Final Report

RESEARCH ON HEURISTIC PROBLEM SOLVING MACHINES

Westinghouse

ELECTRIC CORPORATION

Final Report

RESEARCH ON HEURISTIC PROBLEM SOLVING MACHINES

March, 1965

Contract Nonr 4483 (00)
For the Period March, 1964 to March, 1965

Prepared for the
Office of Naval Research
by

Westinghouse Electric Corporation
Surface Division
P. O. Box 1897
Baltimore 3, Maryland

Approved by: [illegible]
S. E. Lomax, Director

WGD-38093 MDE 4941

Summary

This report describes the results of a study performed for the Information System Branch, Office of Naval Research under contract Nonr 4483(00), reference NR 348-008, Westinghouse reference WGD-38093.

The ultimate goal of this study was the development of a technique which will give a machine the facility to learn a decision process for the solution of problems in a particular class. This will be done from information gained by the performance of both man and the machine on examples drawn from this class. The machine is to have the abilities to infer a model of the man's performance on examples, to perform examples using its model, to describe its model to the man, and to accept and use corrections initiated by man to modify its model. This approach is called Imitation and physical implementation of the imitation process is a machine program called the Imitator. The imitation concept is a general one, applicable to a broad range of problems which man can solve. Its purpose is to provide a machine with the precise program it needs to solve problems. For this study, the Imitator was simulated by a digital computer.

During the past year, subgoals which lead to the ultimate goal have been achieved. The main effort has been the development of a study vehicle to aid in the formulation of concepts which will ultimately result in an ideal Imitator.

The aim of the selected effort was to develop an Imitator which is capable of learning to play a variety of solitaire type card games. The basic structure of the program was formulated and implemented. The program was written in IPL-V for execution on an IBM

7094. Tests resulted in the conclusion that the current system of the Imitator is capable of "learning" to play several restricted types of solitaire. The principal value of the present program is not in its current capabilities, but is its basic, flexible structure which provides for changes and extensions in the future. As some of the proposed extensions are realized, the ultimate goal of developing the concepts for use in formulating a general purpose Imitator will be approached.

Table of Contents

List of Illustrations

BLANK PAGE

Section I

Introduction

Since the advent of the computer some two decades ago, there has been an ever-increasing effort to extend the capabilities of these machines. By far the largest class of problems to which the computer has thus far been applied, is that class for which an algorithmic solution (one where the procedure is completely specified) exists. Thus, by and large, the computer has been used as a sophisticated calculator, neither learning anything from experience nor contributing guidance in the search for a solution, but merely performing exactly what the program told it to do.

If a computer were able to learn, a theory holds, then knowledge gained from past experience can be brought to bear on present problems and their solution made easier. Furthermore, it is held, that a completely new set of problems, not given to algorithmic solution, may be solved through the use of machines with such learning facilities. An example of some members of this set of new problems are certain games like chess, checkers, etc.

The attempt to develop machines which learn, has produced two different approaches. One is through the field of Bionics, where major emphasis has been to develop neuron models and neural nets. Effort has also been directed toward learning more about animal sensory mechanisms in the hope that it will yield knowledge as to the method by which living organisms recognize discrete data through their senses and process this data. The other approach has been to develop programs with a self-organizing capability. Such advanced programming techniques, often used to simulate hypothesized models of the psychological processes, have thus far given us our most significant results. We will now look more closely at the progress and problems of each approach.

1. The Bionics Approach

Probably because they play the major role in transferring, processing, and storing information in the brain, neurons have been the principle target of attack by investigators in the Bionics approach. Various schemes to simulate neuron action[1] have been attempted through the use of electronic components. These artificial neurons are then combined to form simulated neural-nets, originally unorganized. Organization occurs when a sensor connected to the net (the sensor may be real or simulated) supplies data from a certain perceived pattern. The nets have been shown to "recognize" certain learned patterns. Nevertheless, claims for these devices are being made with great caution, and we seem to be many years away from an element with an ability to learn a variety of patterns.

Of potentially great benefit to the development of artificial intelligence has been the investigation of animal sensors. Of these, the eye, which gathers an estimated 90% of information for the human brain has been under considerable scrutiny from researchers. Lindgren[2] describes various attempts to understand and simulate the mechanism by which the eye transmits data which enables the brain to classify and interpret sensed patterns.

Also of note is work similar to that done at the Westinghouse Electric Corporation on conditioned-reflex circuits. Although differing from the methodology of the "neuron" experiments in that their attempt was not to simulate an element of the brain but rather a process of these elements, Karl C. Wehr[3] and his associates were able to demonstrate that the system could learn to give the response "FOE" to a given airplane shape. Basically an experiment in Pavlovian conditioned reflex response, the model was simulated using polystable elements, each such element being a state defined digital sequential circuit.

2. The Software Approach

It can be reasonably assumed that the attempt to simulate cognitive behavior on computers had its beginning with the game-playing programs. Based on groundwork laid by Shannon[4] in 1949 and work done by Turing, Bernstein, the Los Alamos Group and Newell, Shaw, and Simon[5], the chess playing machine became a reality. Its main feature is a look-ahead procedure. Called "minimaxing". Essentially minimaxing involves the generation of a tree of possible moves and then a movement backward through the tree assuming that the opponent will make the best possible move to which the machine will reply with the best possible move, etc.

The same minimaxing procedure was used by Samuel's Checker Player[6]. This program, which was ultimately rated a "better than-average player" had several noteworthy features. It could rely either on rote learning (cataloging previous moves) or through generalization of what it had experienced. In the latter an evaluation polynomial was altered during play, either against a human opponent or against itself. Samuel concluded that test results showed rote learning to be more effective during the opening and the end game while generalization learning was more effective during the mid game.

In addition to their work on the chess program, Newell, Shaw, and Simon tackled the basic subject of problem solving and creative thinking[7], itself. By looking at most problem solving activity as a maze for which a solution tree may be constructed, they concluded that the solution of the problem will be obtained if every branch of the tree is examined. Since, however, for trees of formidable size, the time and effort required to examine each branch is so large as to preclude the discovery of the solution, we must use guides to channel our search along hopefully fruitful paths. These guides to reduce search, they labeled heuristics and their incorporation in problem solving programs as heuristic programming. Their ideas were then applied to two such problem solving programs which they called the Logic Theorist[7] and GPS[8] (for General Problem - Solving Program).

An interesting method by which the intelligence level of a machine may be increased is described by A. Hormann[9]. The system called GAKU after the Japanese word for learning, is based on a type of heuristic mechanism. Five such mechanisms are used, one of which coordinates the other four. Each of the four mechanisms performs one of the following functions: (a) construction of programs from basic operations and prestructured programs, (b) construction of sequences of legal actions to solve the problem (c) planning of the entire solution and subdivision into sub-tasks, (d) utilization of past learned experience in the solution of the problem.

Gaku is applied to an exemplary problem, the Tower of Hanoi Puzzle. This puzzle consists of three pegs and three doughnut-shaped disks, one larger than the next, all on the first peg. The largest disk is at the bottom; the smallest at the top. The object is to transfer this configuration to any of the other two pegs, in a minimum amount of moves, and by moving only one disk at a time. When the problem is solved Gaku is given the same problem with four disks. The objective is to have Gaku learn from the three-disk problem and to apply the gained experience to the four-disk problem.

Hormann explains that most of the research thus far has been on "primary" (by experience) learning. Future work will include "secondary" (by instruction) learning. She concludes that the success of Gaku is dependent on the solution of two basic problems: (a) the ability to communicate effectively with the machine, and (b) the development of an efficient method of generalization.

A new approach is being investigated for computer solution of problems which heretofore have been unprogrammable for digital computers and capable only of human solution. The approach is to develop an IMITATOR which can emulate human performance in a given activity. By gathering information from many examples of man's performance, the IMITATOR is able to evolve a model of the problem solving process used by man. Eventually the IMITATOR may take the form of a specialized digital device, although at present it is being simu' ted on the 7094 computer.

As an introductory effort, an implementation of the process of imitation has been developed. This initial formulation of the IMITATOR which will be capable of learning to play solitaire is described in Section II. The computer program for the simulation of this version has been written.

In developing this program it has been necessary to limit the types of solitaire card games the IMITATOR will be capable of learning to play by placing restrictions on such things as the number of decks which can be used, the attributes of the cards which can be considered, and the ways in which the cards can be moved in relation to the other cards. In future development of the IMITATOR the number of these types of restrictions will be considerably reduced.

Section II

The Imitator

As an introductory effort in the development of an Imitator, a study problem was selected which would aid in more accurately defining problem areas which would be encountered in the imitation approach to heuristic problem solving.

After a study on the general theory of imitation was completed, a study vehicle was selected to test and elaborate on the concepts that were developed and to discover the problem areas that might have been overlooked during the general study.

The aim of the selected effort was to develop an Imitator which is capable of learning to play a variety of solitaire type card games. Before describing the program, it should be remembered that it was not the intent of this study problem to develop solutions to all of the problem areas which are recognized. Indeed, some of the seemingly major problems were either over-simplified or assumptions (see Appendix A) were made which eliminate them for the present. In the development of the basic program for the study problem, the following restrictions pertaining to the play of the game were formulated: a) for each board situation there cannot be more than one proper play. b) there are no arbitrary plays. c) a play has the form of the movement of a card(s) from one board position to another position - included is the creation of a new board position when needed. Hence, any one play can, at most, effect the location of only the elements of two boards positions. In summary, the actions of shifting cards and creating or deleting board positions as needed are required to play solitaire of the variety being considered. The routines for performing these actions are preprogrammed and are referred to as action routines. Now for a description of the current program.

