Tony Leal

TM-1524/000/00

Gaku: An Artificial Student of Problem Solving

16 September 1963

TM-1524/000/00

TECHNICAL MEMORANDUM

(TM Series)

This document was produced in connection with a research project sponsored by SDC's independent research program.

Gaku: An Artificial Student of Problem Solving

Aiko M. Hormann
Stuart S. Shaffer
Theodore A. Van Wormer (Consultant*)

September 16, 1963

SYSTEM DEVELOPMENT CORPORATION
2500 COLORADO AVE.
SANTA MONICA
CALIFORNIA

*Currently at University of Pittsburgh

(page 2 blank)

ABSTRACT

This paper describes research whose main objective is to construct, by programming on a computer, an intelligent learning system capable of handling a set of varying and increasingly complex tasks which, when performed by a human being, are usually said to require intelligence.

Behavior of the system is determined by both a direct and an indirect means. The former involves detailed, explicit specification of responses or response patterns in the form of built-in programs. The indirect means is supplied by mechanisms which themselves are built-in programs but which are capable of collecting, organizing, and transforming information as well as generating and modifying programs under a set of conditions including interactions with the system's environment. The information and programs thus generated or modified in turn influence subsequent action (overt or covert) of the system. The mechanisms are used to supply a basic framework which would provide potential capabilities in a variety of situations.

An attempt to coordinate three mechanisms, each working in a different phase of problem-solving activities, is described in the context of the system solving a specific sequence of tasks. A problem-oriented mechanism is responsible for manipulation and generation of sequences of "unit actions" defined by a given task environment; its actions directly determine the behavior of the system. Influencing force on the problem-oriented mechanism is exerted by a planning mechanism by designating a rough sketch of a possible course of action. The problem-oriented mechanism is also influenced by an induction mechanism which takes a still larger view by attempting to apply the system's past experience with various problems to related problems which have not been previously encountered.

学 GAKU[1]: AN ARTIFICIAL STUDENT OF PROBLEM SOLVING

INTRODUCTION

The ultimate goal of the research described in this paper is to construct, by programming on a computer, an intelligent learning system capable of handling a curriculum of varying and increasingly complex tasks which, when performed by a human being, are usually said to require intelligence. Although we try to adopt some techniques through observing human problem-solving and learning activities, we do not hesitate to incorporate some "inhuman" machine characteristics which might be advantageous for our purpose. The features we consider and the techniques we employ are being tried on a highly experimental basis; we are not sure of their workability or desirability, separately or together. As a result we anticipate many trials and modifications. Before we can build a good learning system, there is a great deal of learning we must do. Gaku is the name given to the first generation of our learning systems.

Our current endeavor is concentrated on a class of puzzles with a sufficient number of restricting features to initiate the "artificial environment" of the Gaku system at manageable size and complexity, yet to place some demands upon Gaku's versatility. We have tried to separate learning processes and problem-solving techniques from specific problem content in order to

[1] Gaku is a combining form in Japanese denoting learning; its character is shown to the left of the title.

achieve generality, i.e., to achieve a system capable of performing in a wide variety of learning and problem-solving situations. At the outset, however, we use rather specialized interpretations of "abstraction," "generalization," "categorization," "planning," and "induction" in terms of machine performance. Our approach is to commence with specific, definite interpretations capable of implementation through programs and to generalize these interpretations as research proceeds. The alternative of starting with more general interpretations makes realization in programs much more difficult, if not impossible.

Our programming is being done in IPL-V on the Philco 2000 computer at the System Development Corporation. Some basic knowledge of IPL-V will help in understanding the last two sections of this paper.

GENERAL SCHEME OF THE SYSTEM GAKU

In previously published papers (Hormann, 1962, 1963), the general schema for constructing the system is described; we shall summarize it here in the first section. The rest of the paper will be focused upon features and experiments within the particular context of a class of board-and-counter puzzles.

Direct and Indirect Specification of System Behavior

Behavior of Gaku is determined by both a direct and an indirect means. The direct means involves detailed, explicit specification of responses or response patterns in the form of built-in programs. The indirect means is supplied by mechanisms which themselves are built-in programs but which are capable of collecting, organizing, and transforming information, as well as

generating and modifying programs. The information and programs thus generated or modified, in turn influence subsequent action of the system. The purpose of having such mechanisms is to supply a basic framework which will provide potential capabilities in a variety of situations. Inasmuch as the initial input and feedback are provided by the environment, the behavior of a mechanism at any particular instant, is not determined until it is given concrete objects (programs and/or data) to manipulate in a specific context.

Four mechanisms are used to enable the total Gaku system to work effectively: a community unit responsible for manipulation and generation of programs; a problem-oriented mechanism whose function is to be defined by a given task; a planning mechanism which takes a larger view of a given task and guides the problem-oriented mechanism by designating a rough sketch of a possible course of action; and an induction mechanism which takes a still larger view by attempting to apply Gaku's past experience to related problems not previously encountered.

These mechanisms represent our way of telling Gaku implicitly how to solve problems. Instead of being told explicitly how to solve a particular problem, each mechanism is given general rules for making decisions within its assigned level of activities, so that a discriminating search through a space of solution attempts can be carried out which would have a better probability of early success than random or exhaustive search. This indirect specification of Gaku's behavior through special purpose mechanisms is our approach to problem-solving in which discovery of means of solution requires search and experimentation whose course is not clearly known in advance.

Built-in Feature Common to all the Mechanisms

Each mechanism is designed to apply a cycling process which is useful in many problem-solving situations where a problem solver is often forced to make a decision or take an action with insufficient information about how to solve a given problem. As the situation unfolds, previously inadequate actions may be corrected or adjusted by utilizing additional information which becomes available as a consequence of the previous action.

The cycle adopted here in each mechanism passes through three phases: an analysis and test phase, a tentative selection or correction phase, and a consequence generation phase. Figure 1 depicts schematically how three of the four mechanisms are interrelated and how the three phases in each are tied together. Note that both input and output of each mechanism must go through the task analyzer whose function is to carry out the analysis and test phase of the cycle; all important decisions are made during this phase but within the boundary of authority specified by the mechanism coordinator, which allocates efforts and delegates corresponding authorities to its constituents.

The mechanical trainer in the environment (Figure 1) represents a program which has comprehensive information about given problems; the trainer is stored outside the Gaku system in the computer memory; its purpose is to eliminate human intervention as much as possible in order to use computer time efficiently. Use of the trainer is possible because, for a few experiments, we intend to use puzzles which have been completely solved by humans; later we will try puzzles for which human beings do not have complete solutions. For later experiments, we anticipate some actual interactions between a human trainer and Gaku.

