76-2

Adaptive Human Behavior and Production Systems

by

Arthur Farley

CS-76-2

The ability of humans to adapt their behavior so as to survive and succeed in their environment is a most fundamental and valuable characteristic of the species. The development of computers and the associated concept of programming has led to the natural metaphor of describing adaptive behavior as program modification and development (Lilly, 1969; Keele, 1973). This paper describes a model of the human cognitive system consisting of production systems and a distributed memory. Production systems are the mental programs which act upon the states of available memories. Within this framework, a means for the adaptation of production systems by production systems is defined.

Production Systems

Production systems have been defined and developed as a means of simulating human cognitive information processing (Newell and Simon, 1972; Newell, 1973). A production system is an ordered list of condition-action pairs called rules. A production system is defined relative to a set of available memories which hold the symbolic contents to which rules apply. The condition-part of each rule consists of a finite, ordered list of conditions which are applicable to the contents of the available memories and which are satisfied or not satisfied by those contents. The action-part of a rule is a finite, ordered list of actions which are applicable to the contents of the available memories and which alter those contents. Memories are distinguished by the classes of content each may hold, by their differing content management (remembering and forgetting) strategies, and by the production systems to which each is available.

The execution of a production system is cyclic in nature. To begin each cycle, the condition-parts of the rules are applied to the contents of the available memories in the order specified by the production system list. This process continues until a condition-part is found whose conditions are all satisfied by the current contents of the available memories. When this occurs, the associated rule is said to "fire" and its action-part is then executed. Following this, the cycle begins again. These execution cycles continue until no condition-part can be completely satisfied by the current contents of the available memories or until the production system is explicitly deactivated by the action part of a firing rule. When a rule fires, the contents of the available memories are transformed by the rule's action-part. The output created by production system execution is a trace of the sequence of rule firings and resultant memory contents.

PSMON

PSMON is a production system monitor written in SNOBOL4/SITBOL to be executed on the Digital Equipment Corporation PDP-10 (Farley, 1976). A production system monitor has three functions to fulfill: 1) production system creation, 2) memory initializations, 3) production system execution. Since all production systems to be discussed in this paper have been written for and executed by PSMON, a discussion of its important features is clearly warranted.

PSMON creates production systems by the input processing of production system files. A production system file consists of a set of rule definitions followed by one or more production system specifications. Figure 1 presents an example of a rule definition. Each rule definition consists of several lines. The first line contains the rule name and the rule title, which is a brief statement of rule meaning. The condition-part, followed by the action-part, then follows. Only one condition or action can occur per line. The condition-part is separated from the action-part by the arrow line. Each rule definition is completed by an ENDRULE line. Following the set of rule definitions, production system specifications occur. Each specification serves to organize the indicated rules into a production system rule list. Each production system which is created is given the name indicated in the specification. Figure 2 presents a formal definition of the necessary aspects of production system file syntax in Backus-Nauw Form. Blank lines are ignored, as are comment lines which begin with an asterisk. Condition, action, and arrow lines allow initial tabs or blanks to increase readability. Figure 5 presents an example of a production system file which is discussed later in the report.

Prior to production system execution, the memories to which the production systems are to be applied must be initialized. PSMON distinguishes fi e memories as being basic aspects of the human cognitive processor. These

FIGURE 1

```
R.E1:     GAVE RIGHT ANSWER SO GET NEW STIMULUS
          (C0:  MARKED(HEAR))
          (C1:  MARKED(SAY) .AND. IS(CONTENT(C0)))
                     ==
                                (MARK(C0,USED))
                                (MARK(C1,USED))
                                (CREATENV())
                                (ACTIVE('ANSWER'))
ENDRULE
```