Observations which the program uses in imitating a man are given to the computer in the form of examples. Each example consists of two parts. The first of these two parts, which is called the situation, is the complete set of data needed to describe a particular problem. A situation in solitaire is the description of the board positions which are to be considered in determining the next play.

The second part of an example is an exemplary response. An exemplary response is the solution to the problem which is described by the situation. A response in solitaire is the description of the board positions after a play is made.

While the Imitator is learning, it is capable of generating answers to questions which have either been implied by (or stated in) a given situation. An answer which is generated by the Imitator is called a machine response. In solitaire a machine response is the description of the board positions after a play is made by the Imitator.

Since the Imitator is a learning program which is taught by the man who is performing a given task, the description of the relationship between the man and the Imitator is employed in explaining how the theory of imitation is used in the current study problem.

This relationship which exists between man and the Imitator is described in the following steps:

a. <u>Step One - Program Imitation</u>

To initiate the development of the Imitator, the man gives the machine an example of a problem which he has already solved. The example will consist of those elements as previously described.

b. <u>Step Two - Observation</u>

When the machine receives an example from the man, its function is to develop an imitation of the man's decision process used to obtain the given exemplary response.

For the study problem it is assumed that the situation is the lists of cards composing board positions being used to play the game. Each board position may contain a sublist of cards whose attributes are unknown. Only the number of cards in the sublist is included as information in the situation. This sublist is referred to as the sublist of unknown, unplayed, cards of a board position. Each board position may also contain a sublist of cards whose suits and numerical values are known and is referred to as the sublist of played cards. This sublist is composed of symbols such as C6 and S9 which would be the designations for the six of clubs and the nine of spades. The number of board positions which are being used is also included in the situation. Figure 2-1 (a) is an illustration of a situation during a solitaire game. Figure 2-1 (b) shows the corresponding exemplary results.

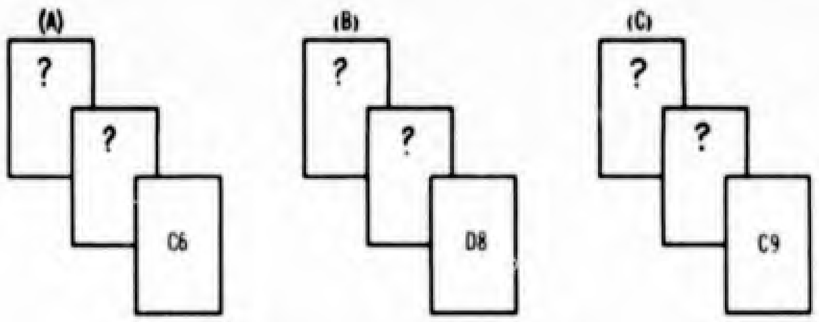

Figure 2-1(a). Situation

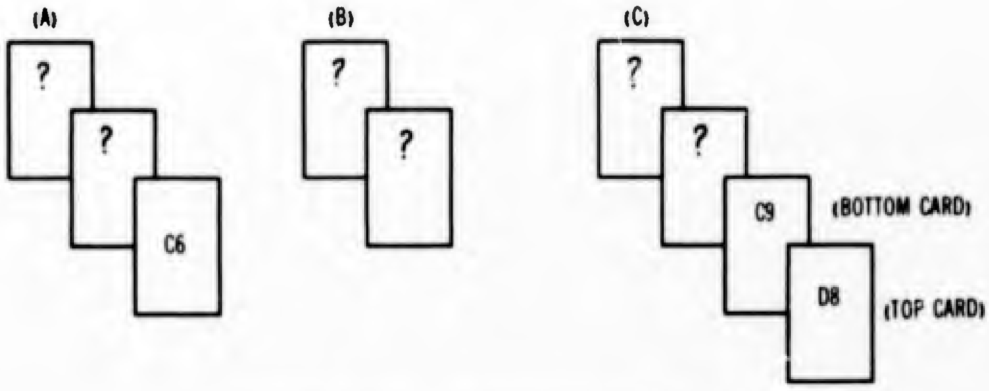

Figure 2-1(b). Exemplary Response

The next operation is to determine which of the board positions in the given example is the gaining board position. This is the position in the exemplary response which, when compared to the corresponding situation position, has additional cards in the played card sublist and/or an increase in the number of unknown unplayed cards. In the example, (c) is the gaining board position.

After determining the gaining board position, the machine compares the given situation and exemplary response to obtain differences which exist between the two states. Except for the first difference, only the gaining board position and its corresponding situation board position are considered in the computation. The differences are computed by subtracting values of attributes in the situation from the corresponding attributes of the exemplary response. Numerical values of one through thirteen are assigned to the ace through king, respectfully. Values are assigned to the suits as indicated in figure 2-2.

Suit	Value
Clubs	1
Diamonds	2
Hearts	4
Spades	8

Figure 2-2. Suit Values

The differences computed in comparing the situation and exemplary response are:

(1) Difference in the number of board positions.

(2) Difference in the number of unplayed unknown cards.

(3) Difference in the numerical values of the top cards.

(4) Difference in the suits of the top cards.

(5) Difference in the numerical values of the bottom cards.

(6) Difference in the suits of the bottom cards.

For the example given in figure 2-1, the resulting set of differences is: $\{0, 0, -1, 1, 0, 0\}$.

The following four parameters are also computed if the number of cards gained in the played card list of the gaining board position is more than one.

(7) Difference in the numerical value of the first card added to the bottom of the list and the old bottom card.

(8) Same as (7) with the suits being considered.

(9) Difference in the numerical value of the first card added to the top of the list and the old top card.

(10) Same as (9) with the suits being considered.

If parameters (7) through (10) are applicable then parameters (3) through (6) are assigned values of zero.

Three more parameters are computed when applicable. They pertain to the case where the played card sublist is empty in the situation, but contains at least one card in the exemplary response.

(11) Is played card sublist empty in situation and non-empty in the exemplary response?

(12) If (11) is yes and (2) is zero, find the value of the first card in the played sublist of the exemplary response.

(13) If (12) is not applicable and (11) is yes, set the value of (13) equal to one.

When parameter (11) is yes, parameters (3) through (10) are assigned values of zero.

Since parameters (7) through (13) are not applicable to the example in figure 1, the complete set of differences is: $\{0, 0, -1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0\}$.

The above differences and other values which result from processes using parameters from both the situation and the exemplary response are called action parameters.

The action parameters are used to determine what type of play has occurred. For instance, if the value of the top card of the sublist of played cards in a particular board position is different in the exemplary response than it was in the situation, then the top card either has a new location in the list of the same board position or it has been moved from a different board position.

The sets of action parameters computed when comparing situations with their exemplary responses are used in determining which action routines to employ during the course of play. These sets also provide the input necessary for execution of the action routines.

Sets of action parameters are classified and grouped in such a way that later, when they are compared to other parameters and the comparison is positive, an action routine is executed which results in a machine response to a given situation. A detailed discussion of the classification procedure and the function of the classification of the sets of action parameters is discussed in later paragraphs.

The formulation and classification of a set of action parameters is the function of Step Two.

c. Step Three - Man-Machine Communication

The results of Step Two are given to the man. The results include:

(1) The set of action parameters.

(2) The classification of the set.

(3) The action routine associated with the set.

(4) The machine response which would have resulted if the action routine had been executed, given the situation in the example.

(5) The results of comparing (4) with the given exemplary response.

Man surveys the results, and if he gives his approval the development proceeds to Step Four. If the man disagrees, it is necessary to take corrective measures before returning to Step Two.

In the current program there is only one corrective measure available. This is satisfactory, based on the present assumptions; however, this is obviously one of the areas which is over-simplified.

d. Step Four - Investigation of Situation

If the ability of the Imitator has reached a level of acceptable performance and the man intuitively feels that the continuation of the development by supplying the machine with another example would not appreciably change the ability of the Imitator to play a game of solitaire, then development proceeds to Step Five. The criterion for determining when the performance of the Imitator has reached an acceptable level is currently rather simple. The man being imitated can list all of the examples that are necessary for the machine to learn any given game, and since the present version does not have the ability to generalize, all of these examples must be given to the Imitator. Therefore, in deciding when an acceptable level is reached, a check is made of the man's list to determine if all examples are included. Let us hasten to add that a technique which will give the machine some generalization capability has been developed, but at this time has not been integrated into the present operational program. When the ability of the Imitator is extended to include this and other capabilities, a new set of criterion for performance will be formulated as experience is gained.

If the machine has not reached an acceptable level, then the man provides the machine with a new example. A pre-programmed routine is initiated which compares board positions of the situation. Values given in the lists which describe a situation and the results of methods performed on sets of data composed exclusively of elements of the situation are called situation parameters. The following situation parameters are computed:

(1) Difference in the number of unplayed unknown cards.

(2) Difference in the numerical values of the top cards.

(3) Difference in the suits of the top cards.

(4) Difference in the numerical values of the bottom cards.

(5) Difference in the suits of the bottom cards.

(6) Difference in the numerical values of the top card of the "losing" position and the bottom card of the "gaining" position. (Note: In comparing two board positions of the situation, the positions are referred to as the "gaining" and "losing" board positions. The "gaining" position is the one which would receive additional played cards if action were taken. The "losing" position is the original location of the moved cards.)

(7) Same as (6) except the difference is between the suits.

(8) Difference in the numerical values of the bottom card of the "losing" position and the top card of the "gaining" position.

(9) Same as (8) except the difference is between the units.

(10) Is the played card sublist of the "gaining" board position empty?

(11) If (10) is yes and the number of unknown unplayed cards in the gaining position is zero, and if the "gaining" and "losing" positions are different, then find the value of the first card in the played card sublist of the "losing" position.