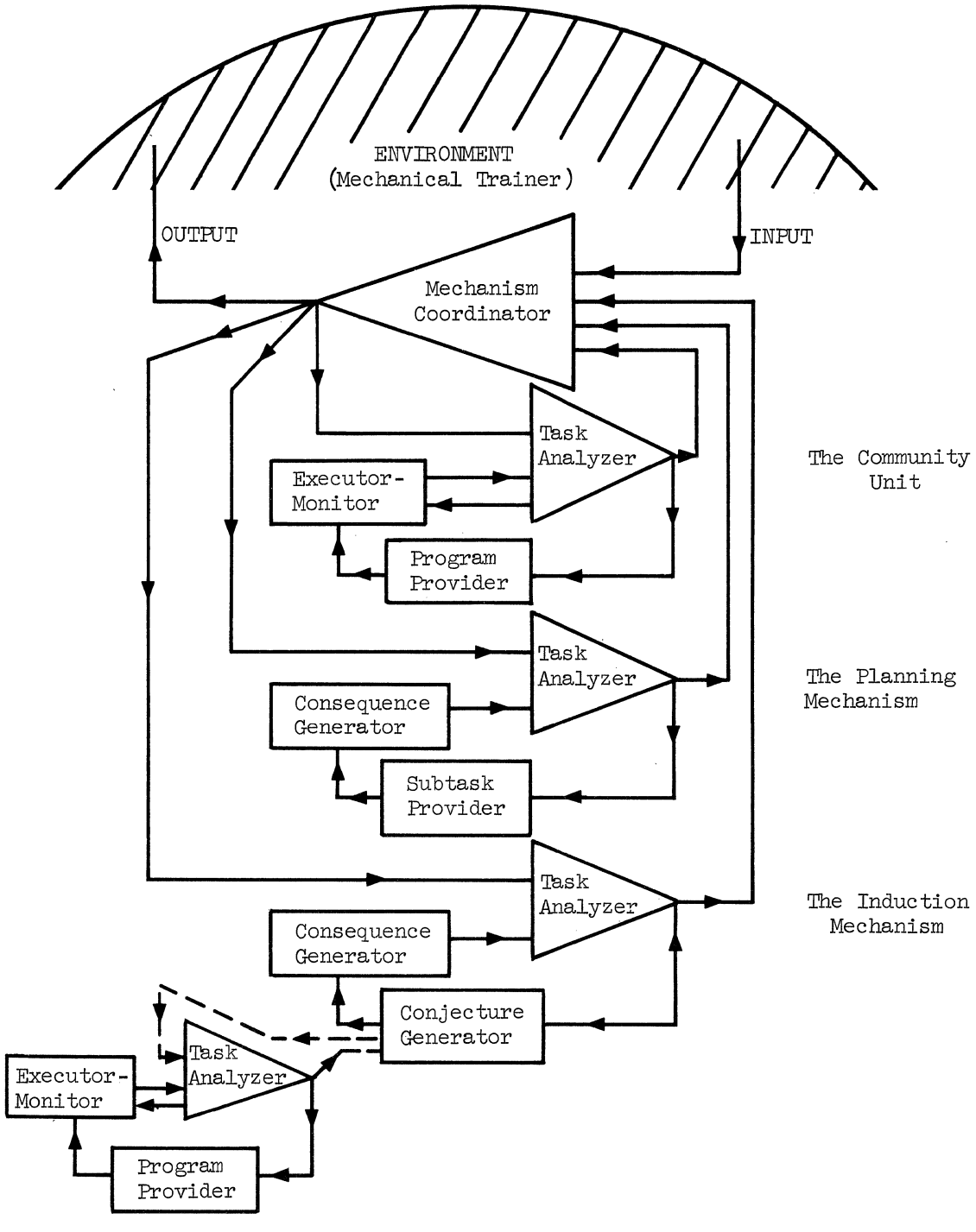


Figure 1.

Three Mechanisms Coordinated

Built-in Features which Characterize each Mechanism

In addition to the common (three-phase cycle) feature, each mechanism is supplied with a set of conventions, although they vary according to the purpose of the mechanism. For example, the selection phase of the community unit is carried out by the program provider which, following the task analyzer's direction, selects a set of concrete objects (operations and subroutines) from its repertory and proposes it to the executor-monitor which does the consequence generation phase of the work. The double arrow shown (Figure 1) between the executor-monitor and the task analyzer represents their interactions and forms what we call the first-order feedback loop within the outer loop. This loop is peculiar to the community unit because the objects which are manipulated by this unit directly influence the system's behavior (while the other mechanisms do not) and so are handled by two special modes of the operation phase. The details of the community unit and analogies drawn with human activities are described in "Programs for Machine Learning" (Hormann, 1962).

Concrete objects manipulated by the planning mechanism are state descriptions which represent stepping stones or intermediate nodes that fill the gap between two initially given nodes; finding a solution can be thought of as establishing a "valid" path between the two nodes. The state descriptions, which are stored in the form of lists with their description lists (Newell, 1961), can exist at various levels of abstraction depending on different planning stages.

The induction mechanism also has its own conventions and techniques; an example is the subunit, which is like a restricted community unit. The subunit is attached to the conjecture generator of the induction mechanism (Figure 1).

The Problem-Oriented Mechanism

This mechanism was originally intended to be generated by the community unit, although its framework (three-phase cycle) and the move-tree generation program are pre-established. When a description of a new puzzle is input to the system, the community unit is to provide, according to the given specifications, a legal move generator program which specifies legal moves, and a move-making program which produces a new configuration of the puzzle.

This type of sophistication has been temporarily dropped from the system; instead the problem-oriented mechanism is to be furnished from the outside with the task-defining information needed to perform its function. The reason for this is two-fold: First, our experience with a simple version of the community unit indicates that more research is necessary to achieve internal representation and manipulation schemes for efficient handling of some complex task descriptions where many relational concepts are involved; second, there are more immediately pressing difficulties even in a small class of puzzle-solving problems, and their resolution may provide clues leading to schemes which will handle complex task descriptions.

Decision-Making, Internal Communication, Tree-Growing, and Characterization

Another built-in feature of the system is that decision-making authorities, together with decision-making rules and criteria, rules for changing

criteria, etc., are present at various points in the hierarchy of the system structure. To execute the decision-making scheme properly, means of internal communication between mechanisms and parts of mechanisms are provided by a set of signal conventions and routines which discriminate between signals and follow a course of actions appropriate to the signal and the situation in which it occurred. Similar features are found in the General Problem Solver (Newell, Shaw, and Simon, 1959, 1960).

There are also tree-growing and tree-locating routines of various kinds. They include a past-experience-record tree, a substate-tree used by the planning mechanism, and a move-tree generated by the problem-oriented mechanism.

Finally, the most important, and yet least developed, part of Gaku is a group of routines for abstraction and characterization (of tasks and situations) which ultimately determine the feasibility and reasonableness of the planning strategies, and of methods used to evaluate difficulty of a task and performance of Gaku. Serious study of the subject has been and is being made by Amarel (1962), Minsky (1957, 1961), and Newell, Shaw and Simon (1959, 1960), among others.

Since these features mentioned above will become more meaningful in a particular problem-solving context, details will be discussed and some difficulties pointed out later.

CLASS OF PROBLEMS CONSIDERED

We shall consider board-and-counter games for which the required solution is a sequence (often specified to be a shortest sequence possible) of legal moves (or admissible operations) which start with the given initial state and reach a desired state. These sequence-seeking problems are contrasted with enumeration problems (where the task is to discover how many possible ways there are to do something) and other board-and-counter games such as a magic square where the final arrangement of counters (numbers can be thought of as counters with numbers on them) on the board must satisfy a certain requirement.

The selection of the type of problems is based on our belief that there are some principles in common between the sequence-seeking activities found in puzzle-solving tasks and those observed in theorem-proving and programming tasks. We hope to find such underlying principles and operational methods which are generalizable and transferable to a wider variety of problem situations.

For our convenience, we further restrict this class of problems. The restrictions are:

1) Number of players: We have been working with one-person games or puzzles rather than competitive games in which two or more players are involved.
2) Dimension of the board: Both one and two dimensions are allowed.
3) Characteristics of the board: Size and configuration of the board is determined by the number of cells and the links connecting them, each

link between two cells indicating immediate neighbor cells. Each cell is given a distinct name (e.g., C1, C2, . . ., Cn) and may or may not be characterized by an attribute, A100, which can take on values KØ (special symbol for empty or unoccupied cell) and the name of the counter occupying the cell. More than one counter may occupy one cell; the cell is then called a "stack cell" and its value of attribute A100 is the name of the top counter. One other attribute, A101, may be introduced to each cell; some cells, for example, may be black and some others may be red, and legal moves are often specified in terms of the cell color. The size and the configuration of the board is fixed for each game but to provide variations, different boards may be used for the same puzzle.

4) Number of counters: Currently the number of counters must remain the same during one game. We may later allow the number of counters to decrease. Examples of games with decreasing counters are jump-and-remove-a-peg, checkers, and chess. Examples of games with increasing counters are three-man's morris, gomoku, and hex. The game of go uses both increasing and decreasing counters.

5) Characterization or attributes assigned to counters: Each counter is given a distinct name and may be characterized by up to two attributes, A102 and A103. We shall first consider puzzles in which each counter is uniquely identified (n attribute values for n counters), e.g., each counter may be numbered 1, 2, . . ., n. In this case we usually use such attribute values as names of counters. Later we shall consider the possibility of having two or a few more attribute values for n counters

thereby dividing counters into groups, e.g., red counters and black counters. Since a maximum of two attributes are allowed, combinations of both can result in many varieties; one example is ①$_b$, ②$_b$, ③$_w$, ④$_w$, meaning black counter ①, black counter ②, white counter ③ and white counter ④, respectively. Each counter, once given attribute values at the beginning of a game, must retain those values during that game. This restriction may be contrasted with a pawn in chess game becoming a queen (or other value).

6) Legal moves: Usually conditions under which a move can be made legally are expressed in terms of attribute value(s) of the counter being moved and attribute value(s) of the cell to which the move is made. The conditions can also be expressed in terms of attribute value(s) of the "from cell" and attribute value(s) of the "to cell." One simple prerequisite usually stated is that the "to cell" be empty. Another restriction may be that the cell be a certain color. In case of a stack-cell, the top counter in the cell (attribute value of the cell as previously defined) may be restricted to be of a certain kind.

