action and environment, that we see in the world.

Programming languages, the one known class of action languages able to express truly complex behaviors, are more intriguing. However, programs are extremely obdurate languages for encoding descriptive experiences, which is why they have not been used much in learning programs. Early examples, such as Friedberg's,[5] did not incorporate enough power and structure to fashion programs that could accomplish useful tasks. Work by Kilburn, Grimsdale, and Summer[6] removed some of the deficiencies, and was considerably more successful. Currently, at least two efforts are developing programs that construct programs. One, by Saul Amarel,[7]

takes a set of examples of the desired inputs and outputs and tries to discover a program that yields these; the other, by H. A. Simon,[8] takes general defining statements about the input and output and attempts to find the program. These programs are not learning programs, in the same way as LT is not one.* They solve the problem of constructing a program that meets certain conditions. Their significance lies in developing techniques for translating from descriptive information to a language of action rich enough to express complex behavior. Continued advance along these lines should finally permit representations of experience which are much richer and more directly reflect the environment than the action-oriented representations we use currently.

Another kind of action language is worth noting. Pattern recognition of the non-network variety tends to use a set of features of the sample to be identified, along with learning schemes for selecting and weighting these features. The representational limitations of sets of weights have already been mentioned. Recently, programs have been written that produce their own features. The most advanced scheme is that of Uhr and Vossler.[9]**

---

*Nevertheless, the early programs called themselves learning programs.

**This program was also reported on at the IFIP Congress. Interestingly, the very early program of Selfridge[10] and Dineen[11] also created its own features.

From our viewpoint the advance of these programs derives from representing past experience not merely as numerical weights, but as features.

The ways of representing experience discussed above were developed directly out of the action structure of the machine. If we turn to the other side--to representations that mirror the environment--less has been done. The work of Remus[12]* moves in this direction by basing actions directly on a descriptive classification of the environment. More generally, these would be programs that seek to construct models, theories, or explanations of their environment, independent of particular action implications. A hypothesis-testing program by Feldman, Tonge, and Kanter[13] is an example. Programs designed to obey natural language[4,14] also constitute exceptions, since they face the problem of first understanding what is being said. These latter programs, by exploring the possibilities for environment-oriented representations, have great relevance for generality and learning, even though they are not cast directly in the form of learning machines.

I have been emphasizing the critical role of the internal representation of experience, and the limits it puts on our vision in constructing general machines. Whole ranges of mechanisms seem to me to lie outside the

---

*Also presented at the IFIP Congress.

view of much of the current work on learning machines, limited as they are by implicit assumptions about how experience should be inducted and represented. Consequently, I have been stressing those areas of research--programs that construct programs, recognizers that create new features, and programs that build descriptive models of the environment--which expand our horizons by making available new forms of representation. An example will illustrate further the possibilities for using experience.

V. AN EXAMPLE FROM GPS

Most readers are probably acquainted with the program called GPS, which was described at the International Conference on Information Processing, Paris, 1959,[15] and has been reported several times since.* GPS is a computer program for solving problems, a member of the class of game-playing and theorem-proving programs. It has been used both for explorations into artificial intelligence and for detailed simulation of human problem-solving.[16,17] Its performance in both of these aspects has been dominated by the problems it has presented in program organization.[18]

To refresh your memory, Figure 3 shows the general scheme by which GPS operates. Much goes on under the surface of this diagram; yet it is still the most adequate simple picture. GPS is a program for accepting a task environment defined in terms of discrete objects, operators that manipulate these objects, and particular tasks like transforming one object into another. It performs these tasks by growing a structure of goals. Each goal is a data structure that describes some state of affairs to be achieved and gives ancillary information about methods, history, and environment. Different types of goals exist, each with its

---

*GPS is the joint work of J. C. Shaw, H. A. Simon, and the author. I am indebted to Charles Bush for his help in running GPS.