(12) If (10) is yes and the "losing" and "gaining" board positions are the same, and if the number of unknown unplayed cards is positive, then set the value of (12) equal to one.

The above set of situation parameters are compared with each of the sets of action parameters that have been computed from previous examples. If no favorable comparison is made, the development proceeds to Step Two. When a favorable comparison is made, the situation is acted upon by the associated action routine. (For a discussion of the formation of associations refer to the later paragraphs describing the classification procedure.)

The resulting machine response is compared with the given exemplary response, denoting any differences which appear in the lists which describe the board positions. The results of this comparison are referred to as response comparison parameters.

The man then receives the following data from the machine.

(1) The machine response.

(2) The set of response comparison parameters.

(3) The set of action parameters used.
  (a) Classification of set
  (b) Action routine associated with set.

The man's action then follows those outlined in Step Three.

e. <u>Step Five - Game Execution</u>

For the action to have reached this step it is assumed that the Imitator has reached a point in development where it is now ready to play the game.

Problems (situations) are given to the machine. The situation parameters are computed and compared with the available sets of action parameters. If a negative comparison is found this information is given to the Man. This indicates further development is necessary. When a positive comparison results, the appropriate action is performed and the machine response is given to the Man.

Using these results the Man determines if the machine is performing satisfactorily. If at any time Man determines that the machine needs further development, action returns to Step Four.

In previous paragraphs reference is made to the associations between the action parameters, the situation parameters, and the action routines. The method used to form these associations is the classification procedure. It enables the Imitator to derive solutions to situations by using the information it has learned.

Due to the complexity of classification the following example is used to explain the operation of the method.

Assume a game of solitaire in which three board positions (a, b, and c) are being used. Each of the board positions contains a list of played cards whose numerical value, denoted with a number, and suits, denoted with a letter, are known.

(a)	(b)	(c)
H9 (Bottom of list)	D7	H7
S10	S8	C8
D11 (Top of list)		

The above board positions compose the situation which is used in determining the next play. However, in an example, the Imitator is also given the exemplary response, i. e., the condition of the three board positions after a play is made. The symbol Ø denotes that a board position is empty.

(a)	(b)	(c)
D7	Ø	H7
S8		C8
H9		
S10		
D11		

Referring to the previous sections of this paper which explain the development of the Imitator, the first action of the Imitator is the computation of a set of action parameters. These action parameters are differences found by comparing the given situation with the given exemplary response.

The first computation is to determine if the number of board positions being used in the situation and exemplary response is different. For the given example this action parameter has a zero value.

The next step is to compare the board positions of the situation with the corresponding board positions of the exemplary response. First check each pair to determine a board position which shows a gain in the number of cards. In this case, a gain of two cards is indicated for board position "a". The set of action parameters listed previously in the article is computed by comparing the situation and exemplary response of the board position which has gained cards, position "a".

For each difference other than zero a key value is assigned. The key value assignment is made from the following table.

Action Parameter Number	Key Value
1	2°
2	2^1
3	2^2
4	2^3
5	2^4
6	2^5
7	2^6
8	2^7
9	2^8
10	2^9
11	2^{10}
12	2^{11}
13	2^{12}

For the given example.

Action Parameter Number	Computed Difference	Key Value Assignment
1	0	
2	0	
3	0	
4	0	
5	0*	
6	0*	
7	-1	2^6
8	4	2^7
9	0	

* Differences are assigned values of zero because action parameters (7) through (10) are applicable.

Action Parameter Number	Computed Difference	Key Value Assignment
10	0	
11	0	
12	0	
13	0	

Add the key values to obtain the Key Number.

$2^6 + 2^7 = 192$

The set of non-zero differences with their identifying key number is stored in a list accessible for later use by the Imitator.

Now for a moment, attention is turned toward the available action routines. With each of the preprogrammed action routines there is associated a Primary Key Number. The primary key number for an action routine is determined by the action parameters which result with non-zero differences if the action routine is used to change a given situation.

An example of an action routine is: Move a multicard played sublist of one board position to the bottom of another board position. Such an action routine may result in non-zero differences for action parameters 7 and 8. Each of the action parameters has a key value, the sum of which is 192. Therefore, the primary key number associated with the above action routine is 192.

Also associated with each action routine is a list of secondary key numbers. Their use is discussed in subsequent paragraphs.

Returning to the present example and without describing the action, it is assumed that the processing as stated in step three was completed and the man found the results to be satisfactory. Further assume that a new example, similar to the previous one, is given to the machine.

Situation

(a)	(b)	(c)
H3	D1	H1
S4	S2	C2
D5		

Exemplary Response

(a)	(b)	(c)
D1	Ø	H1
S2		C2
H3		
S4		
D5		

After receiving the example, the machine computes the following set of situation parameters when it compares board position "a" to board position "b". (b-a)

Situation Parameter Number	Difference Computed
1	0
2	-3
3	6
4	-2
5	-2
6	-1
7	4
8	-4
9	0
10	0
11	0
12	0

Associated with each situation parameter is also a Key Value. This key value is determined by the relationship which exists between situation parameters and action parameters. There is a one-to-one correspondence between situation parameters and action parameters with key values of 2^2 or greater.

The Key Values associated with the situation parameters are:

Situation Parameter Number	Key Value
1	2^1
2	2^2
3	2^3
4	2^4
5	2^5
6	2^6

Situation Parameter Number	Key Value
7	2^7
8	2^8
9	2^9
10	2^{10}
11	2^{11}
12	2^{12}

After computing the situation parameters, the Imitator starts a systematic search through the list which contains all of the sets of previously stored non-zero action parameters with their identifying key numbers. Assume that the set computed in the first part of the example is the first set in this list.

The key number, 192 of the set indicates that a comparison between the action parameters with key values of 2^6 and 2^7 should be made with the currently available situation parameters which have the same key values.

The comparison is favorable since both sets of differences are identical. This would initiate a search for an action routine with an associated key number of 192. When the appropriate routine is located, the given situation is acted upon and the machine response is determined.

The resulting machine response and the given exemplary response is then compared and the man determines from the set of zero valued response comparison parameters that a correct action has taken place.

In the search through the list of the sets of non-zero action parameters each key number of a set indicates the subset of situation parameters to be used in the comparison. The time required for searching this list could become very lengthy. Methods are currently being developed to enable the Imitator to be selective in the sets which will be compared instead of the current method of checking each of the available sets.

In a later version of the Imitator, when it will no longer be necessary to pre-program the action routines, the key numbers which are associated with an action routine will also be determined by the Imitator. However, for the current effort is necessary to include an interim method for handling an important problem.

If only the primary key number associated with an action routine is used in triggering the routine, then the type of game which the Imitator can learn to play is restricted to those games which require non-zero differences in the comparison of all attributes which are pertinent to any given action routine. In the case of the previous example, both a difference in the suits and in the numerical values have to be present before the given action routine is used.

To overcome the above "non-zero difference" restriction, the following method is used.

Associated with each primary key number is a list of secondary key numbers. Each member of the secondary key number list is formed by summing one of the possible subsets of the key values used in computing the primary key number.

In determining the action routine to be triggered, the primary key number and the associated secondary key numbers of each action routine are considered. The secondary key numbers indicate which of the differences considered in triggering a given action routine must have a non-zero value. The primary key numbers are unique in their association with the action routines. This does not hold true with the secondary key numbers and this will result in the selection of incorrect action routines during the development of the Imitator. However, as the incorrect choices are made evident to the man through use of the response comparison parameters, the secondary key numbers erroneously assigned to the action routine are removed from the list associated with that routine. This is the corrective measure which was previously referred to in Step Three. In later versions of the Imitator all of the key numbers associated with an action routine will be unique to that action routine.

An Imitator as described in the preceding paragraphs has been programmed in IPL-V (Information Processing Language -V) for the IBM 7094. The program is capable of "learning" to play some solitaire card games. The following game was used in testing the playing ability of the program.

The rules of the game were:

(1) Use three board positions. Each board position has two sublists. One contains cards which are "face down" and is referred to as the unknown unplayed sublist. The other sublist contains cards which are "face up" and is called the played sublist.

(2) To play the top card from one played sublist on the top card of another played sublist, there must be a numerical difference of minus one and the color of the cards may not be the same.

(3) If a board position is completely empty and if there is a board position whose played sublist consist only of a king and whose unknown unplayed sublist contains cards, then the King may be moved to the completely empty board position.

(4) If a board position has an empty played sublist, but has cards in the unknown unplayed sublist, then the top card of the unknown unplayed sublist may be "flipped" to become a member of the played sublist.