7) Task specification: Currently, in specifying a task, the initial state and the desired state of the counter arrangement on the board must be explicitly given. Therefore, if more than one goal situation is possible, all the possible states must be listed. Since we consider only finite games, we can always meet this requirement. However, it is not always convenient or desirable to exhaustively enumerate the goal conditions.

The desirability of representing goal conditions (or even intermediate positions) in terms of relative positions of counters is apparent in complex games, such as in chess when describing checkmate conditions.

8) Number of pieces moved at one time: We allow only one counter to move at one time.

The restrictions above are mainly for separating, at least partially, problem-solving techniques from problem-defining techniques. We are painfully aware that adequate techniques in both areas are necessary for more sophisticated learning systems. However, we are currently focusing our efforts on flexibility in problem-solving techniques and fix problem-defining conventions to be within a limited complexity, thus reducing confusions which often arise in evaluating difficulties caused by the latter as intrinsic to problem-solving techniques. Note that although goals and rules of play for this class of puzzles can be presented precisely in a uniform format, solution strategies for different puzzles can be quite diverse and not precisely known in advance (to the system or even to the experimenter for some problems).

OBSERVATION OF GAKU ATTEMPTING A PUZZLE:

THE MECHANISM COORDINATOR AND THE PROBLEM-ORIENTED MECHANISM

We shall now describe the behavior of Gaku in detail as if we were tracing or monitoring its overt and covert actions as it attempts to solve a sequence of puzzles.

Tower of Hanoi Puzzle

One of the puzzles of the class mentioned in the second section is the "Tower of Hanoi."[2] The problem posed in the Tower of Hanoi, illustrated in Figure 2, is to transfer the tower of disks from one peg to either of the two empty pegs in the fewest possible moves, moving one disk at a time and never placing a disk on top of a smaller one. In our board-and-counter interpretation, three pegs are three "stack cells" named C1, C2, C3 with counters N1, N2, . . ., Nn stacked in one of the cells. The following illustration (Figure 3) shows schematically the task for the three-disk case.

Internal representation of the board and counters. In IPL-V representation, the board configuration is input as a list, with its description list, as the following:

```
L1: 9-Ø      9-Ø: Ø
    C1            C1
    C2            9-1        9-1: Ø       9-2: Ø       9-3: Ø
    C3.Ø          C2                C2           C1           C1
                  9-2               C3.Ø         C3.Ø         C2.Ø
                  C3
                  9-3.Ø
```

[2] Gardner, Martin. Mathematical Puzzles and Diversions, New York: Simon & Schuster, 1959, pp. 57-59.

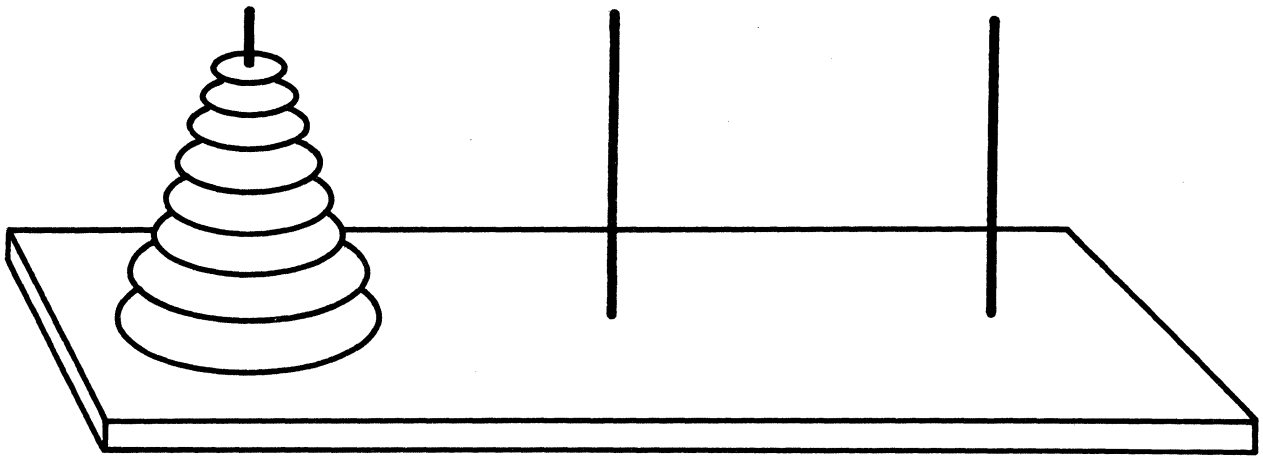
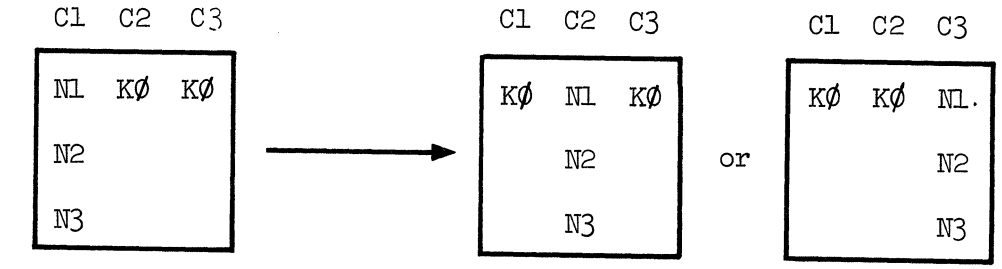


Figure 2.

The Tower of Hanoi Puzzle



The Initial State

The Disjunctive Goal States

Figure 3.

The Three-Disk Case of The Tower of Hanoi

The main list L1 is a describable one containing all the cells on the board, in this case C1, C2, and C3. Its description list, 9-ϕ, contains lists of connected cells for each Ci, e.g., C1 (acting as an attribute) is connected to C2 and C3 under the sublist, 9-1 (acting as values of attribute C1), C2 is connected to C1 and C3 under the sublist 9-2, etc. 9-ϕ would have its description list (in place of ϕ at its head) if C's had an additional attribute. Note that this board representation determines the total set of six possible moves (not necessarily legal moves).

List L2 contains all the counters used for a given task, and if the head is not empty, its description list contains information on an additional attribute(s).

L2: ϕ
N1
N2
N3.ϕ

Task representation. A given task is represented by two or more description lists, L3 for the initial state and L4, L5, . . ., L20 for the goal states if there are more goal states than one. Corresponding to Figure 3:

L3: 9-ϕ 9-ϕ: ϕ
ϕ C1
9-1 9-1: ϕ (Kϕ is a special symbol to
C2 N1 indicate emptiness rather
Kϕ N2 than ϕ.)
C3 N3
Kϕ.ϕ Kϕ.ϕ

L4 will be similar to L3 except for 9-2 under C2 which contains N1, N2, and N3 as in 9-1 for L3 above. Similarly, L5's 9-3 contains N1, N2, and N3.

Conditions for legal moves. Conditions for legal moves are stored in L21 as below. K100 is an indication that the restriction is of the "counter-counter" type, i.e., it indicates what kind of counter can move on top of what other kind of counter. Other types possible are the "counter-cell" (K101), the "cell-counter" (K102), and the "cell-cell" (K103).

```
L21:  9-Ø          9-Ø:  Ø          9-1:  Ø          9-2:  Ø
      K100.Ø             N1               KØ               KØ
                         9-1              N2               N3.Ø
                         N2               N3.Ø
                         9-2
                         N3
                         KØ.Ø
```

The above list indicates exhaustively that counter N1 can move to an empty cell, on top of N2, or on top of N3; N2 can move to an empty cell, or on top of N3; but N3 can move only to an empty cell. This list together with the L1 list of connected cells, will completely describe the legal moves at any intermediate board-and-counter state. The problem-oriented mechanism, with its built-in routine, then can use this information to determine legal moves for a particular current state of the puzzle under consideration.

It may appear to be a complicated method to describe one simple puzzle, but most of these lists can be modified when a new case of the same puzzle is presented. It still does not solve our problems, however; we would want ultimately to indicate conditions for legal moves in terms of relations, e.g., a counter with an attribute value i can move on top of a counter with its value j, as long as $i<j$. This can be done with an added sophistication with the community unit which will program the "legal move generator," a routine which will provide a set of legal moves when it is given a current board-

and-counter configuration. This program will then be embedded in the problem-oriented mechanism which already has the general framework to handle such programs. This type of sophistication, however, has not been incorporated for the current system.

Limits of effort. List L25 contains two names, K104 and K105, whose contents are the maximum number of nodes allowed in the move-tree and the maximum depth (actual number of moves) of the move-tree. When either limit is reached before the task is completed, it will notify the trainer who will then feedback appropriate suggestions (this will be discussed later).