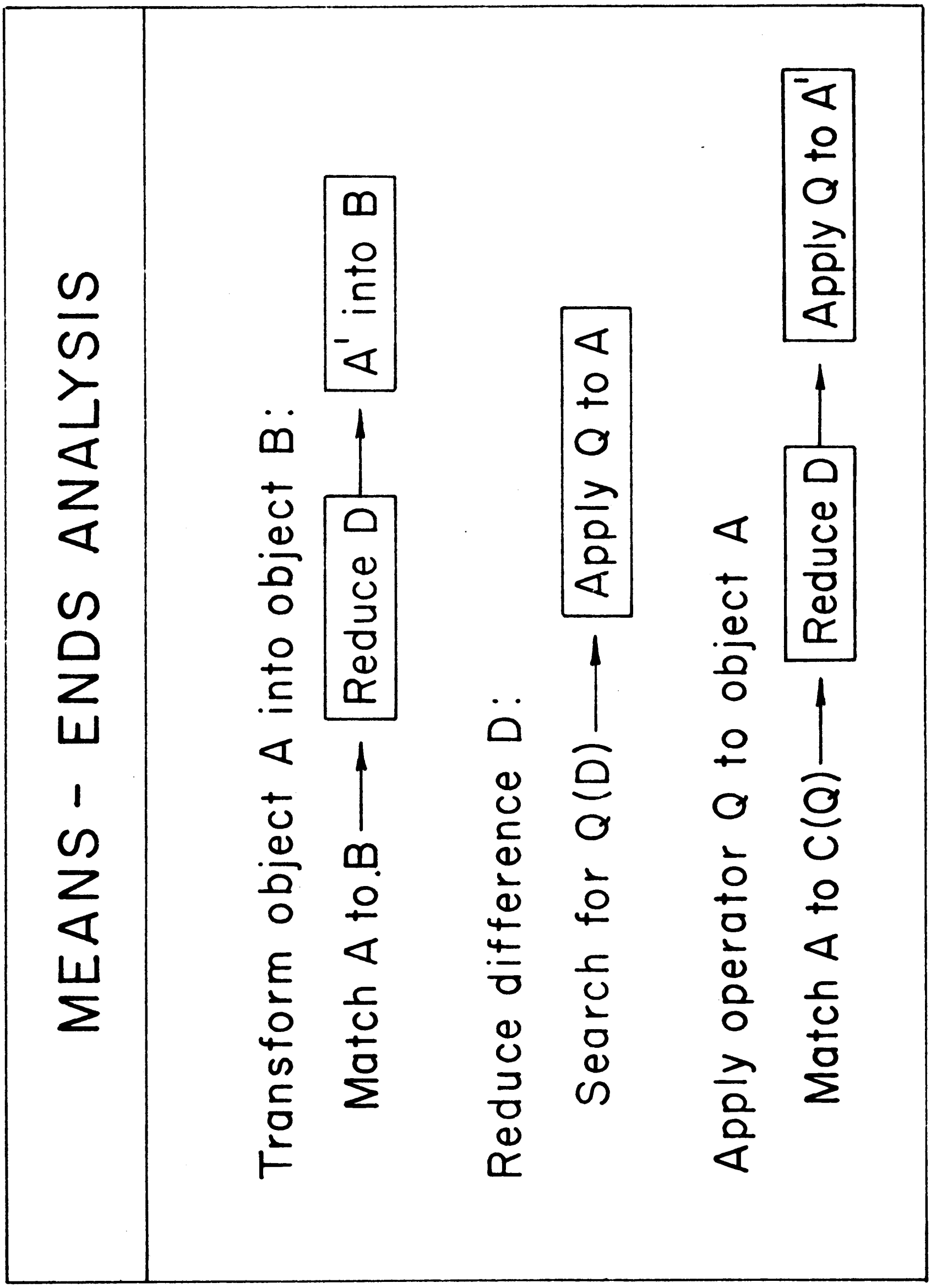

Fig. 3 - Means-Ends Analysis

own methods. The three methods shown are the crucial ones. To transform an object A into an object B, the first method matches the two objects; if they are not the same, a difference is found, which leads to setting up a subgoal of reducing that difference. If this subgoal is attained, a new object A' is produced, which hopefully is more like B. The subgoal is then set up to transform A' into B. Secondly, the principal method to reduce a difference is to find an available operator relevant to that difference, and to set up the subgoal of applying it. In the third method, to apply an operator the operand is matched to the conditions required by the operator. This may lead to a difference, which requires setting up a subgoal to reduce it, similar to the earlier case. The methods, these three along with others, grow a tree of subgoals in the process of solving a problem. In addition, GPS has devices for pruning and shaping the tree of subgoals: checking for duplicate objects and goals, rejecting goals as unprofitable, and selecting goals as especially worthwhile.