The following three examples illustrates the above rules:

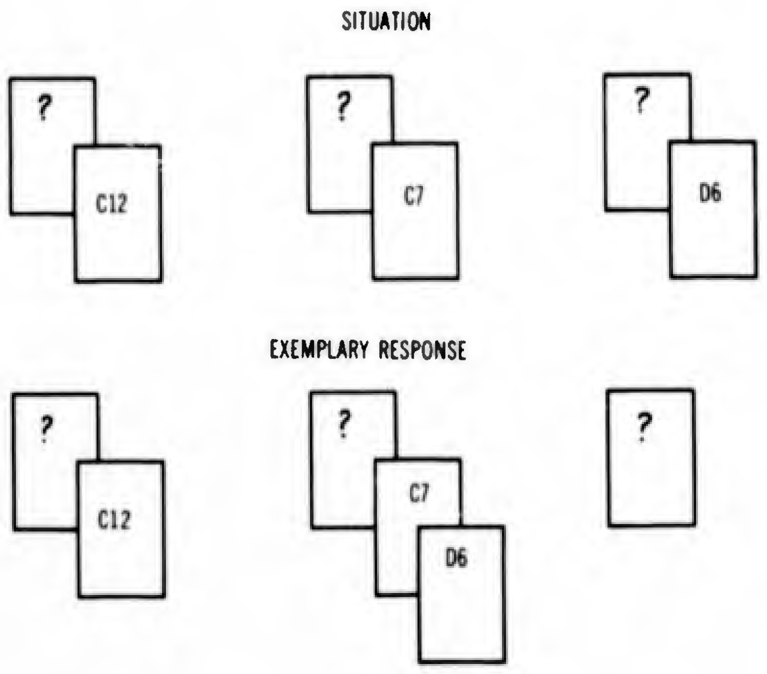

Figure 2-3. Example 1 (Rule #2)

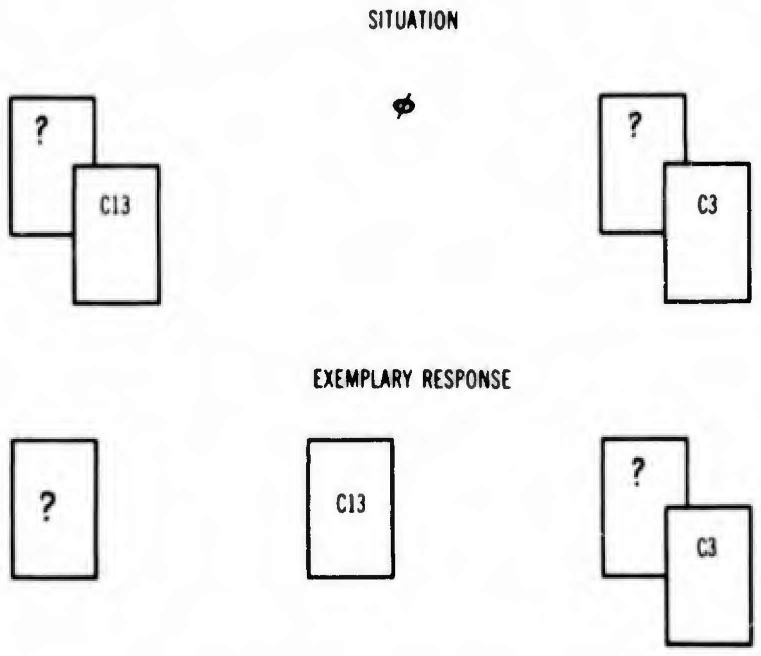

Figure 2-4. Example 2 (Rule #3)

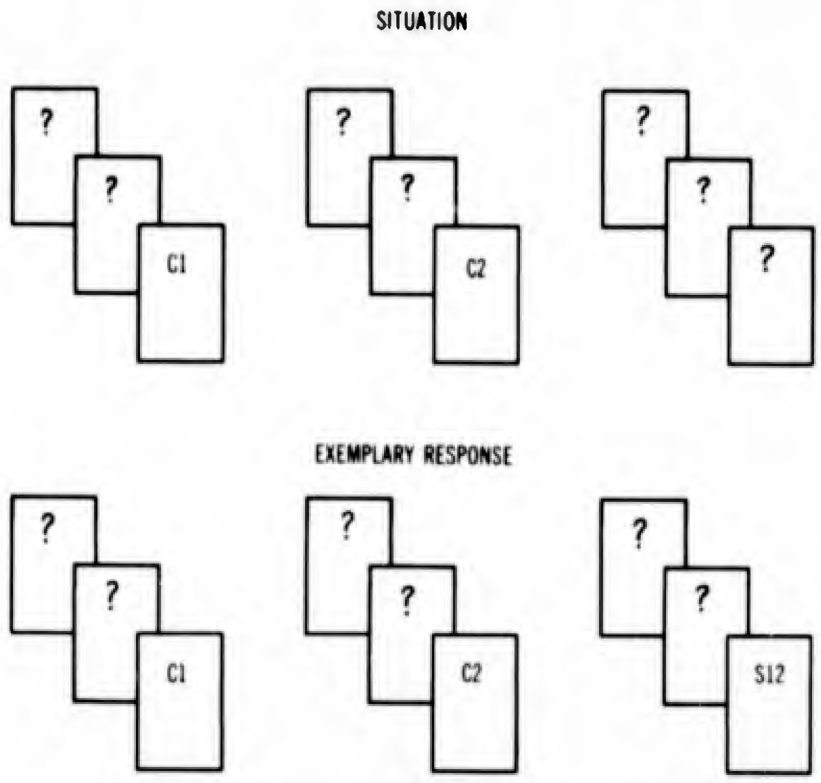

Figure 2-5. Example 3 (Rule #4)

Since the current program cannot generalize, it was necessary to give the Imitator a set of thirteen selected examples. The set selected was sufficient for the Imitator to "learn" to play the game. After processing the examples, the Imitator responded correctly to a group of random situations given to it by the man. Therefore, it was concluded that the machine is capable of "learning" to play the test game.

BLANK PAGE

Proposed Revisions

In developing the basic structure of the Imitator many recognized problems were oversimplified or assumptions made which eliminated these known difficulties. Also, many desirable capabilities were omitted in the initial version. However, in developing the basic structure, consideration was given to these problems and omitted useful capabilities by purposely designing this basic structure to allow for changes to be made without major alterations to the existing program. The two parts of this section discuss some of the changes and extensions that were considered.

A. CHANGES

An assumption was made in the initial version to limit the number of possible plays in any given situation to one. This assumption is not very realistic and changes were considered which would eliminate the need of this oversimplification.

The present program will search until it finds a possible move. When the first possible move is discovered, action to perform the move is initiated. In the proposed change, the program would store in a list, all possible moves found in searching completely through a given situation. If more than one move is found, a decision would be made by the machine as to which move is optimum. In order to make a selection, the program would rely on information gathered and processed by an additional routine. The description of a technique for establishing an heirarchy to be used in making optimum selections is discussed in part B of this section.

In the present Imitator, the action routines which perform the moves used in the play of the game are preprogrammed. Techniques have been developed and tested by others which are applicable to computer formulation of these same action routines. "Means-ends" analysis has proven to be very useful and may possibly be used in the development of a technique for machine-programming of action routines used in playing solitaire-type card games.

Appendix B shows a method by which programs may be synthesized from preprogrammed routines. The methods suggested in the report are also applicable to the area of machine-programming.

Although no final formulation for the solution of this problem is proposed, the investigations in this area point toward a successful conclusion.

Another problem which was previously neglected was the selection of the list of measurements to be taken when comparing board positions of the exemplary response and/or situation. It would be most desirable for the system to have the capability to select the measurements which are pertinent to a given problem. No plan was formulized which would give the machine this capability. However, the routines used in the generalization technique (described below) are throught to be partially applicable to the solution of this problem.

B. EXTENSIONS

Generalization is considered to be one of the most important attributes which a synthetic intelligent system must contain. The application of the theory of imitation without the ability to generalize would be nothing more than a glorified information storage and retrieval system.

The following is a description of a proposed generalization technique which can be added to the existing basic structure.

This method will use as its data base, the information which is being stored in the Action Parameter Triggering (APT) list. Initially man will control the machine's application of the generalization process. After test data pertaining to the rate of generalization is obtained, this control will probably be a function of the number of examples which has been presented to the machine.

When the generalization routine is called, the first operation will be to tally the number of occurrences in the APT list of all primary key numbers. For the tally to perform its intended function it will be necessary to make a change in the current version of the Imitator. Henceforth, each example will contain an identifying number. The first example will be assigned the number one and each subsequent example will be assigned a value of one greater than the last. As an entry is made to the APT list, a list associated with the entry will be created. This list will initially contain the number of the current example and the list will be referred to as the example number list.

If later, during the learning sequence, an entry is used to initiate a move and the results are verified (no new entry is made to the APT list), then the number of the example which resulted in the above action will be inserted in the example number list associated with that particular entry. Therefore, in tallying the number of occurrences of a primary key number, the example number list will be used to determine the number of times a particular APT entry has been used. The total number of occurrences of any primary key number will be the number of examples which appear on all of the example number list associated with any given key number.

A threshold is set which will determine if a key number has been associated with enough entries to the APT list to indicate that generalization may be possible. A temporary threshold of 20 is established at this time. This number was selected for convenience of use in the statistical method to be employed in the generalization system.

If there are no key numbers that occur at least twenty times, the generalization routine would be terminated. For each case where the threshold is satisfied, the following procedure is implemented:

For any given key number, there is a finite number of action parameters. The number of action parameters associated with a key number will determine how many working lists will be created. The example number lists of the entires in the APT list which are associated with the subject key number will be searched to determine the example with the smallest number. When this is determined the sublist of differences contained in the associated entry will be used for obtaining values to be inserted in the newly created working lists. The first value will be inserted in the first list, the second value in the last list, the example number lists will again be searched for the next smallest example number. The previously described procedure of adding values to the working lists will be followed, inserting the values at the end of the lists. This routine will be followed until all example numbers have been processed.

Figures 3-1 and 3-2 are examples of an APT list and the working lists created by the previously described procedure.