The Mechanism Coordinator

The mechanism coordinator takes care of immediate input-output to and from the environment or the mechanical trainer in addition to its main task of coordinating the functions of the mechanisms. In the following sections we will describe Gaku's action for the three-disk case when it is input as the first task with limits of effort, 50 for K104 and 10 for K105. (We currently restrict the value of K104 to be a multiple of 10 and that of K105 a multiple of 5.)

Abstraction and characterization. An abstraction routine compares two given list structures; in our particular example, it takes the information in L3 (the initial state) and L4 (one of the goal states) and produces three kinds of data abstracts. Abstract #1 simply lists the number of counters in L3 and L4 states; in this example #1 contains 3 and 3. Abstract #2, taking order of occurrence into account, indicates what cells from L3 and L4 have the same contents and what cells do not; in our particular

example, C1's and C2's do not have the same contents, but C3's do. For the cells where mismatch occurred, contents of both are copied into a sublist. Abstract #3, without taking order of occurrence into account, lists cells whose contents match and then lists cells whose contents do not match--in our case, #3 would indicate that C1 and C2 (coming from L3 and L4, respectively) match, having contents N1, N2, and N3; and also C2 and C3 from L3, and C1 and C3 from L4 all match, having KØ as their common content. The description list of the list containing abstracts #1, #2, and #3 contains symbol K106 to indicate that they are abstractions between the initial state L3, and the goal state L4.

A characterization routine then processes these three abstracts to determine characteristics of the given task. First of all, the fact that some cells contain more than one counter indicates they are stack cells, and a symbol to represent the characteristic is stored in a list of characters (i.e, symbols representing characteristics) created by the routine. Next the total number of cells (in this case, 3) and the total number of counters (3) are recorded in the list. Abstract #3 shows that the contents of L3 and L4 are identical except for their arrangement; from this and from the information in #2, the characterization routine can conclude that the task involves moving the contents of C1 to C2. A similar procedure of abstraction and characterization is used for L3 and L5. The conclusion is that C1, containing N1, N2, and N3 at the initial state, must become empty for <u>both</u> goal states. Therefore, an abstracted subgoal that the content of C1 be KØ is stored (in abstracted form, because C2 and C3 contain a symbol K111,

which means the content is unspecified). In addition, moves which take these counters out of C1 are made into a separate list as preferred moves. This list will be consulted before other legal moves are considered at each state. By examining L1 and L21, the characterization routine also records the total number of possible moves (6), the total number of conditions (6), and the total number of attributes for both cells and counters (0).

Initial setup of trees and working cells. The mechanism coordinator also sets up a "past-experience-record tree" which is to contain all the information about solved and unsolved problems the system ever attempted. We have not tried the method of "injecting" the organized body of knowledge, so this tree is currently empty. The task environment memory is also set up in T cells, which makes up the pool of current information relayed back and forth inside the system. In particular, the initial state of the task is copied as the current state in memory, and its address is stored in T50.

Backward reasoning: subgoal generation. The mechanism coordinator also provides subgoals by backtracking from the given goal state(s). The total number of subgoals and the number of levels away from the goal state(s) are determined in relation to K104 and K105, the maximum number of nodes and the maximum depth of the move-tree, respectively. Our current method is simply to let the mechanism coordinator divide the value of K104 by 10 and the value of K105 by 5. For the three-disk case, the corresponding numbers turn out to be 5 and 2. Subgoal generation will be carried out until one of the limits

is reached. For our example, a subgoal generator routine produces from the goal state, $\begin{array}{|c|}\hline \phi\ N1\ \phi \\ N2 \\ N3 \\ \hline \end{array}$, two subgoals, $\begin{array}{|c|}\hline \phi\ N2\ N1 \\ N3 \\ \hline \end{array}$ and $\begin{array}{|c|}\hline N1\ N2\ \phi \\ N3 \\ \hline \end{array}$; and from the goal state, $\begin{array}{|c|}\hline \phi\ \phi\ N1 \\ N2 \\ N3 \\ \hline \end{array}$, subgoals, $\begin{array}{|c|}\hline \phi\ N1\ N2 \\ N3 \\ \hline \end{array}$ and $\begin{array}{|c|}\hline N1\ \phi\ N2 \\ N3 \\ \hline \end{array}$, are produced. One more subgoal at the second level, $\begin{array}{|c|}\hline N2\ N3\ N1 \\ \hline \end{array}$ is produced from the first subgoal shown above and the subgoal generation terminates because five subgoals have been generated. These are stored in a list of subgoals which is consulted whenever a new node is created in the move-tree.

Activation of mechanisms. If Gaku had had previous experience with similar puzzles, their characters (symbols stored as representations of characteristics) in the past-experience-record tree would indicate possible assistance toward solution by the planning mechanism and the induction mechanism. Proper coordination of the functions of the mechanisms then will be necessary and it will be illustrated in the next section. For this first task, however, there is no record of past experience, so the mechanism coordinator activates the problem-oriented mechanism which had replaced the community unit in Figure 1.

The Problem-Oriented Mechanism

Figure 4 depicts the problem-oriented mechanism; it consists of the task analyzer, the move selector which suggests move(s) by using L1 and L21, and the consequence generator which uses L21 to test conditions for the suggested move(s) and to produce consequences of the suggestion to report back to the task analyzer.

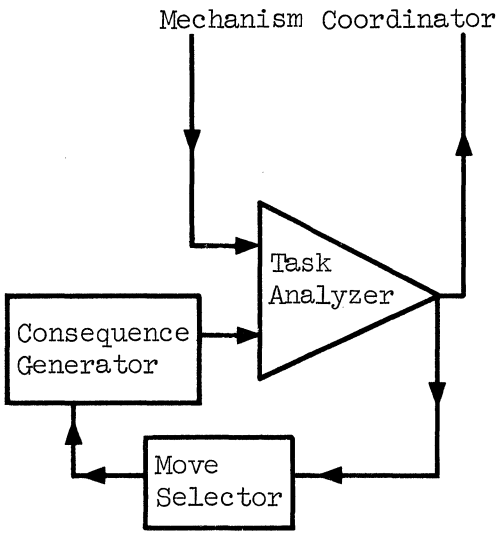


Figure 4. The Problem-Oriented Mechanism

The task analyzer of the mechanism, having received the information from the mechanism coordinator, creates the first node of a move-tree whose address was specified by the mechanism coordinator. The move selector then finds two moves, N1-C2 and N1-C3, meaning move counter N1 to cell C2 and move N1 to C3, respectively. These are the only two moves possible at this state, and consultation with the list of preferred moves indicates that both are in the same list. Since there is no further preference list provided to discriminate between the two, both are given to the consequence generator. The consequence generator then takes the current state, which now contains the initial state of the task, and generates two new states, S_{11} and S_{12}, as the consequences of two suggested moves (see Figure 5). The task analyzer compares the two states but finds no discriminating character which would meet previously specified desirability. It then chooses one of them randomly, say S_{11}, and stores it as the new current state and creates a new node corresponding to S_{11} in the move-tree. Each node contains information about the address of a previous node, address(es) of the next node if generated, a list of legal moves generated, the name of the disk moved, the name of the cell to which the disk has been moved, and a preference criterion, if any, for choosing the move. It does not contain the particular board-and-counter configuration obtained at the node; only one "current state" contains an ever-changing configuration of intermediate state. Therefore, all the states in Figure 5 never exist simultaneously; they are depicted together only for the purpose of describing our experiment.