With this brief sketch, let me consider some of the issues raised earlier. A problem for GPS is basically defined by objects and operators. The differences are the constructs that mediate between them, that allow GPS to select artfully the operators that are appropriate to transform one object into another. Figure 4 shows the table of connections for

LOGIC TABLE OF CONNECTIONS

	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12
Add Variables									X	X		X
Delete Variables								X			X	X
Increase Number			X				X		X	X		X
Decrease Number			X				X				X	X
Change Connective					X	X	X					
Change Sign		X			X	X						
Change Grouping				X			X					
Change Position	X	X										

Fig. 4 - Logic Table of Connections

logic, whereby differences lead to relevant operators. The differences are not given explicitly by either the task environment or the specific task; clearly GPS should learn them for itself. The fact of the program is otherwise: we, GPS's designers, give it differences as part of its equipment to deal with a task when we define a task environment. Thus, differences represent a sensitive point of dependence of GPS upon its designers.

Three years ago in an article entitled "A Variety of Intelligent Learning in GPS,"[19] we discussed these same issues, and there also focused on the differences. First we observed that, although a simple standard learning paradigm could be used to learn the connections in the table of Figure 4, the information could be generated directly. If the input and output forms of the operators were matched, the differences so found would be the table entries--that is, the operators were relevant to precisely those differences they produced. Looking, rather than learning from repetitive trial, was the appropriate way to gain experience here.

We went on to consider how the differences themselves might be generated by GPS. As initially created, each difference was simply a symbol, linked to a machine language program to detect the difference. From GPS's viewpoint differences were unanalyzable unities; from ours they were members of the class of all programs. From either viewpoint

no learning was possible. We first had to invent a representation of differences in terms of a simpler programming language--one in which differences, though programs, were represented as objects. We then gave GPS operators to construct and modify these difference programs, and differences to detect features of these programs. We thus cast the learning of differences into the form of a problem for GPS in terms of objects and operators, a problem it could handle by the same means it handles all problems. We carried this effort through a small hand simulation, problems of program organization still preventing our simulating it in the metal. Our exploration went far enough to make the point that intelligent learning might look more like problem-solving than like a stochastic learning process.

I would like now to consider the matter along an alternative path, although I shall make much the same point. Again, GPS is to obtain somehow its own differences. However, GPS cannot be given only the objects and operators; some elementary form of perceptual discrimination must be available to it. We assume that for each elementary attribute of an object structure, GPS can tell its location in the structure, and whether corresponding attributes of two expressions have the <u>same</u> or <u>different</u> values. If we provided GPS with less perceptual capability than this, we would be hiding some of the environment from it.

In the best of all worlds, GPS would have an operator available to remove directly every elementary difference between a given object and a desired object. If at position P attribute A for the given object had the value X while for the desired object it had the value Y, then an operator would exist which would change X to Y without disturbing anything else. In reality, GPS must work with operators that are not nearly so obliging as these operators of immediate perception. (Real saws produce sawdust, as well as cut.) Matching can be looked on as an act of wishful thinking--of seeing the problem in terms of the immediate perceptual operations that would make one expression like the other. By way of example, Figure 5 shows two simple logic expressions as GPS would see them. Each consists of a collection of nodes, linked together by the attributes. If the two expressions are matched, the difference structure shown below them is obtained; it specifies all the elementary things one would do (if one could) to change the first expression directly into the second.

That the real-world operators are inconvenient and do not allow such direct transformations, does not make them unpatterned. Each one can be seen as a composite, made up of these same elementary perceptual operations. Figure 6 shows the difference structure derived by matching the input form to the output form of an operator. Thus, the