Figure 3-1. APT List

```
192 (Key Number)
     4 (Differences Associated
    -4 with Key Number)
ENL (Associated Example Number List)

         1
        23
192
     1
     4
ENL
     2
    25

12
     1
    -3
ENL
     3

192
    -4
     4
ENL
     4
    29

192
     7
     1
ENL
    11

192
     6
    -3
ENL
    12

192
     3
     6
ENL
    13
    27
192
    -8
     4
ENL
    14
    28
```

```
768
     1
    -6
ENL
     5

5122
     1
    13
ENL
     6
    24

192
     2
     4
ENL
     7
    26

3072
     1
     1
ENL
    15
    30

768
     1
     3
ENL
    16

192
   -10
     4
ENL
    17
192
    -7
     4
ENL
    18
```

```
48
    -1
    -1
ENL
     8

192
     5
     4
ENL
     9

192
    10
     6
ENL
    10

192
    -6
     4
ENL
    19
12
     1
     1
ENL
    20

192
    -3
     4
ENL
    21

48
    -1
    -4
ENL
    22
```

Figure 3-2. Generalized Working Lists

(a)	(b)
4	-4
1	4
-4	4
2	4
5	4
10	6
7	1
6	-3
3	6
-8	4
-10	4
-7	4
-6	4
-3	4
4	-4
1	4
2	4
3	6
-8	4
-4	4

The ordering of the examples is included to provide for the possibility of testing each of the working list for randomness by using either an auto-correlation technique or Goutereau's Constants[10]. When the number of entries in the working lists becomes large, these tests can be used to determine which of the lists should be processed. However, for a relatively small number of entries, the following procedure is used.

A tally is made to determine the number of occurrence of each difference which appears in the list. Since each of the working lists is made up of entries which are differences associated with a particular action parameter, the probability of the occurrance of any one difference can be computed. Having both the number of observed occurrences and the number of expected occurrences, a chi-squared test can be performed to determine if the number of times a difference appears on a list is significant. If there are no significant differences, the processing would move to the next working list. In the case where a significant difference exists, new working lists are created for further processing. The new lists are composed of the elements associated with the subject difference. The processing of working list (b) of figure 3-2 would result in finding that "4" is a significant difference. Two new lists would be formed consisting of the elements associated with the significant difference. The newly created lists are illustrated in figure 3-3.

(b)	(a)
4	1
4	-4
4	2
4	5
4	-8
4	-10
4	-7
4	-6
4	-3
4	1
4	2
4	-8
4	-4

Figure 3-3. Significant Working Lists

The first associated list is checked to determine if all of the entries are identical. If they are, processing of that list is complete and action would move to the next unprocessed associated list. If all of the entries are not identical, the corresponding original working list is checked to determine if any significant differences are included. Numbers which are not significant are marked as non-pertinent in the newer corresponding working list; significant numbers are left unchanged. Each of the newer working lists are processed in the above manner. The results of processing the example in figure 3-3 are illustrated in figure 3-4.

(b)	(a)
4	*
4	*
4	*
4	*
4	*
4	*
4	*
4	*
4	*
4	*
4	*
4	*
4	*

* Non-Pertinent

Figure 3-4. Processed Working Lists

The rows of the processed working lists are checked to determine if there are any rows, containing differences marked as non-pertinent, which are identical except for the marked values. If there are no such identical rows, then no generalizations result from the given value of the working list. However, in the case of the example, there is a set of rows which meet the qualifications. One row from the set is transposed and a sublist is formed which is associated with an identifying key number. This list is then entered into the Super-APT list. This entry is the result of the proposed generalization technique. The Super-APT list is used in the same way as the APT list. However, in comparing the entries in the Super-APT with situation parameters, those differences, which are marked non-pertinent are not tested for equality with the corresponding situation parameters. The partial results of the generalization process performed on the given example is illustrated in figure 3-5.

```
                Super-APT List
                      .
                      .
                      .
                 192
                      *
                      4
```

Figure 3-5. Partial Generalization Results

Each of the original working lists are processed by the above method. Duplications and contradictions will probably occur. After each complete generalization attempt, the man will be given the final results. A method will be provided for the man to delete redundant or incorrect entries from the Super-APT list. After the generalization technique is implemented and tested, and improvements are made, a method could easily be devised to allow for machine elimination of any redundant entries.

Small sample sizes from which the machine may possibly try to generalize will reduce the effectiveness of the proposed technique. However, experience gained in implementing and testing the proposed method should prove to be most valuable in formulating a revised version of a generalization technique for the Imitator.

In describing the first of the proposed changes, mention was made to a process which would establish a heirarchy of all the possible moves. This heirarchy would be used to select the optimum move when more than one possibility exists.

A complete formulation of an heirarchy system has not been developed but, consideration has been given to this problem and the following attributes are thought to be desirable in an heirarchy system.

The entries of the Super-APT and APT lists would be arranged in sets. Each set would contain those entries which are used to trigger moves that are either identical or weighted the same in relation to the desirability of their selection.

In turn, each of the sets would be arranged on the APT lists in the order of their selection desirability. Hence, after a set of situation parameters are computed and a list of all possible moves is constructed, the optimum selection can be made by determining the possible move that is found to occur first in the heirarchy. In the case where two or more possible moves occur in the same set, and that set occurs first in the heirarchy, the optimum move will be selected randomly.

The establishment of the heirarchy will be a function of determining what moves were possible in a learning example and noting which of the moves was selected by the man.

Section IV

Comments

The formulation, implementation, and testing of the study vehicle and the formulation of possible changes and extensions to the system required the greatest time and effort during this study.

Most of the results of other tasks as outlined in the proposed research plan[11] are implicitly, or in some cases explicitly, contained in the main sections. However, there are a few observations which should be made. These pertain to the definition of the roles of the computer and the human, and the communications between these two parties in reference to the theory of imitation.

When the goals of the Theory of Imitation are realized, the Imitator and the man should ideally perform the same function. But during the teaching-learning process they must each have definite duties. If the Imitator is less than ideal, the final use of the Imitator will find the man and machine sharing the duties of problem solving.

During the development of the study vehicle, the activity of the man (except for giving examples) was purposely minimized. In some areas of activities, where the machine is required to perform functions which could be greatly simplified if man could place certain information directly into the machine, it may seem that man's abilities are not being used beneficially. The purpose of requiring the machine to perform these functions is to gain experience in developing machine techniques which will be available when man is not able to supply some of the pertinent information.

Many portions of the program are preprogrammed by the man. It has been previously indicated that this preprogramming will be replaced by machine self-programming where possible. This is not an attempt to forget the man; by minimizing man's abilities in areas where machine performance is possible, and by using man only where necessary, a more useful system should evolve.

Once the Imitator has gained the ability to emulate man's decision process, than man aid can be provided to the Imitator to facilitate a more rapid machine convergence on man's decision process.

Another task as outlined in the original research plan raised the following questions. "How general should the corrections supplied by the man to the imitator be?" and "To what extent should the computer describe its decision process to the man?" In both cases it is desirable that the messages convey a very clear picture, yet the statements should be relatively simple. These goals are generally conflicting. The answer has been to communicate in clear pictures which are relatively simple to a person who is thoroughly familiar with the program and actions of the Imitator. This is not, however, a satisfactory solution to the stated problem. The problem can be solved only by the development of a natural programming language and compiler which provides the user with an implicit programming capability. This would allow the expert to communicate with the machine, using his own language. Hence the desirability of complete and "simple" statements could be achieved.

Conclusions and Recommendations

This study has given strong indications that the Imitator's chance of attaining its goal of duplicating the decision-making process of man can be successful if additional effort is expended. Although the present value of the Imitator, which was implemented and tested, is limited, its main usefulness is the basic structure which it provides for extensions that can be made. Although the proposed techniques of generalization and of establishing an heirarchy system have not been implemented, the possible capabilities which these would provide are considered most valuable in the attainment of the Imitator's goals.

A course of development is proposed which should bring these goals a step forward. After implementing the proposed changes and extensions, the Imitator would be tested to determine if it is approaching its desired capabilities. Intuitively, it is felt that many problem areas will be more clearly defined and the development of a new study program will be desirable. This is not discouraging. In fact, this is most gratifying. Experience has been gained in the development of a synthetic intelligent system and with this new-found knowledge man will be closer to establishing new man-machine relationships.

This study clearly adds to the ever-increasing evidence that the goals of developing useful systems, which demonstrate abilities that man considers to be intelligence, are not within easy grasp as indicated in the past decade. Likewise, it does not add to the proof of those who consider the goals to be either impossible or impractical. It does point toward a long and effort-demanding path which will be most interesting to follow.

Project Team

The principal investigator for this project was R. H. Forbes. Other Westinghouse personnel who contributed to this effort included:

J. E. Thompson	Fellow Engineer
W. C. Mann	Senior Engineer
P. A. Jensen	Engineer
G. M. Schechter	Associate Engineer

Section VII

Bibliography

A. PRIMARY REFERENCES

1. Corneretto, A., Electronics Learns from Biology, Electronic Design, September 14, 1960, p. 39. Survey article lists some of the present efforts in Bionics; describes neuron simulation efforts; brief description of Rosenblatt's Percepron.

2. Lindgren, Milo, Animal Sensors and Electronic Analogs, Electronics, February 16, 1962, p. 40. Concentrates on research dealing with the eye and gives brief description of several such experiments; describes artificial retina developed by RCA.

3. Wehr, Karl C. Conditioned Reflex Circuits, Electronics, August 24, 1964. Conditioned reflex machine which can be implemented with commercially available logic, is simulated on a high speed digital computer.

4. Shannon, C. E. Programming a Computer for Playing Chess, Philosophy Mag., vol 41, pp 356-375; March 14, 1950. Discussion of basic problems involved in playing a game such as chess.

5. Newell, A.; Shaw, J. C., and Simon, H. A., Chess Playing Programs and the Problem of Complexity. IBM J. Res. & Dev., vol 2, no. 4, pp 320-335; October, 1958. Traces the development of chess playing programs; refers to work of Shannon, Tuning, the Los Alamos group, and Bernstein; describes authors' own program.

6. Samuel, A. L., Programming Computers to Play Games, Advances in Computers, New York; Academic Press; p. 165 Good summary of various game playing techniques; the author's own checker playing program is treated more thoroughly.

7. Newell, A.; Shaw, J. C.; and Simon, H. A., 1958. The Process of Creative Thinking, RAND Corporation Paper p. 1320, Santa Monica, California. Compares creative thinking and problem solving; develops theory of Heuristics and discusses its value in problem solving systems.