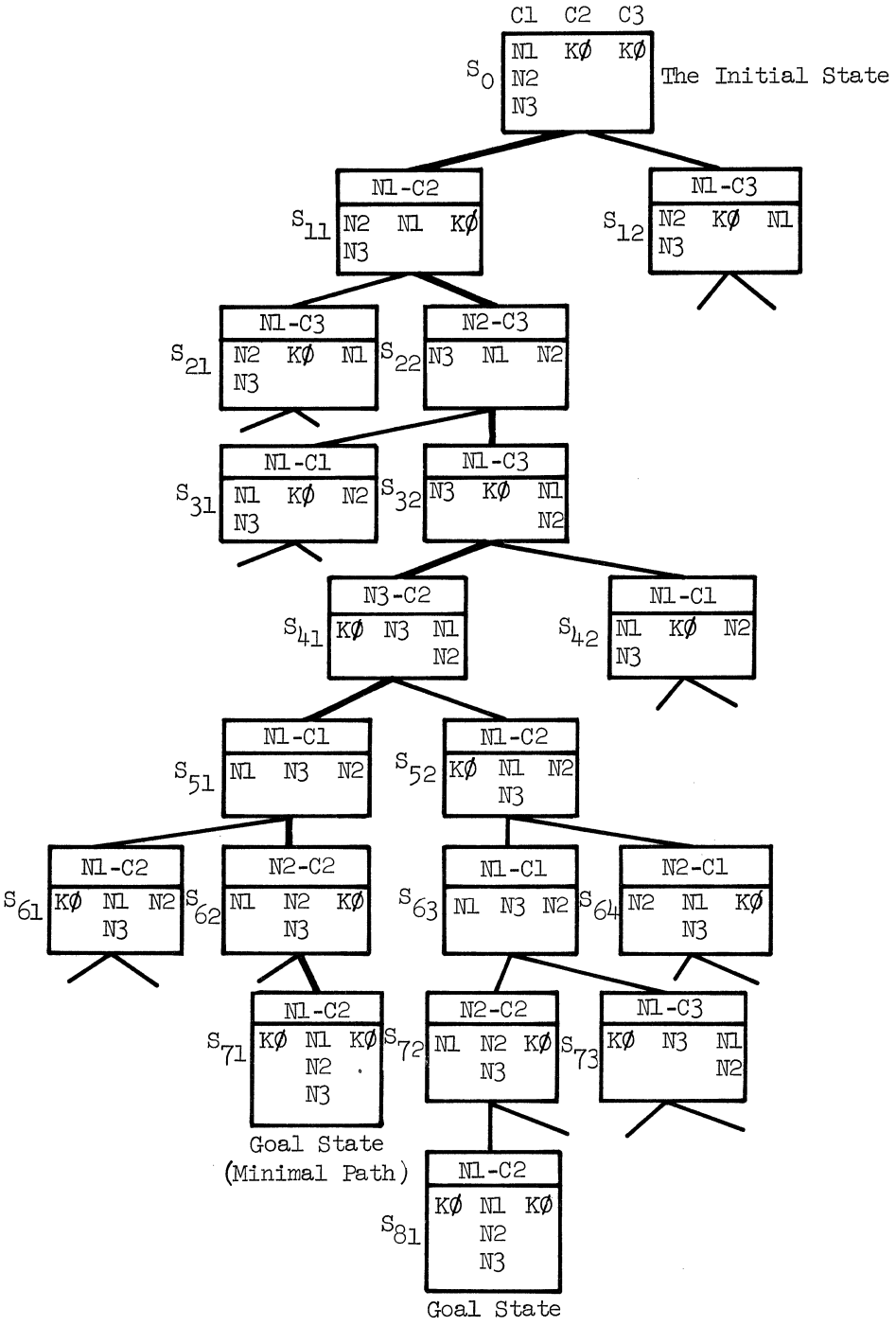


Figure 5
Annotated Move Tree for the 3-Disk Puzzle

The next move chosen is N2-C3 which is in the preferred list, but the other legal move N1-C3 is not. S_{22} is then obtained as the current state. At this state, neither of the two legal moves, N1-C1, N1-C3, are in the preferred list. Instead of choosing one randomly, the move selector attempts a scheme (which will be described next) whose effect is similar to the "look-ahead" feature; the feature provides consequences of future moves as if they were actually made. If the attempt fails to provide any further information, then a random choice will be made.

Recursive use of the problem-oriented mechanism. Since the unused portion of the preferred move list indicates that moving N3 either to C2 or to C3 is desirable, the move selector checks if it can be done. The move selector gives suggested moves, N3-C2 and N3-C3. The consequence generator, in an attempt to make the suggested moves, checks the legality conditions in list L21 and finds that N3 can move only on top of KØ. The current state, S_{22} in Figure 5, clearly shows that neither moves N3-C2 or N3-C3 can be made. Removal of this impediment is now attempted. The consequence generator makes a request of the task analyzer that either C2 or C3 be made empty (KØ) with C1 remaining the same and enters the problem-oriented mechanism (note that the consequence generator is already in the problem-oriented mechanism) so that the entire cycle of the mechanism becomes involved again at one lower level. At the same time the location of a new node for its move-tree is also specified.

The task analyzer automatically takes care of this second-level entrance by a push-down cell (Newell, 1961), T51, so that the second entrance to the

mechanism, before exit is made from the first involvement of the mechanism, does not destroy the information needed for exit from the first one.

Using the same current state information and the request, the task analyzer makes up its own preferred list of moves, C2-C3 and C3-C2. These, in "cell-cell" notation, indicate that moves involving counters at C2, moving to C3, and those at C3, moving to C2, are preferred. The move selector generates N1-C1 and N1-C3 as legal moves from which N1-C3 is selected because its cell-cell representation (C2-C3) is in the preferred list. The consequence generator produces S_{32} as the new current state (Figure 5), and then the task analyzer creates a new node whose name is stored at the specified location. When the requested condition has been satisfied, and the signal S1 (task accomplished) been given, exit from the task analyzer is made by popping up cell T51 after popping up T55, another push-down cell for lists of preferred moves.

When the control is returned to the consequence generator which initiated the request, the current state S_{32} now has C2 empty and the move N3-C2 is made to produce S_{41} as the new state. Note that the node created by the second-level task analyzer has been automatically connected in the proper place of the move-tree.

A wrong move leading to more explorations. At state S_{41} in Figure 5, the move N1-C2 is preferred to N1-C1 because the former keeps C1 empty, and the abstracted subgoal to make C1 empty had not been erased. This leads to S_{52}, and more exploration of the move-tree is necessary until one current state S_{72} is found, to match one of the subgoals generated by the

initial backtracking. At S_{81}, the task analyzer of the problem-oriented mechanism outputs to the mechanism coordinator the signal S1 (task accomplished) and a list of actual moves made to reach the goal state. The mechanism coordinator relays this information to the mechanical trainer which finds the total number of moves (8) to be larger than the known number (7) for a shortest sequence of moves in the three-disk case and feeds back a signal F3 --shorter solution should be found--back up the tree, to the mechanical coordinator.

The mechanism coordinator, after discriminating the signal, changes the value in K105 from 10 to 7 (7 because less than 8 moves must be found) and signals (T10), "backtrack and explore," to the problem-oriented mechanism.

Backtracking by the mechanism involves checking the node to find the address of its parent node and then checking that node for an unexplored path (untried legal moves). After the mechanism backtracks to the node corresponding to S_{41} and explores downward again, S_{71}, the goal state, is finally reached within the prescribed limit. The task analyzer of the mechanism signals (S1), which is reached through the mechanism coordinator, to the trainer who in turn feeds back F1 a signal which indicates "the solution is accepted--conclude this case."

Recording of the Results and Clean Up

The mechanical coordinator, after receiving the signal from the trainer, processes the information collected so far in T's (task environment memory) to decide which parts are to be recorded in the past experience record tree, and which are to be erased. Additional abstraction and characterization of

the record is done so that top levels of the tree (really an inverted tree) will contain more general and abstracted information, and lower levels will contain more detailed and specific information. In case the available space in the core memory becomes scarce, the mechanism coordinator is designed to transfer the lower part of the tree to the auxiliary memory by leaving that information at truncated parts of the tree.

In addition, periodic "contraction" of a body of information is achieved by grouping together formerly separate trees of information and by forming a new node containing an "abstract," to combine the trees. Therefore, the past-experience-record tree will grow, not only downward and sideward, but at the top levels. An example of this might be that after Gaku experiences many cases of the Tower of Hanoi puzzle, trees of case history can be grouped together to form a new node which will contain general characters and a generalized form of solutions; before this is done, each case is stored as an independent entry.

Returning to our present three-disk case, the mechanism coordinator stores the following information as a subtree in the past-experience-record tree: the list of characters of the given task derived by the abstraction and characterization routines (described previously); the sequence of moves accepted as the solution; an abstracted form of the accomplished task as C1-C2 with C1's content indicated by a symbol V1 (grouping parameter) whose attribute has values N1, N2, and N3; the name of the list of preferred moves whose performance attribute, E7, contains 4 (number of moves it influenced with no definite record of failure); the name of the abstracted subgoal which

specifies that C1 be empty and whose performance attribute, E7, has a negative sign to indicate it has a negative influence (this recording is important so that this subgoal will not be generated again); the name of the list of subgoals with its E7 value 1; the name of the move-tree; and copies of the initial task-defining lists (L1, L2, . . ., L25) together with their identifying names so that a new set of task-defining lists can be compared with their counterparts in the old set.

T cells, where most of the intermediate results were handled, are erased.