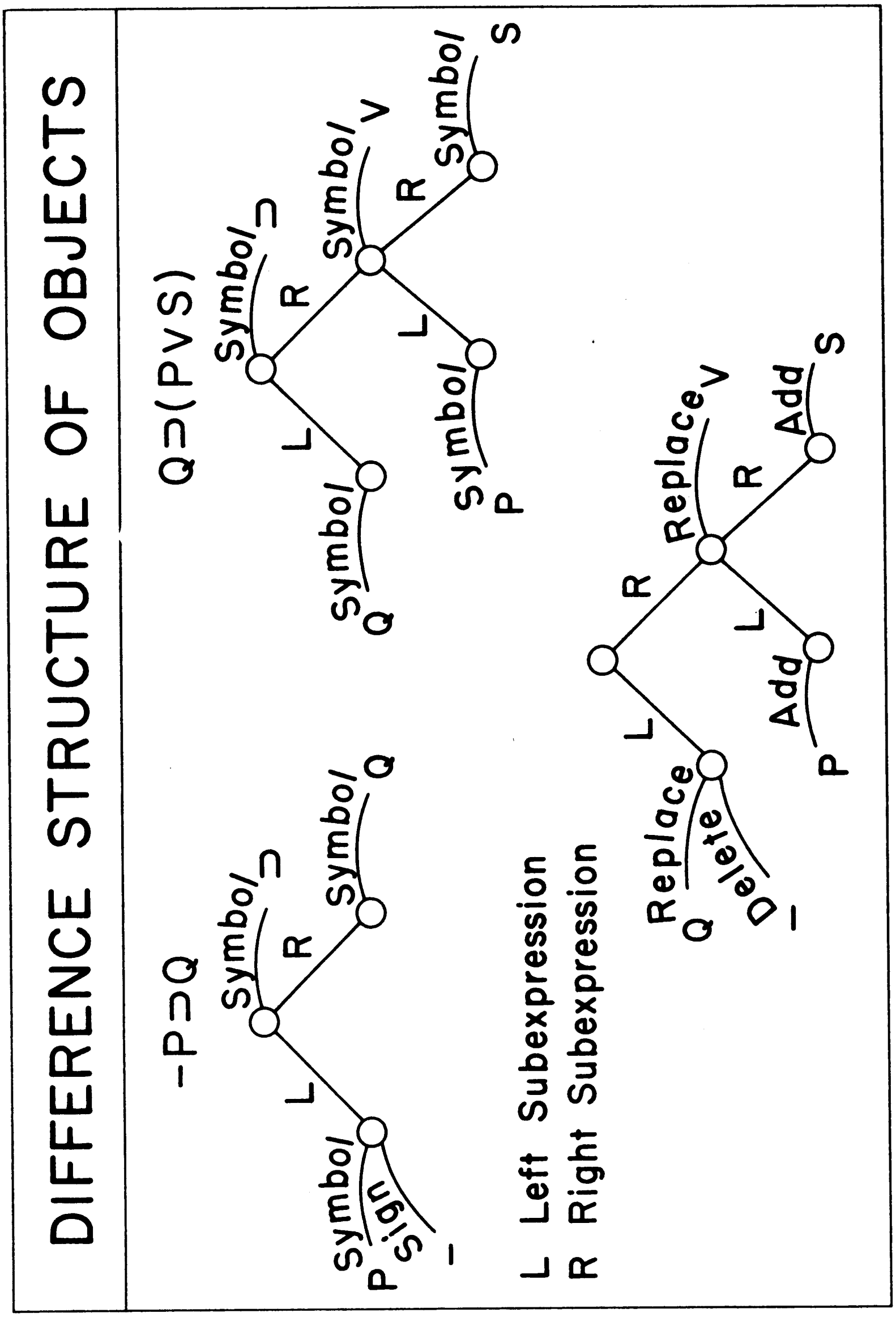

Fig. 5 - Difference Structure of Objects

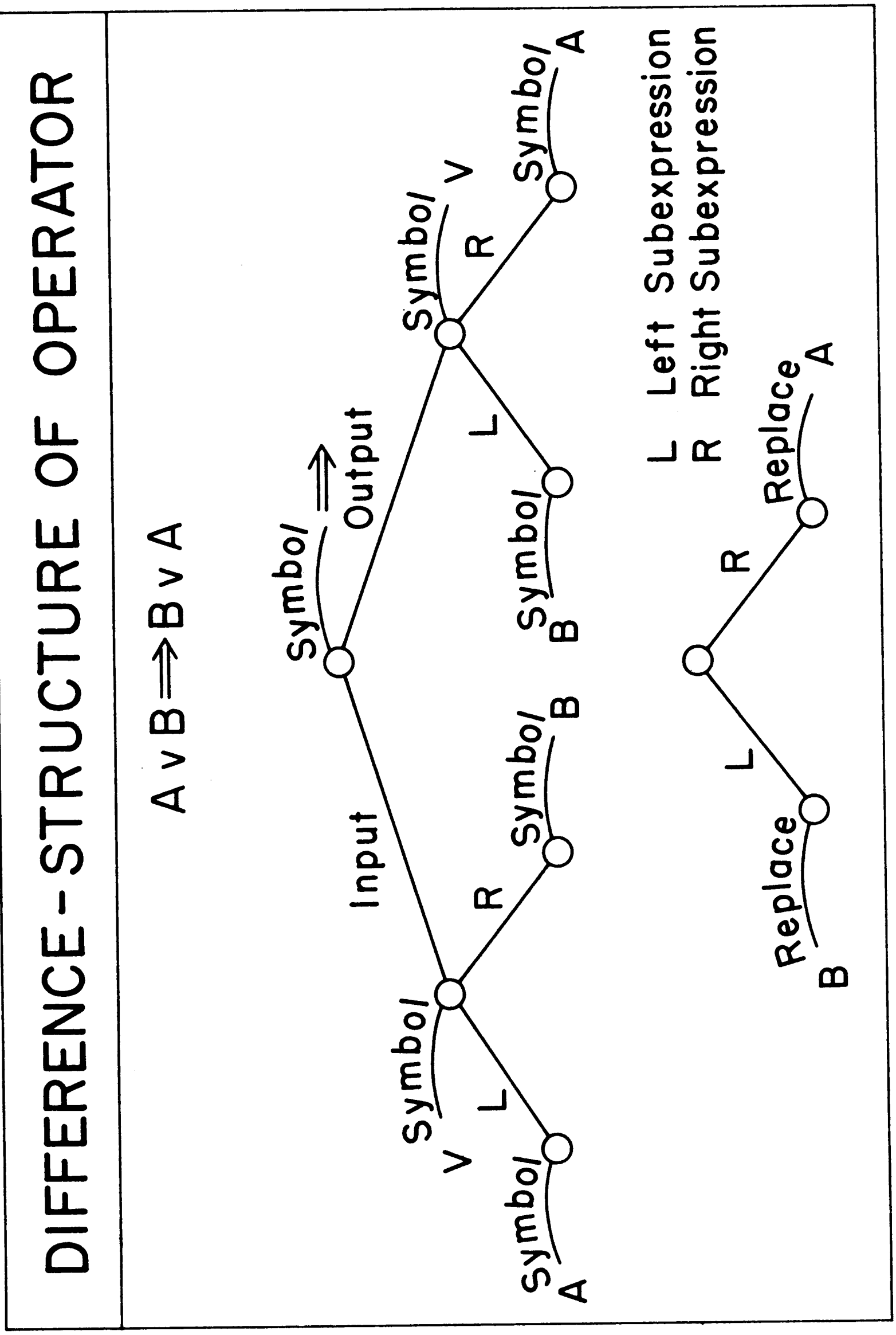

Fig. 6 - Difference Structure of Operator

basic bridge from environment to action is built; both objects and operators are described in a common language of immediate perceptions.

The price paid for this becomes evident as soon as we attempt to construct the table of connections from differences to operators. When the differences were specified by the designers, each difference represented a macroscopic change (some pattern of the simple differences just discussed), known to be of utility in solving problems in terms of the set of operators available. Thus there was a difference, change of position, and operators, such as A.B⇒B.A, AvB ⇒ BvA, and A ⊃ B ⇒ -B ⊃ -A, which dealt simply and directly with this difference. The table of connections was formed between the names of the individual differences and the names of the individual operators. With the elementary differences we are now considering, this is no longer possible, since each operator has a complicated structure of differences.

I will borrow a technique from the work of two of my colleagues to deal with this. E. A. Feigenbaum and H. A. Simon have reported on a program called EPAM.[20] Basically, EPAM consists of a very simple but intriguing scheme for discriminating among a collection of objects by growing a tree of tests when confronted with the task of sorting the objects. GPS already uses EPAM-like trees. During the course of problem-solving each newly created

structure (either a goal or an object) is sorted through a tree, which normally grows to accommodate it. However, if an identical structure has already been stored in the tree, it is discovered, since the new structure comes to rest at the same place in the tree. Appropriate action is then taken. For example, by this sorting process GPS immediately recognizes the final desired object if it ever generates it, independently of its reason for generating it. GPS does not have to ask the question deliberately of each new object, "Are you the final answer?"

An EPAM net will be used for the table of connections, growing it under the impact of the operators that are available for a problem. Figure 7 shows the tree that might be produced by discriminating the standard set of logic operators. The difference structure of each operator was sorted down this tree until it came to rest either at an unoccupied slot, or at a place holding a previously stored difference structure. In the latter case, the two difference structures were matched and the most important difference between them was used to define a test from which new branches would develop to discriminate the two operators. Once the tree of operators has been grown, it can be used to select operators for reducing the differences found between two objects. Their difference structure can be sorted down this tree to select the operators that most closely fit the pattern of elementary differences in the structure.