8. Newell, A.; Shaw, J. C.; and Simon, H. A., 1958. Report on a General Problem Solving Program, RAND Corporation Paper p. 1584, Santa Monica, California. Describes GPS-I a general problem solving program, where emphasis is placed on separating problem content from problem solving technique; techniques discussed are illustrated by example.

9. Hormann, Aiko, Howa Computer System Can Learn, IEEE Spectrum, vol. 1, no. 7, July 1964, p. 110. Gaku, a learning program which learns to solve the Tower of Hanoi Puzzle is described; Gaku's role in primary (experience) and secondary (outside source) learning is discussed.

10. Davis, H. T., The Analysis of Economic Time Series, The Principia Press, 1941, pp 142-149.

11. Westinghouse Electric Corporation, Proposal for Research on Heuristic Problem Solving Machines, RNG 10068, January, 1964.

B. SUPPLEMENTARY REFERENCES

Bobrow, D. G., Raphael, B., A Comparison of List Processing Languages, RAND Corporation report RM-3842-PR, October, 1963.

Farber, D. J., Griswold, R. E., Polonsky, I. P., SNOBOL, A String Manipulation Language, Journal of the ACM, Jan. 64, vol. 11, no. 1, p. 21.

Findler, N., Some Further Thoughts on the Controversy of Thinking Machines, Cybernetics, vol. 1, 1963.

Gelernter, H., Hansen, J. R., Geherich, C. L., A Fortran-Compiled List-Processing Language, Journal of the ACM, vol. 7, no. 2, April, 1960, p. 87.

Gelernter, H., Rochester, N. Intelligent Behavior in Problem Solving Machines IBM Journal October, 1958.

Gordon, C. K., Jr., In Defense of the Search for the Pretercomputer, Cybernetics vol. 2, 1963, p. 57.

Green, B. F., Jr., Computer Languages for Symbol Manipulation, IRE Transactions on Human Factors in Electronics, vol. HFE-2 no. 1, March, 1961, pp. 3-8.

Newell, A., Tonge, F., Feigenbourn, E., Green, B., Nealy, G., Information Processing Language -V Manual, 2nd Ed., The RAND Corporation, Santa Monica, Calif., 1964.

Simon, H. A., Newell, A., Heuristic Problem Solving? The Next Advance in Operations Research, Operations Research vol. 6, no. 1, January, 1958, pp. 1-10.

Reitman, W. R., Programming Intelligent Problem Solvers, IRE Transactions on Human Factors in Electronics, vol. HFE-2 no. 1, March, 1961, pp. 26-33.

Samuel, A. L., Artificial Intelligence: Progress and Problems, Computers and Automation, March, 1963.

Simon, H. A., Newell, A., Information Processing in Computer and Man, American Scientist, Vol. 52, September, 1964, p. 281.

Slagle, J. R., A Heuristic Program that Solves Symbolic Integration Problems in Freshman Calculns, Journal of the ACM, Vol. 10, No. 4, October, 1963, pp. 507-520.

Szekely, D. L., On the Episternology of the General Purpose Artificial Intelligence, Cybernetics, Vol. 2, 1963, p. 57-75.

Appendix A

Assumptions

A. CHOSEN TASK ASSUMPTIONS

In proposing imitation as a solution to heretofore unprogrammable problems, the following assumptions have been made which arise from the nature of the chosen task.

1. The man has a decision process for dealing with the problem information.

The fact, that there are men who are experts in solving these types of problems, demonstrates that men have developed procedures for handling such problems.

2. The man's decision process is not modified during the development and operation of an Imitator.

Though it is expected that man will learn facts about his decision process which he feels should result in a change in the process, he will not modify it unless the current Imitator is terminated and the imitation of the revised decision process is initiated.

3. For all possible given situations, S, the man will make a decision D.

Given a situation the man will make a decision pertaining to the problem described in the situation. The decision may be of the form of a statement that more information is needed before a formal decision can be reached.

4. If the man's decision process is not probabilistic, then for each S there is an unique D.

When the man's decision process is deterministic there is only one decision possible for each situation.

5. The total number of examples to be used during the developmental stage of the Imitator before acceptable behavior is expected is finite. (See Step Four.)

The Imitator will converge on the decision process of the man during the developmental stage.

B. STUDY PROBLEM ASSUMPTIONS

There are also assumptions which have been made in designing the study problem. These assumptions are grouped according to their relative permanence. In later versions of the study problem it is planned to drop most of the temporary assumptions.

1. Relatively Permanent Assumptions

This group of assumptions are probably "here to stay" for the life of the current study problem.

a. Decision methods developed by the Imitator are presented to the man only in relation to specific examples, not in general.

Examples are not stored by the Imitator and are not used collectively to aid in the development of a decision method. The development of a decision method is affected only by one example at any one time.

b. Man is capable of recognizing bad responses which are generated by the Imitator.

When a non-optimum response is formulated for a given situation, the man is capable of determining that the behavior of the Imitator is not acceptable.

c. The only parameters effected by an action routine are those parameters that led to the use of the routine.

This assumption is necessary for implementing the method of classification used to interrelate action parameters, situation parameters and action routines.

2. Temporary Assumptions

The following assumptions are considered to be temporary.

a. Incorrect information inadvertently given to the machine by the man will be recognized by the man during the development of a decision process.

The Imitator is not capable of spotting errors in the man's decision process; therefore, it assumes the data given by the man to be unquestionably correct.

b. The probability that an erroneous decision method will fail during the developmental stage of the Imitator is 1.

If an erroneous decision method is included in the list of available methods, then before the development of the Imitator is completed, an example will be given to the machine which will result in the failure of the erroneous decision method.

c. Man is capable of determining the validity of a method generated by the Imitator.

When the Imitator gives to the man information pertaining to a decision method which is being developed, the man is capable of determining when the Imitator has been incorrect.

d. In the course of determining the validity of a method, man can recognize errors in the method.

The man may not necessarily pinpoint the exact cause of an error, but may only recognize the general area of the error.

e. The Imitator has the ability to recognize the situation parameters which are important to the determination of plays.

Situation parameters which would be redundant to the solution of a problem are not computed.

3. Ephermeral Assumptions

The following assumptions will be the "first to go" in later versions of the current study problem.

a. Comparisons made in the course of a decision process are prespecified.

The Imitator is given the information concerning which comparisons to make in developing a decision method.

b. For every board situation there cannot be more than one proper play.

In most games of solitaire it is possible for a situation to contain information which could lead to any one of several plays. However, for the current version of the study problem there is only one play considered, i.e., the first play the Imitator finds that it can make.

Appendix B

The Program Synthesist

I. SYNOPSIS

This report describes preliminary thoughts and their initial development of a basic and general system of imitation. The culmination is a semidetailed diagram of such a system.

Based on communications which describe man's decision process, the system will synthesize a program of this process, from a set of routines. Presently it is restricted to programs which use no single routine more than once.

II. THE NATURE OF ROUTINES

General Description

The word "ROUTINE", as used in this report, denotes a set of preprogrammed machine instructions for the addition, alteration, or deletion of information in the machine memory. For our discussion here, we will restrict ourselves only to routines pertinent to playing the game of solitaire. It is assumed, however, that the philosophy developed may easily be extended to most other subjects. In order to obtain generality and flexibility, the routines are written in symbols which are not dependent on any one problem or situation. Thus, if we want to remove the top card of some specific list of cards called List Z, we call a general routine which removes the top card of some Arbitrary List α. List α, of course, must first be equated to list Z. The sequence of instructions might then be as follows:

```
List α = List Z
Call "ROUTINE"
```

Henceforth, the concept of "List α " will be referred to as an internal representation for it is internal to the routine. The concept of "List Z " will be referred to as an external representation, for it is common only outside the routine.

The word "program" will be used in this report to mean a set of routines and instructions, linked together, so that they perform some meaningful function. The act of linking will be called Program Synthesis and the mechanism for doing this, The Program Synthesist.

Routine Storage and Selection

All routines to be used by the Program Synthesist will be stored in an area of memory called the Routine Hold (abbreviated RH). Based on information given to the machine, a mechanism known as the Routine Selector can select any one of the routines in the RH and cause it to be executed. Such execution will add, alter, or delete information in the memory.

Nodes and Linking

If any two routines are, in succession, selected by the Routine Selector mechanism and executed, the second routine to be chosen may be linked to the first routine. Henceforth, whenever the first routine is selected and executed, the second or "linked" routine will automatically be selected and executed. If some third routine is linked to the second, than it is selected and executed likewise, etc. By this method of linking, complete programs, consisting of linked routines, may be created or synthesized.

Figure B-1 shows three linked routines, called R, V, and X.

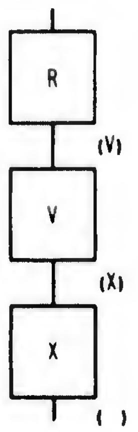

Figure B-1. Three Linked Routines

The (v) at the end of R and the (x) at the end of V, denotes respectively that V is linked to R and X to V. These two designations, called exit nodes, are said to be full because they contain the name of a linked routine. No routine is linked to X in figure B-1, and its exit node is said to be empty. Empty exit nodes are designated by (   ).

It follows then, that at the outset, before synthesis has begun, all exit nodes are empty. As synthesis proceeds and routines are chosen in succession, exit nodes fill with the names of succeeding routines.

The Flow of Information

Suppose we had two lists of cards Y and Z, one of which is longer than the other. If we wanted to remove the top card from the longer list, we could perform this simple program by using the two routines, A and B, shown below in figure B-2.