OBSERVATION OF GAKU ATTEMPTING A PUZZLE: THE MECHANISM COORDINATOR AND ITS INTERACTIONS WITH THREE MECHANISMS

When the mechanical trainer gives Gaku its second task, the four-disk case of the Tower of Hanoi puzzle, the mechanism coordinator tries to utilize its past experience as much as possible. Gaku is so designed that it uses its past experience with a lower "confidence value" at the early stage but with a higher confidence value when a particular device or a method shows some sign of positive influence in a preset number of situations. This confidence value should not be confused with influencing power. When Gaku has very little experience, some factors with low confidence value can strongly influence the system's action because there is not much else to depend on. However, as Gaku gains more experience, its decision-making must depend on more refined discrimination among valued pieces of information, in addition to the ability to separate relevant from irrelevant information. This more advanced stage is where we need more empirical information, and technical and theoretical know-how.

While the previous section described Gaku's action without any past experience, this section describes Gaku's early stage of learning and shows how a little information on past experience can be utilized to best advantage by coordinating the functions of the problem-oriented mechanism, the planning mechanism, and the induction mechanism.

We have previously tried each mechanism independently by both programming and hand simulation, and we have tried the combination of the problem-oriented mechanism and induction mechanism as indicated before (Hormann, 1962, 1963).

Initial Preparation by the Mechanism Coordinator

The new task now given is the four-disk case, shown schematically in Figure 6.

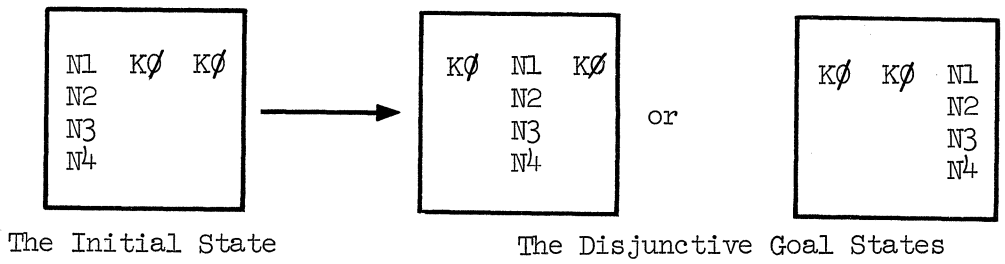


Figure 6. The Four-Disk Case

The mechanism coordinator activates the abstraction routine and the characterization routine as before to produce a new list of characters. This new list is then compared item by item with the old one. The comparison discloses that they are identical except for the numbers of counters and the numbers of

conditions on moves. These numbers are then compared numerically to find that the new task involves one more counter and four more conditions than the previous task. This leads to another use of the abstraction routine, this time on the initial states of the old and new tasks. The result shows that the one counter added to the new task is N4. The mechanism coordinator finds in the past-experience-record tree the values of performance attributes attached to the list of preferred moves, the list of subgoals, and the abstracted subgoal. New lists of the first two kinds are prepared for the new task but not for the last item because of its negative value. The mechanism coordinator stores all the newly generated information in T memory and also brings back the top portion of the subtree for the three-disk case to T memory. The latter consists of the list of characters, the sequence of moves accepted as the solution, and the abstracted form of the accomplished task as C1-C2 with C1's content indicated by a symbol, V1 (grouping parameter), whose attribute has as its values N1, N2, and N3.

With this much preparation, the mechanism coordinator now decides to activate the planning mechanism and the induction mechanism to aid the performance of the problem-oriented mechanism. The two mechanisms can be thought of as working simultaneously (in parallel) and independently of each other, each reporting its suggestions to the mechanism coordinator. Since we have to work with the serial computer, the planning mechanism works first.

The Planning Mechanism

Since general discussion of planning and the planning mechanism (Figure 1) is given previously (Hormann, 1963), this subsection will describe only what is directly relevant to the current task.

With the information in T memory provided by the mechanism coordinator, the task analyzer of the planning mechanism uses the abstracted form of the accomplished task to suggest the context in which the subtask provider is to work. The abstracted form is expressed as a move from C1 to C2 treating three counters, N1, N2, and N3, as a single item V1. Accordingly, the initial state and the goal states of the task are modified. The list of conditions for legal moves is also modified to contain V1-KØ, V1-N4, N3-KØ in our counter-counter notation. The list of preferred moves is modified similarly. The information is then channeled to the subtask provider.

The subtask provider and the consequence generator. The subtask provider finds two legal moves, V1-C2 and V1-C3, from the initial state but chooses V1-C2 because its cell-cell notation, C1-C2, is recorded as already solved. This choice is given to the consequence generator which then provides the new current state S_{11} as shown in Figure 7. Note that three counters are moved together as if they were one. It would be illegal in the problem-oriented mechanism to move more than one counter at a time but this is allowed in the planning stage only because Gaku has the record that moving three counters from one cell to another can ultimately be done in terms of legal moves. The task analyzer, which has previously created a list with the initial state as its first item, now stores S_{11} as the second item.

In the next cycle around the loop of the mechanism, N4-C3 is chosen because it was found in the preferred list, and S_{21} is formed and stored as the third item. The fourth item is the goal state. This list of states will be processed by the mechanism coordinator taking a pair of states at a

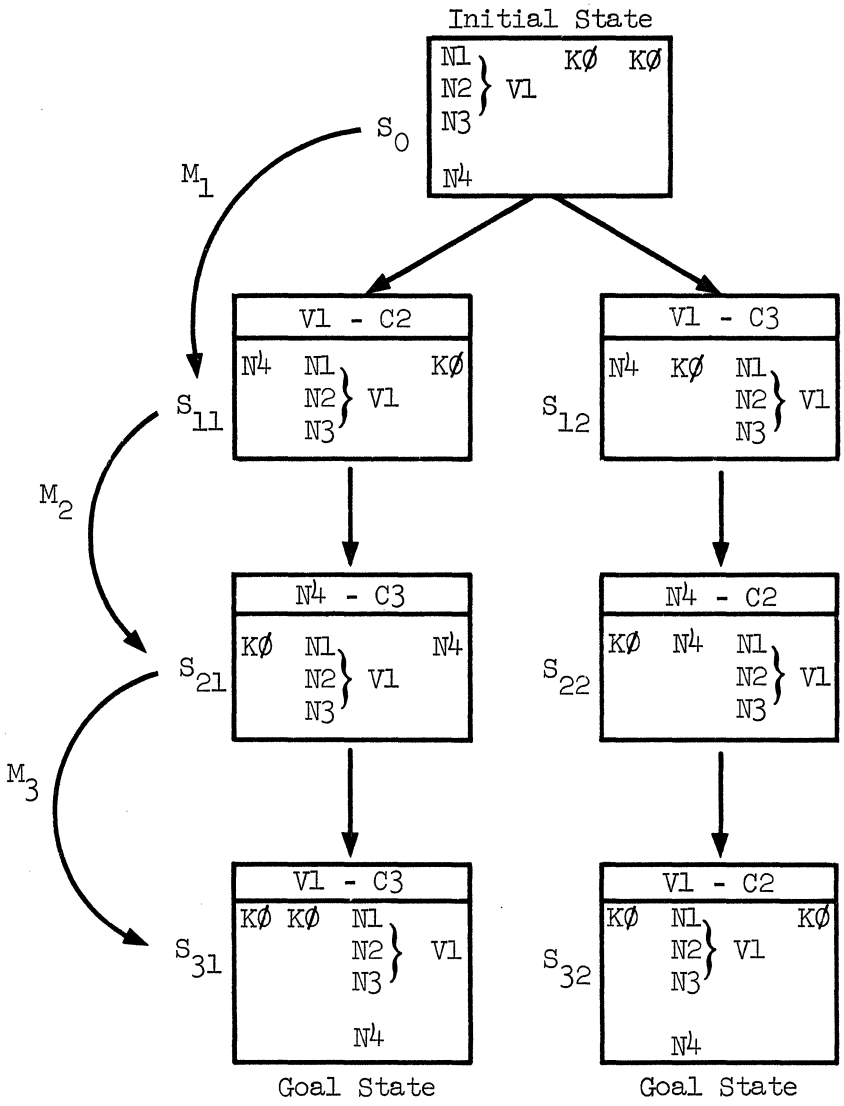


Figure 7. Subtasks of the Four-Disk Case

time to form a subtask (e.g., M_1 with S_0 and S_{11} in Figure 7); the subtask thus created will be given to the problem-oriented mechanism for the actual performance (legal moves) which will be acceptable by the environment.