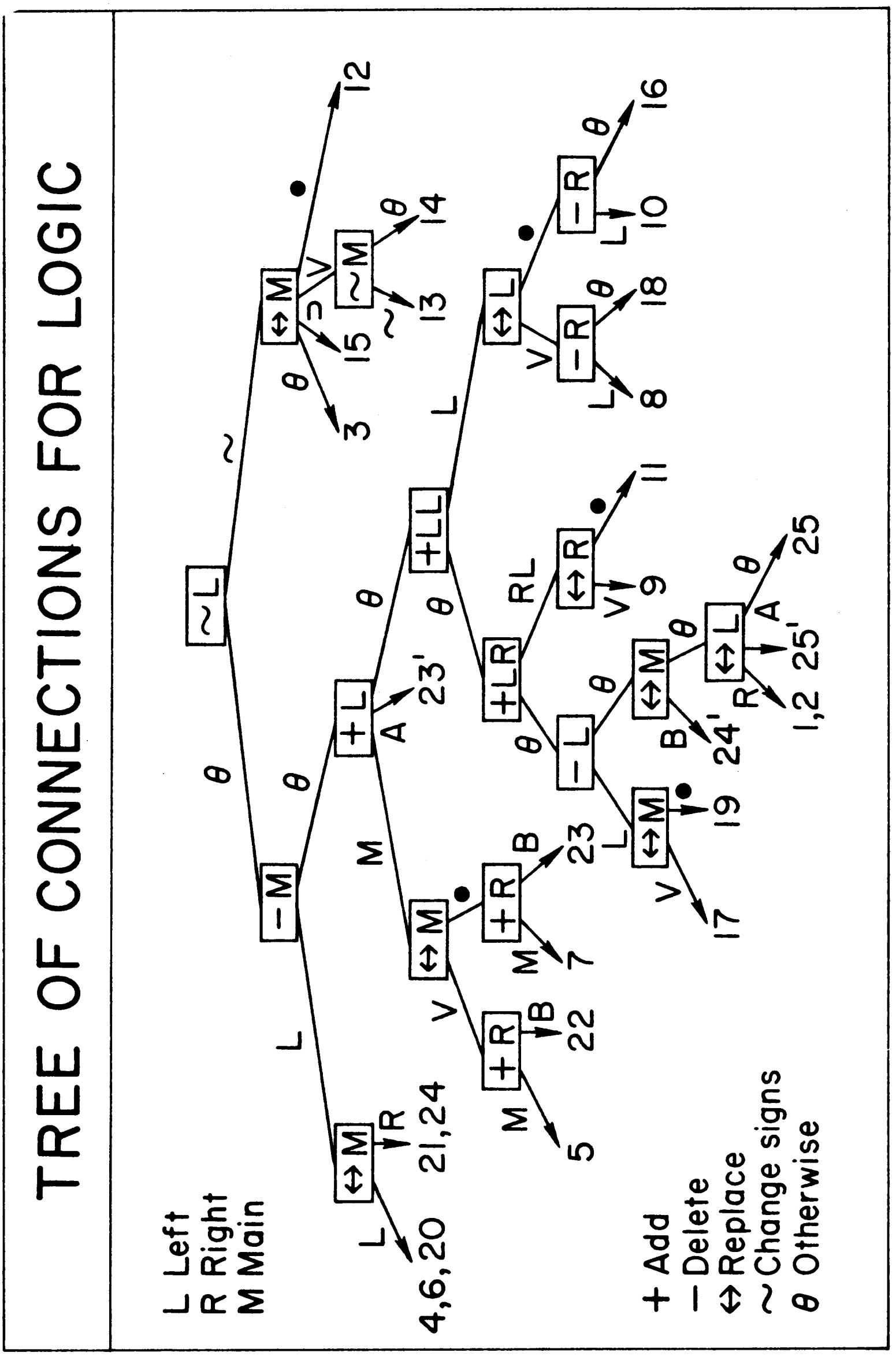

Fig. 7 - Tree of Connections for Logic

Let us consider the total operation of GPS with such a modification. In a new task area, GPS first explores the environment by processing the operators as has been shown; the tree that is grown represents the set of operators in their capacity to modify differences in this environment. Then GPS is prepared to turn to particular tasks, using this representation as a central part of its problem-solving.

One aspect of this description is still missing. Generalization was a central feature of the discussion of learning; one part of the environment is relevant to another only by an inductive leap. In most learning problems one is trying to pass between essentially analogous situations, and the basis of induction from one to the other is usually some form of invariance: what was good there is good here. In the present arrangement, although the inductive bases are different, generalization is by no means absent.

At least three generalizing assumptions are incorporated in this scheme. The most important is the assumption of "conservation of symbols." A term in a desired expression must come from somewhere: if SvT is to transform into TvS, then it is a fair assumption that the T in the second expression "comes from" the T in the first. Thus GPS creates a <u>generalized difference structure</u> by replacing each term in the <u>raw difference structure</u> with an expression representing the location of that term in the expression.