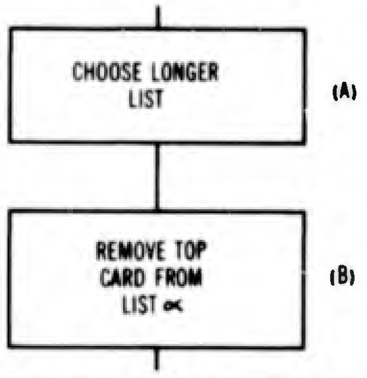

Figure B-2. Simple Program for Choosing Top Card of Longer of Two Lists

Ideally we would want the input to routine A to be the lists Y and Z and following the execution of A, the longer list to be List . Both of these and similar problems may be solved by the use of an area of memory which receives and contains the output of the last executed routine. Thus, after the execution of routine A, this area in memory would contain the name of the longer of the lists. This memory area will be referred to in this report as the Flowing Access Memory or the FX. At the finish of the execution of routine B, the FX will contain the top card of the longer of the lists. All routines, then, will obtain their inputs from the FX and write their outputs there. Each time information is written in the FX, all previous information is cleared.

To prepare for the possibility that the FX will not, at times, contain the proper information, a "SELECT" instruction is provided to allow the man to enter desired information into the FX. An example of the need and use of the "SELECT" instruction follows.

If, prior to the execution of routine A, the FX does not contain the desired names, i. e. list Y and Z, then these names may be placed there as follows. The instructions "SELECT LIST Y" and "SELECT LIST Z" erase all previous information and place the names of the intended lists in the FX.

The Classification of Routines:

Routines may be classified into two sets: Basic or Compound and Unconditional or Conditional.

Routines which are Basic are routines which cannot be synthesized from other routines. They must, of necessity, be placed into the RH by man. Compound routines, although they may be placed in the RH by man, can nevertheless be synthesized from Basic routines.

Routines may also be either Unconditional or Conditional. These two types are shown in figure B-3a and 3b, respectively.

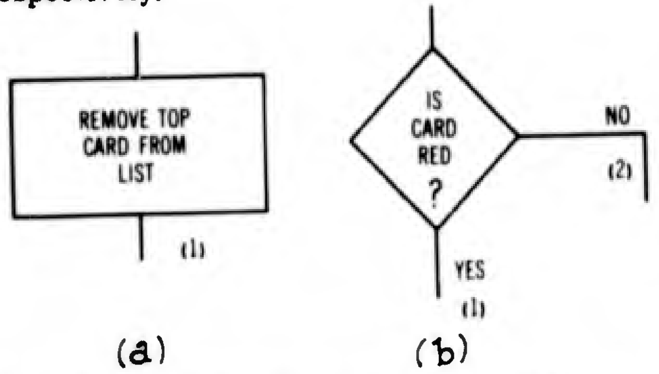

Figure B-3. a) Unconditional Routine, b) Conditional Routine

An Unconditional routine has only one exit node. It executes its assigned task and proceeds to its exit node regardless of existing conditions.

The Conditional routine, however, has two or more exit nodes and selects the proper exit node depending on the condition of the system. In figure B-3(b), for instance, the routine named in exit node (1) is selected if the card in the FX is red. If this card is black, the routine named in exit node (2) is selected.

Assortment of Routines in the RH

For the synthesis of some decision process, the Routine Hold, must contain at least all those Basic routines which will be used in such a decision process. Since, however, it is not practical or possible to anticipate exactly which basic routines will be used for some given process, the RH may be filled with an overabundance of Basic routines and the machine be allowed to select only the necessary ones.

III. THE SYNTHESIS MECHANISM

General Description

The Program Synthesist is an imitator which constructs programs of man's decision methods by "watching" man or following his orders or suggestions. This input information from man results in the successive selection of routines in a meaningful order. The routines are linked in the order they are selected. When synthesis is complete, the routines are executed in the selected order and thereby repeat the methods of man.

Operation may be divided into three modes; synthesis, execution, and tracing. During the Synthesis mode, the machine selects routines based on incoming information, and links these routines in the proper sequence. During Execution, the machine executes the linked routines in the proper order. The tracing mode is the same as the learning mode except that the machine goes, or traces, over old established links. Where the communications from man indicate a link which is different from the existing link, then the machine informs the man of this discrepancy. Such a difference may be resolved by breaking off the old link and substituting the one being dictated presently.

Routine Selection

Routines are selected on the basis of information supplied by man to the machine. Such a system would imply a method of classification of routines and a means of retrieval. Figure B-4 shows the general scheme.

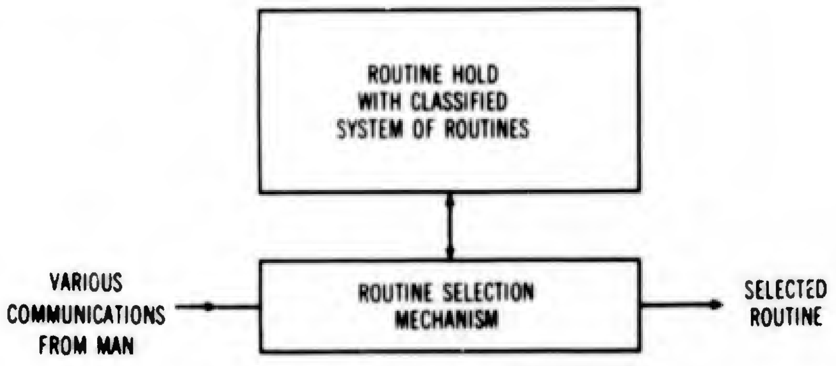
ROUTINE HOLD WITH CLASSIFIED SYSTEM OF ROUTINES
VARIOUS COMMUNICATIONS FROM MAN
ROUTINE SELECTION MECHANISM
SELECTED ROUTINE

Figure B-4. Routine Selector Mechanism

Of the various means of classifying the set of routines in the RH, the one dealing with routine function seems most promising at this time. A simple example of this is shown in table B-1. All routines in the first column deal with the numerical characteristics of a card,

TABLE B-1. AN EXAMPLE OF CLASSIFYING ROUTINES

Routines Dealing With Number	Routines Dealing With Color	Routines For Removing Cards	Routines For Adding Cards
A	B	B	C
B	C	X	X
C	H	F	Y
D	K		Z
E			

those of the second column deal with the color characteristics of cards, those in the third column deal with removal of cards from lists, etc. Thus, if the information from man indicates the need for a routine which removes the highest red (or black) card, then that routine would be B, for it is the only one having the characteristics of the first three columns. Conversely, the routine for adding the highest red (or black) card would be C.

Communications by man to the machine may be of several different types. These are:

1. Direct order: man tells the machine what to do. Example: "Select the top card from List Z".
2. Description: man tells machine what *he* has done. Example: "I have selected the top card from List Z." Essentially this is the same as 1 above, however, with an ultrasophisticated system the machine may forsake man's description and go off on its own, whereas it *must* follow orders.
3. Situation-Exemplary Response: man gives the machine a before and after picture. The machine tries to select the routine which will cause this transformation. Example: Before-picture AB C; after-picture A BC.
4. Selection Heuristic: man gives the machine some general information which may be helpful in the routine selection process. This may be given in addition to the communications described in 1, 2, and 3 above, thus speeding up the selection process. Often, however, the selection heuristic is the *only* information available. In this case the Routine Selector cannot choose *the* one routine to be used but must narrow the search down to all likely routines and then use the method of trial and error (5, below).
5. Trial and Error: man supplies one or more selection heuristics and the desired result. The machine executes routines from a selected list obtained through the use of heuristics. The routine which produces the desired result is consequently chosen.

Memory Banks: RH, TH1, TH2

In addition to the Routine Hold, already mentioned, two other memory banks play a vital part in synthesis and execution of programs. These are:

1. Temporary Hold 1 (TH1): Contains the name of the routine most recently extracted from the RH.
2. Temporary Hold 2 (TH2): Contains the routine selected immediately before the routine which occupies TH1. The routine in TH2 is always linked, during the synthesis mode, with the routine in TH1.

Essentially, during synthesis, the first routine selected from the RH moves to TH1. The second time a routine is selected, the first routine moves from TH1 to TH2, and the most recently selected routine moves to TH1. The routine in TH2 is linked to the one in TH1, for the latter has followed it.

In the execution mode TH1 receives the routine linked to the one in TH2 after which the latter is executed. The routine in TH1 is moved to TH2 and the cycle continues. Figure B-5 illustrates the three memory banks just discussed.

Detailed Operation

A more detailed explanation of the Synthesis, Execution, and Tracing Modes can be obtained from figure B-6 and the following description.