To illustrate the power of planning, suppose the four-disk case was given with a unique goal state, S_{32} in Figure 7, instead of disjunctive goal states. Then arriving at S_{31} is not a success. The mechanism will have to explore more intermediate states to finally reach S_{32}, possibly ending up with an exhaustive search. However, planning of this kind is relatively cheap--it takes six examinations at this planning state to find the path. But if we were to consider individual moves at the level of detail necessary in the problem-oriented mechanism, an exhaustive search for the four-disk case would take 65,534 examinations of intermediate states.

Peculiarity of a tree formed in planning. As mentioned earlier, the task analyzer of the planning mechanism created the list of states, S_0, S_{11}, S_{21}, S_{31}, which will serve to define subtasks. For our particular example, they will immediately serve as subtasks to direct the behavior of the problem-oriented mechanism. However, for more complex tasks, further subdivision of subtasks may be desirable.

A tree produced in the usual course of planning starts with the top node which consists of the initial state and the goal state. From this node (see Figure 8) there may be several nodes at the second level of subdivision indicating alternative plans; the list of states created in the four-disk case is one of them. If there need be finer subdivision of these subtasks, the third level node will be created. If such a refining process continues,

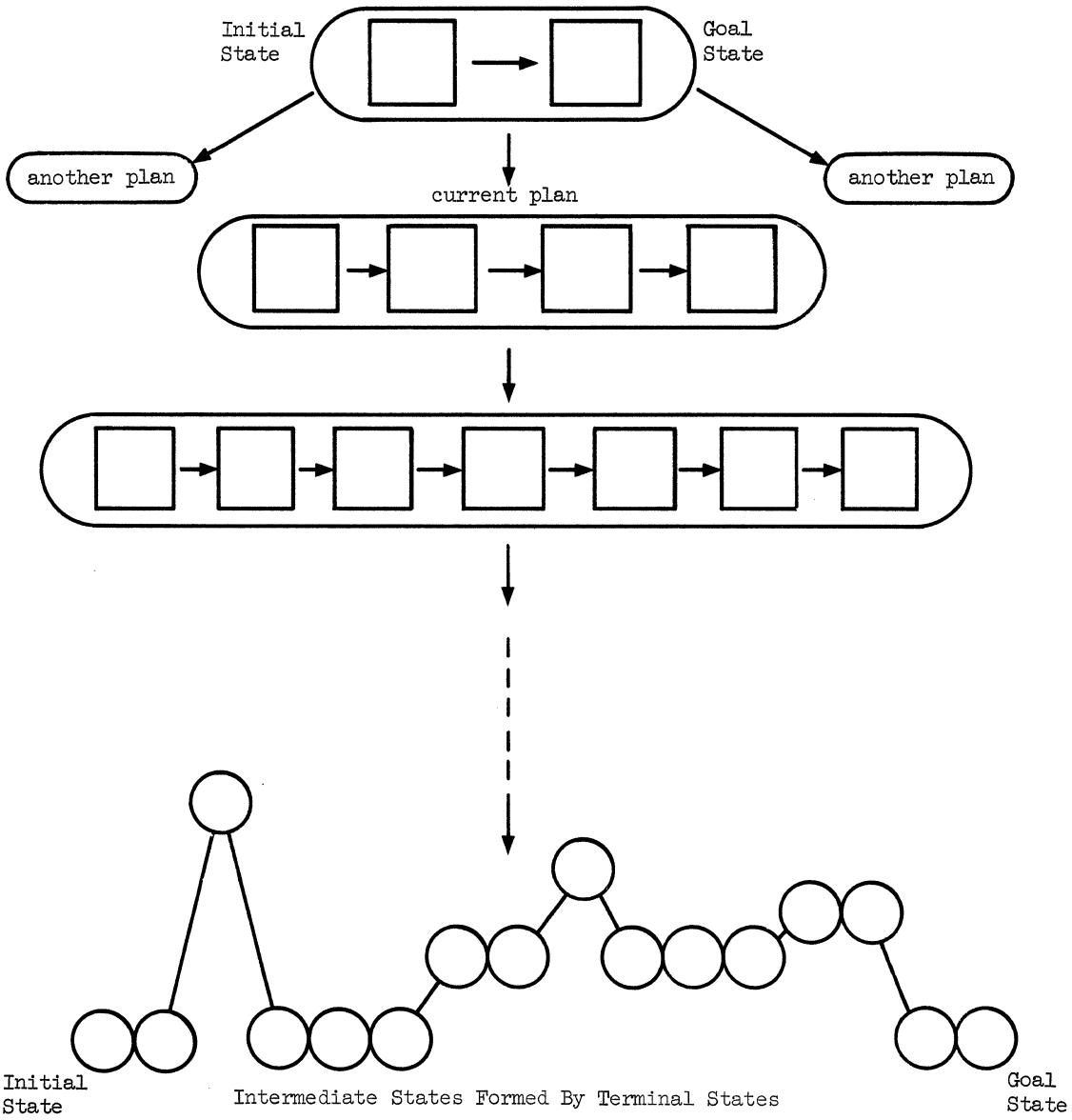


Figure 8 A Substate-Tree Formed in Planning

supplying more details at each lower stage, it will finally reach a set of terminal states where no further breaking down is possible. This final set of terminal states (Figure 8) would then represent a solution. Some states are at different levels as shown in Figure 8 because some smaller subtasks will reach terminal states at earlier stages.

The difference between this type of tree and the kind formed by the problem-oriented mechanism has strategic significance. Here, to find a solution is to find a set of terminal states. In the other kind, as in the move-tree, a solution is represented by a path from the top node (the initial state) of the tree to the goal state at the earliest level node possible (a shortest sequence of moves desired--see Figure 5). Of course, the planning mechanism alone will never attempt to find terminal states. Only when the work of the problem-oriented mechanism is incorporated to fill the gaps will both trees be completed.

Back to the Mechanism Coordinator

Theoretically, by the time the information from the planning mechanism reaches the mechanism coordinator, the induction mechanism would have completed its first round of inductive attempts and suggested some pattern of moves to the mechanism coordinator which would then use items of information to direct the problem-oriented mechanism. Let us use a "flashback" to see what the induction mechanism has been doing.

The Induction Mechanism

The information given to the induction mechanism (Figure 1) consists of the sequence of moves accepted as a solution to the three-disk case: a signal attribute indicating that the new task uses more counters than the previous

one, the corresponding value being N4; and both lists of characters, one from the three-disk case and the other from the current task.

From the sequence of moves, N1-C2, N2-C3, N1-C3, N3-C2, N1-C1, N2-C2, N1-C2, the task analyzer of the mechanism first makes two separate lists, one for counters and one for cells. It then finds a set of distinct elements from each list. The set for the list of counters is found to be N1, N2, and N3, and the one for cells to be C1, C2, and C3. They are then compared with counters and cells in the task description to find the exact match. The conclusion is that all the counters and cells have been involved in the solution (some puzzles leave one or more counters fixed). A conjecture is made that N4 is involved in one of the sequence of moves for the four-disk task. Of course, there is no clue as to where or how the additional counter is used, so it merely stores a signal to indicate that in case of an uncertainty that other criteria do not resolve, a move which involves N4 will be attempted.

Where 7 is the total number of moves in the successful sequence and 3 is both the number of counters and the number of cells, then $\left[\frac{7}{3}\right] = 2$; this bracketed quotient shows that each element in the sequence is used twice on the average. Taking note of this fact, the task analyzer gives a signal, "cyclic,"[3] as a suggested pattern to the conjecture generator together with other generated information.

---

[3]At the outset, we are considering only sequences of repeated patterns; the pattern to be repeated may be of fixed or varied length, and we can handle a pattern-within-a-pattern type to a certain degree of complexity. An example of a repeated pattern of varied length is "2431,246531,2468753l, ..." Sequential pattern detection and generation problems themselves deserve serious research. For the class of puzzles we are considering, repeated patterns are most likely candidates.

Conjecture Generation

Using the information from the task analyzer, the conjecture generator, with the aid of its own subunit (see Figure 1), produces programs which represent conjectures. The subunit is very much like the community unit in structure and function but has very limited domain and range for its input and output. Its input is a sequence of symbols in a list form and a symbol or a set of symbols which characterize the type of patterns to try. The latter information is supplied by the task analyzer of the induction mechanism for trial and error, and many patterns may be tried. The output of the subunit is a program which will regenerate the given sequence under the influence of the characterizing signal(s).

In our example, the conjecture generator receives the signal, "cyclic," together with the list of counters and the list of cells out of the sequence of successful moves. The conjecture generator makes a request of its subunit for each list.