Figure 8 shows the generalized difference structures for the two examples of Figures 5 and 6.

The second generalizing assumption is spatial invariance: most operators in logic can be applied at any location in the expression. Hence the position of the total pattern of differences in the total expression is irrelevant. Consequently, the lowest node of the raw difference structure that covers all differences is made the top node of the generalized difference structure. This assumption is quite analogous, both in its power and in its inductive basis, to the transformations, such as centering, focusing, and smoothing, that are standard in many pattern recognizers.

The third generalizing assumption concerns the sign of the logic expressions. Whenever operations on a binary variable (say with values + and -) are symmetric in the values, it is possible to describe both changes from + to - and from - to + simply as "change value." This is a familiar transformation in all work with Boolean variables. As shown in Figure 8, the generalized difference structure is transformed this way.

GPS, of course, neither learns nor discovers these bases of generalization. To do so would require inventing a representation of the environment that would include in a plausible way many possible bases. Neither GPS, nor its designers, know yet how to specify such a representation.

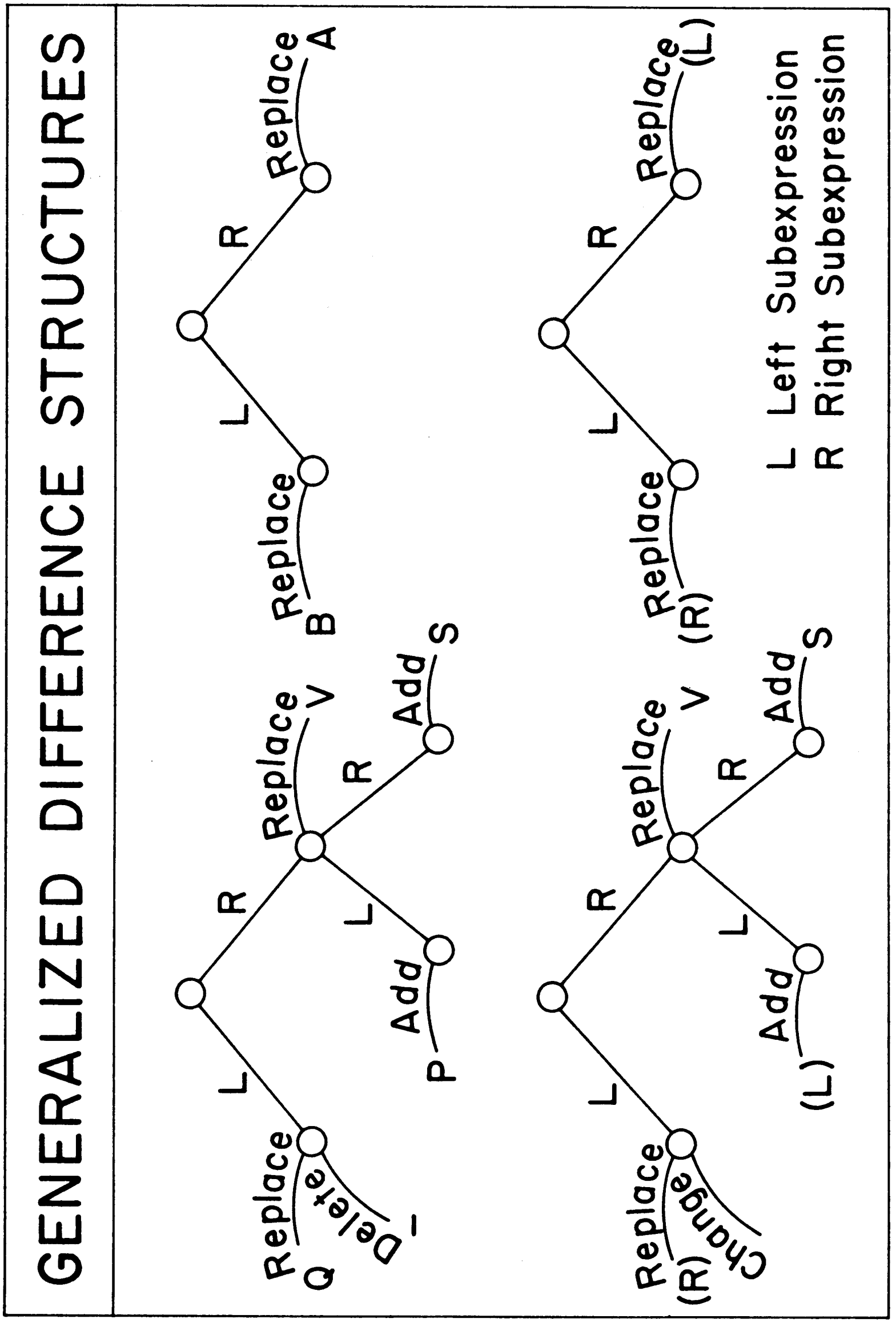

Fig. 8 - Generalized Difference Structures

Surely what we have been describing here is learning, even though there is no repetition of experience, no success or failure, no piling up of statistical data about the past. The key features of learning--preparation for an indefinite future and the use of generalization to bridge separate experiences--are both clearly present. Something akin to an analysis of the environment is going on. So, does one problem-solving program slowly grow independent of its designers.