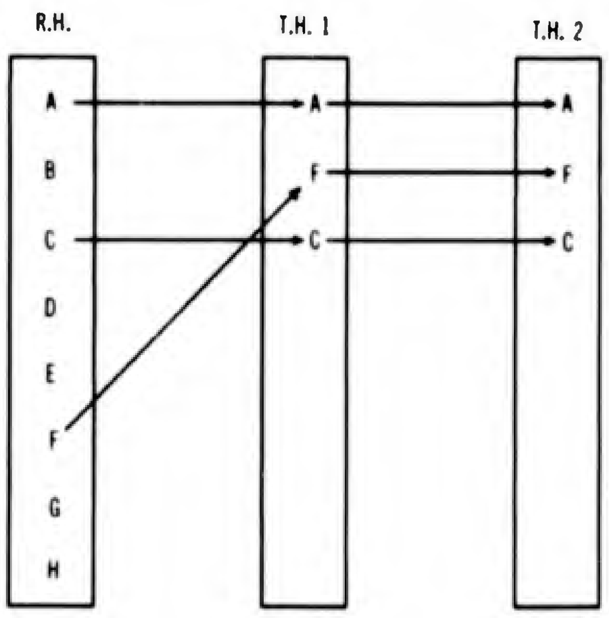

Figure B-5. Memory Banks

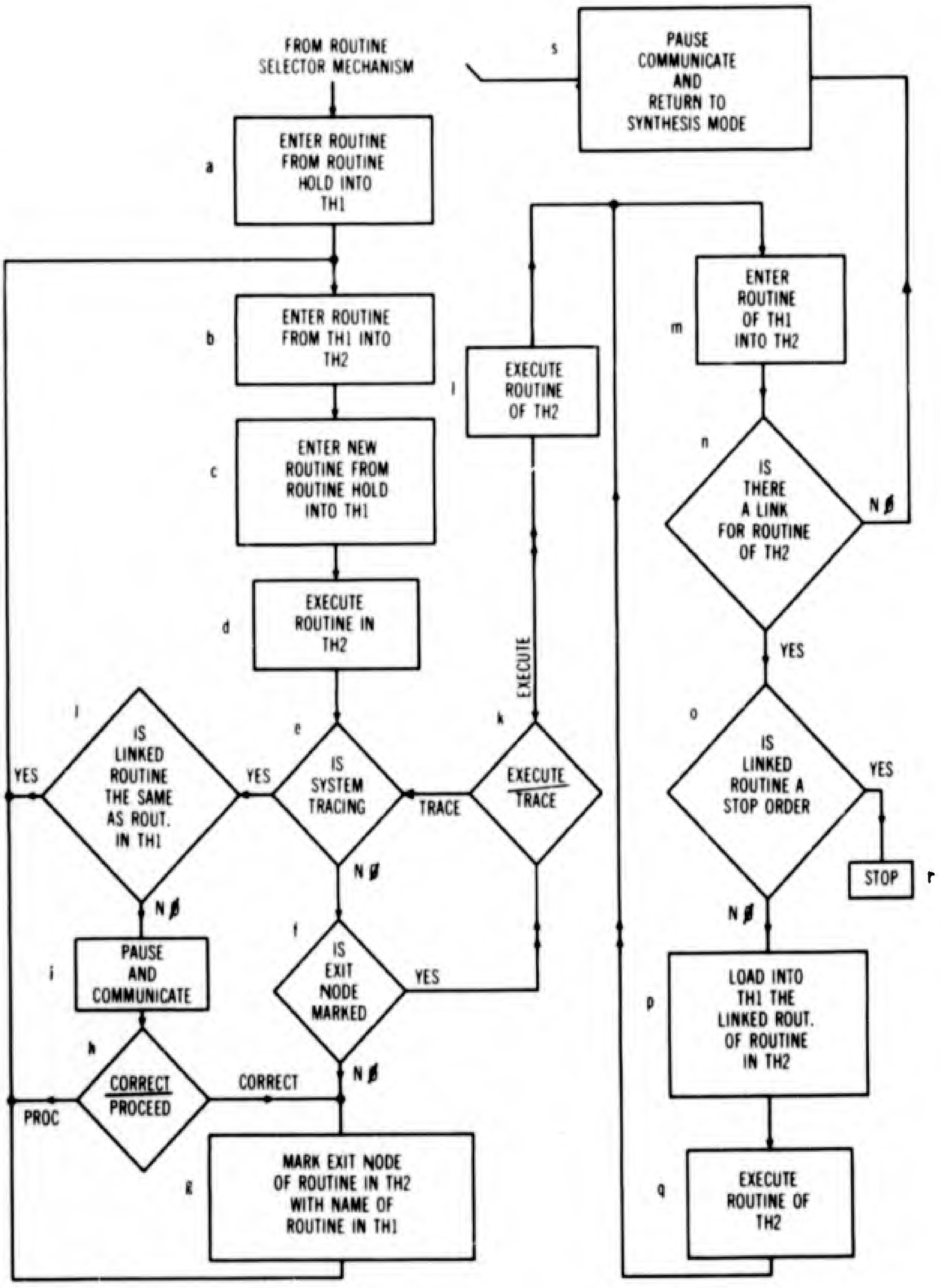
FROM ROUTINE SELECTOR MECHANISM
a
ENTER ROUTINE FROM ROUTINE HOLD INTO TH1
b
ENTER ROUTINE FROM TH1 INTO TH2
c
ENTER NEW ROUTINE FROM ROUTINE HOLD INTO TH1
d
EXECUTE ROUTINE IN TH2
e
IS SYSTEM TRACING
j
IS LINKED ROUTINE THE SAME AS ROUT. IN TH1
YES
YES
NØ
i
PAUSE AND COMMUNICATE
h
CORRECT PROCEED
PROC
CORRECT
NØ
f
IS EXIT NODE MARKED
YES
NØ
g
MARK EXIT NODE OF ROUTINE IN TH2 WITH NAME OF ROUTINE IN TH1
TRACE
k
EXECUTE TRACE
EXECUTE
l
EXECUTE ROUTINE OF TH2
s
PAUSE COMMUNICATE AND RETURN TO SYNTHESIS MODE
m
ENTER ROUTINE OF TH1 INTO TH2
n
IS THERE A LINK FOR ROUTINE OF TH2
NØ
YES
o
IS LINKED ROUTINE A STOP ORDER
YES
STOP
r
NØ
p
LOAD INTO TH1 THE LINKED ROUT. OF ROUTINE IN TH2
q
EXECUTE ROUTINE OF TH2

Figure B-6. Diagram of Program Synthesis

Synthesis begins when the Routine Selector mechanism, based on information obtained from man through the various means of communication, selects the proper routine, which we will assume is Routine A. The flow is now at a of figure B-6.

a. Routine A is entered from the RH into TH1. (See also figure B-5).

b. The routine in TH1 is moved to TH2.

c. A new routine is selected and moved from the RH into TH1. (This is F in figure B-5).

d. The routine in TH2 is executed.

e. The system is in the Synthesis - not the tracing - mode and the flow proceeds to f.

f. This is the first time that A has been used. Its exit node is empty (not marked). The flow goes to g.

g. The exit node of A is marked with the name of the routine in TH1, i. e. F. A is now linked with F. The flow returns to b.

Let us assume now that the machine, in linking A to F, is merely repeating something it already knows, i. e. A had previously been linked to F and the exit node of A is already marked, with F. Returning to f:

f. The exit node of A <u>is</u> marked. The flow goes to k this time.

k. We decide, at k, to go into the tracing mode, wherein the machine will check to see whether the "old" link is the same as the link being dictated now. The flow proceeds to l.

l. The system <u>is</u> tracing.

j. The exit node of A <u>is</u> marked with F. F is then the linked routine of A. The system dictates that the routine which should be linked to A is the routine in TH1, also F. This implies that the "old" link is in agreement with the newly dictated link. The system remains in the tracing mode and the flow returns to b. Had there been disagreement, the machine would have communicated this discrepancy in i, and in h the man would make the choice of "correcting" the discrepancy by going to g and creating a different link or ignoring the discrepancy and proceeding to b.

Upon finding at f that the exit node of A was marked, man could have decided that he had a high degree of confidence in the links and no need to trace presented itself. Instead he wished the system would run through the synthesized parts of the program - those parts which already have established links - but stop at the first point where no link existed so that the synthesis process might continue from there. This choice could have been made at k.

k. The man decides to go into the execution mode and the flow goes to l.

l. The routine in TH2 is A. This routine is executed.

m. The routine in TH1 is F. It is entered into TH2.

n. Assume F is linked to C, which is not an order to stop.

o. Routine C is not an order to stop.

p. The linked routine of the routine in TH2 (which is F) is the routine C. C is loaded into TH1.

q. The routine in TH2, routine F, is executed. The flow returns to m, where routine C is moved to TH2 from TH1. The cycle continues as before.

If there is no link for C, then at n the machine will go to S, pause, communicate that there is no link, and return the machine to the synthesis mode so that a link may be established. Had the routine which was linked to C been a STOP order, then the flow would have proceeded to r and the machine would have stopped.

IV. RESTRICTIONS AND PROBLEMS

The most severe problem results from the restriction that programs to be synthesized must be such that no single routine is used more than once. Surely if some routine, Z, were used more than once, then a link of "Z" would be ambiguous. With added sophistication, however, it is believed that the machine could discriminate between several Z's.

An additional problem area is that of routine selection. Where the list of routines becomes fairly large and the distinctions between routines very small, the Routine Selector will need to become an ever increasingly sophisticated device. We restrict ourselves presently to situations where selection can be made easily and correctly.

Thirdly, the methods of information flow will need to be improved far beyond their present state. It would be of great advantage, whenever the contents of the FX are not of the proper type or number, to have the machine recognize this fact and report it to the man. At the present stage we must restrict ourselves to situations in which the FX will always contain the correct information.

V. SUMMARY AND RECOMMENDATIONS

The system described has the following advantages:

1. It is very basic in nature and therefore lends itself nicely to changes which increase its sophistication.
2. It employs a man-machine relationship whereby man supplies his intelligence wherever the system is weak. With each upward step in sophistication, the man is relieved of some of this burden. The price paid is added complexity.

3. It is built on the concept of generality. Although solitaire was used as an exemplary problem, the basic system would be the same for any other problem. All that is needed in other situations is a set of pertinent basic routines.
4. The capabilities of the machine increases with its age and experience. Since basic routines are often combined to form compound routines, and these compound routines may be stored in the Routine Hold, then an "older and more experienced" machine will contain a larger number of compound routines, thereby saving the time needed to construct compound routines, first.

It is recommended that the principles of operation, thus far developed, now be tested by simulating the system on a digital computer. In this way unforeseen problems may be discovered as well as the assumed principles verified. A clearer course of direction should result.