Performance of the subunit. Inasmuch as the reference (Hormann, 1963) describes a step-by-step account of the subunit's performance, this subsection will only summarize it. When the list of counters, N1, N2, N1, N3, N1, N2, N1, is given with the "cyclic" signal, the task analyzer of the subunit looks for the first recurrent position of the first item on the list and finds it to be the third item. It now takes the first two items N1, N2 as defining a cycle phase and asks the program provider to construct a program which will generate N1, N2, N1, N2,.... The constructed program is then executed by the executor-monitor and the results are checked by the task analyzer, one by one, against the given list until a point of mismatch, the fourth item, is detected.

The rest is the usual cycling through of the unit, modifying and retesting, until N1, N2, N1, N3 is generated repeatedly. The task analyzer now outputs the program with a "success" signal to the conjecture generator which activates the subunit. As far as the subunit is concerned, the task is successfully accomplished, although the produced program is proved to be inadequate at a later stage, as we shall see.

The same procedure is used for the list of cells C2, C3, C3, C2, C1, C2, C2, and the repeated pattern finally accepted turns out to be C2, C3, C3, C2, C1, C2. The conjecture generator now combines these two programs so that they will produce together a sequence of "counter-cell" pairs and outputs the resulting program to the consequence generator.

The Consequence Generator and the Task Analyzer

We have changed this part of the mechanism in order to have the consequence generator produce a sequence of proposed moves by executing the generated program, instead of producing one move at a time and waiting each time for the feedback to reactivate it. Since the total number of moves for the four-disk case is currently unknown, the mechanism uses one of the limits of effort K105, given for this case as 20. Therefore, 20 moves are produced by this mechanism, stored as a list whose name is then output to the mechanism coordinator together with a signal, "suggested moves have been produced."

Back to the Mechanism Coordinator

The mechanism coordinator now uses the total information obtained from both mechanisms to guide the problem-oriented mechanism.

Referring back to the subtasks generated by the planning mechanism (see Figure 8), note that the first subtask is solved because it is identical to the three-disk case previously solved. The next one is really a single move because no grouped counters are involved. The third subtask is a variation of the three-disk case which has not been solved yet; this is the area where some possible trial and error can occur.

The mechanism coordinator gives the first subtask to the problem-oriented mechanism and lets it generate the move-tree as before, because there is no performance attribute attached to it yet. Actions of the mechanism, however, are more straightforward because, instead of generating all the legal moves at each state, it uses the suggested move as the first choice trial.

For the third subtask, there is some exploration of the move-tree, but the details will not be reported here since its nature is almost identical to the exploration process in the first task.

After the complete sequence of moves is found and its validity certified by the trainer, the correct moves and suggested moves are compared and unmatched elements noted. A comparison of the suggested and correct move is:

suggested moves:

N1 N2 N1 N3 N1 N2 N1 [N3] N1 N2 N1 N3 N1 N2 N1

C2 C3 C2 C2 C1 C2 C2 C3 C3 [C2] C1 [C2] C2 C3 C3

correct moves:

N1 N2 N1 N3 N1 N2 N1 N4 N1 N2 N1 N3 N1 N2 N1

C2 C3 C2 C2 C1 C2 C2 C3 C3 C1 C1 C3 C2 C3 C3

Squared items indicate where the mismatch occurred.

Analysis of the Results and Modification Attempts

The mechanism coordinator, in an attempt to evaluate the goodness of the suggested moves by the induction mechanism, makes the following test:

Is $\frac{\text{Number of wrong moves}}{\text{Total number of moves}} < K108$?

If the answer is no, then induction mechanism is to find a different pattern generation program. If the answer is yes, modification of the current program by parameterization (which will be explained next) will be suggested. Our current value for K108 is 0.3. The left-hand-side value for the list of counters is $\frac{1}{15}$ and that for the list of cells is $\frac{2}{15}$, so the answers are both yes. A signal for yes case is then given to the induction mechanism to activate its modification action.

Modification of the Conjecture by the Induction Mechanism

The conjecture generator of the induction mechanism, upon receiving the signal through its task analyzer, modifies previously constructed programs by parameterization, i.e., it replaces unmatched symbols with parameters, this time without its subunits assisting it. Special symbols (Vi's) are used to indicate variables, and a sublist containing possible values is created for each Vi. Resulting programs, when executed, would produce a sequence like this:

N1 N2 N1 V1 N1 N2 N1 V1 N1 N2 . . .

C2 C3 C3 V2 C1 V3 C2 C3 C3 V2 . . .

Underlined parts represent cycle phases. V1 finds in its sublist N3 and N4, V2 finds C1 and C2, and V3 finds C2 and C3. The fact that N4 is used for the

four-disk puzzle is consistent with the conjecture made earlier that successful moves for the four-disk case must contain the element N4. The value of the performance attribute is increased so that when the five-disk case is presented, the task analyzer tentatively includes N5 as one of the possible values of V1.

The same procedure of making suggested moves and comparing them against the successful moves is followed for the five-disk case. One mismatch is found and the subsequent modification by parameterization yields the sequence of moves like this:

N1 N2 N1 V1 N1 N2 N1 V1 . . .

C2 V4 C3 V2 C1 V3 C2 V4 C3 . . .

When the new programs are used to suggest moves for the six-disk case, all turn out to be correct. In fact the parameterized program which has now been constructed will solve any n-disk case for three pegs, as long as the initial state has n disks in C1 cell and the disjunctive goal states are given. Of course, Gaku itself will never know the fact unless told by the trainer. However, as Gaku gets more and more experience with the puzzle, and the conjecture program is used successfully, performance value of the conjecture increases so that the mechanism coordinator will tend toward directing a straightforward use of the program.

CONCLUSION

Coordinating the functions of three mechanisms in a learning system we call Gaku is being attempted. To discover capabilities and limitations of such a system, a study is being made to see how it works in a relatively small class of puzzle-solving situations. Sequence-seeking types of activities are being studied since many analogous situations are found in theorem-proving and programming activities. We hope to find some underlying principles and techniques which are generalizable and transferable to a wider variety of problem situations.

Since Gaku has not had much experience, the potentially serious problem of categorization and other related problems have not yet been faced. It may be desirable to "inject" an organized body of knowledge in the same internal representation as if many classes of problems were actually experienced by Gaku, in order to test the feasibility of known or new techniques. Besides, if a powerful learning mechanism becomes available, we shall want to preprogram Gaku to the limit of our capabilities before we let it attempt truly challenging problems.

REFERENCES

Amarel, S., (May 22-24, 1962), On the automatic formation of a computer program which represents a theory. In M. C. Yovits, G. T. Jacobi, G. D. Goldstein, eds., "Self Organizing Systems," pp. 107-175. Spartan, Washington, D. C.

Hormann, A. M., (December 1962), Programs for machine learning, Part I. Information and Control, 5 (4), 347-367.

Hormann, A. M., (December 1963), Programs for machine learning, Part II. Information and Control (in press).

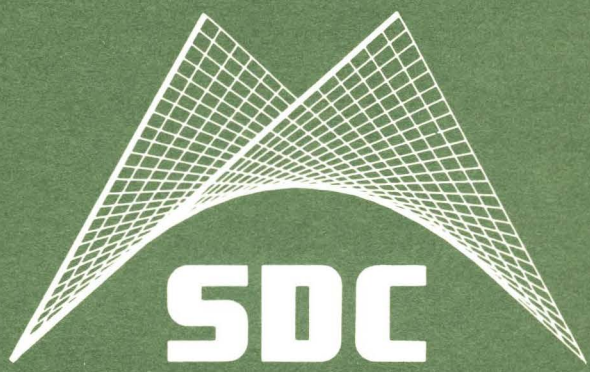
Minsky, M., (1957), Learning systems and artificial intelligence. "Applications of Logic to Advanced Digital Computer Programming," University of Michigan, College of Engineering, Summer Session.

Minsky, M., (1961), Steps toward artificial intelligence. Proc. IRE, 49, 8-30.

Newell, A., ed., (1961), "Information Processing Language-V Manual." Prentice-Hall, Englewood Cliffs, N. J.

Newell, A., Shaw, J. C., and Simon, H. A., (1959), A general problem solving program for a computer. Computers and Automation, 8, 10-17.

Newell, A., Shaw, J. C., and Simon, H. A., (1960), A variety of intelligent learning in a general problem solver. In M. C. Yovits and S. Cameron, eds., "Self Organizing Systems," pp. 153-189. Pergamon, London.



SYSTEM DEVELOPMENT CORPORATION / 2500 COLORADO AVENUE / SANTA MONICA, CALIFORNIA