REFERENCES

1. Samuel, A. L., "Some Studies in Machine Learning Using the Game of Checkers," IBM J. Res. Develop., Vol. 3, No. 3, July 1959, pp. 210-229.

2. Newell, A., J. C. Shaw, and H. A. Simon, "Empirical Explorations of the Logic Theory Machine: A Case Study in Heuristic," Proceedings of the Western Joint Computer Conference, 1957, Institute of Radio Engineers, New York, 1957, pp. 218-230.

3. Newell, A., and H. A. Simon, "Computer Simulation of Human Thinking and Problem Solving," M. Greenberger (ed.), Management and the Computer of the Future, Wiley, New York, 1962.

4. Lindsay, R. K., "Toward the Development of a Machine Which Comprehends," Unpublished PhD thesis, Carnegie Institute of Technology, 1961.

5. Friedberg, R. M., "A Learning Machine: Part I," IBM J. Res. Develop., Vol. 2, No. 1, January 1958, pp. 2-13.

6. Kilburn, T., R. L. Grimsdale, and F. H. Sumner, "Experiments in Machine Learning and Thinking," Information Processing, UNESCO, Paris, 1959, pp. 303-309.

7. Amarel, S., "On the Automatic Formation of a Computer Program Which Represents a Theory," Proceedings of Conference on Self Organizing Systems, Chicago, May 1962 (to be published).

8. Simon, H. A., Experiments with a Heuristic Compiler, The RAND Corporation, P-2349, June 1961.

9. Vossler, C., and L. Uhr, "Computer Simulations of a Perceptual Learning Model for Sensory Pattern Recognition, Concept Formation, and Symbol Transformation," International Federation for Information Processing Congress, 1962.

10. Selfridge, O., "Pattern Recognition and Modern Computers," Proceedings of the Western Joint Computer Conference, 1955, Institute of Radio Engineers, New York, 1955, pp. 91-93.

11. Dineen, G. P., "Programming Pattern Recognition," Proceedings of the Western Joint Computer Conference, 1955, Institute of Radio Engineers, New York, 1955, pp. 94-100.

12. Remus, H., "Simulation of a Learning Machine for Playing GO," International Federation for Information Processing Congress, 1962.

13. Feldman, J., F. Tonge, and H. Kanter, Empirical Explorations of a Hypothesis-Testing Model of Binary Choice Behavior, SP-546, System Development Corporation, Santa Monica, California, 1961.

14. Green, B. F., A. K. Wolf, C. Chomsky, and K. Laughery, "Baseball: An Automatic Question-Answerer," Proceedings of the Western Joint Computer Conference, 1961, Institute of Radio Engineers, New York, 1961, pp. 219-224.

15. Newell, A., J. C. Shaw, and H. A. Simon, "Report on a General Problem-Solving Program," Information Processing, UNESCO, Paris, 1959, pp. 256-264.

16. Newell, A., and H. A. Simon, "GPS, A Program That Simulates Human Thought," H. Billings (ed.), Lernende Automaten, Oldenbourg, Munich, 1961.

17. Newell, A., and H. A. Simon, "Computer Simulation of Human Thinking," Science, Vol. 134, No. 3495, December 22, 1961, pp. 2011-2017.

18. Newell, A., Some Problems of Basic Organization in Problem Solving Programs, The RAND Corporation, RM-3283, November 1962. Presented at Conference in Self Organizing Systems, Chicago, May 1962.

19. Newell, A., J. C. Shaw, and H. A. Simon, "A Variety of Intelligent Learning in a General Problem Solver," M. C. Yovits and S. Cameron (eds.), Self Organizing Systems, Pergamon, New York, 1960.

20. Feigenbaum, E. A., and H. A. Simon, Generalization of an Elementary Perceiving and Memorizing Machine, International Federation for Information Processing Congress, 1962. The RAND Corporation, P-2555, March 1962.