FIGURE 2

```
<ps-file> ::= <rule-defs> <ps-defs>
<rule-defs> ::= <rule> | <rule> <rule-defs>
<rule> ::= <name-line> <cond-lines> <arrow-line>
                       <action-lines> <end-line>
<name-line> ::= <rule-name> : <title>
<rule-name> ::=  R. <alphnums>
<alphnums> ::= <letter>|<digit> |<alphnums> <letter> |
                                 <alphnums> <digit>
<title>  ::=  any string
<cond-lines> ::=  < c-line> | <c-line> <cond-lines>
<c-line>   ::= ( <cond-name> : <specs-list> )
<cond-name> ::=  C <digit> | E <digit> | I <digit>
<specs-list> ::=  any condition function | any condition
                    function  .AND. <specs-list>
<arrow-line> ::=  ==>
<action-lines> ::= <act>| <act> <action-lines>
<act>  ::=  (  any action function  )
<end-line> ::=  ENDRULE
<ps-defs> ::= <ps-line>| <ps-line><ps defs>
<ps-line> ::=  PS(' <ps-name> ',' <rule-list>  ')
<ps-name> ::= <letter> |<ps-name> <letter>|<ps-name><digit>
<rule-list> ::= <rule-name>|<rule-name> , <rule-list>
<letter> ::=  A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y
<digit> ::=   1|2|3|4|5|6|7|8|9|0
```

memories are the ENVIRONMENT, STM (Short Term Memory), ITM (Intermediate Term Memory), LTM (Long Term Memory), and IM (Immediate Memory). Such a "distributed memory" theory of the human cognitive processor has been proposed with supporting argument elsewhere (Hunt, 1973; Farley, 1974).

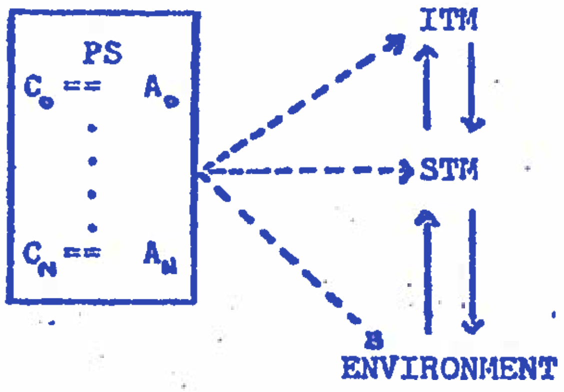
Three of these memories (ENVIRONMENT, STM, and ITM) form the heart of the active processor. Figure 3 presents a graphic representation of the available channels for information transfer between these memories. PSMON provides a straightforward means for specifying to which of these three memories a condition is to apply. The memory to whose contents a condition is to be applied is determined by the initial letter of the condition name (<cond-name> of Figure 2). C indicates that the condition is to be applied to the chunks of STM; I indicates to elements of ITM; E indicates to elements of the ENVIRONMENT.

ENVIRONMENT is an operational representation of current perceptual memory. It consists of an ordered list of elements. Each element is a proposed perceptual representation of some aspect of the current external environment. Once created, it remains constant until explicitly altered. When altered, the old ENVIRONMENT is overwritten.

STM is a simplified, operational representation of the limited active memory of the human cognitive processor. STM is implemented as a short, ordered, list of informational units called chunks. Every chunk has two aspects, a mark and a content. The mark aspect is a symbol used by the production system to facilitate anticipated future access to the chunk's content. The content aspect may be any symbolic structure which is a fundamental unit of representation for a given context. The true unit of representation employed by the human cognitive system can rarely, if ever, be determined by experimentation. The exact number of chunks in STM is accordingly indeterminate, and is a variable to be set during initialization. Once set, it remains constant throughout a processing context.

When a new chunk is created, it is placed at the head of the STM list. When an existing chunk satisfies a condition of the firing rule (is attended), it is moved to the head of the STM list. Whenever a condition is satisfied by an element of ITM or ENVIRONMENT, that information is automatically embodied in a new chunk in STM. Any new information created by rule action is likewise embodied in a new STM chunk. Thus, STM is the "active memory"

FIGURE 3 Production System and Available Memories



Solid arrows indicate normal information flow.

Broken arrows indicate conditions can apply to contents of all three memories.

Figure 4

Condition Functions

IS(STR)	--is satisfied if the chunk has as content the string STR.
MATCH(PATTERN)	--is satisfied if the contents of the chunk is a string which is completely matched by the SNOBOL pattern PATTERN.
MARKED(MARK)	--is satisfied if the mark of the chunk is MARK.
NO('condition function')	--is satisfied if no chunk currently exists which satisfies the condition function in quotes.

ACTION FUNCTIONS

MARK(CHUNK,MRK)	--marks CHUNK with the mark MRK.
NEW(NAME,CONTENT)	--creates a new chunk with the content of CONTENT evaluated, and sets NAME to the CHUNK, as a part of rule, or immediate, memory.
SAVE(CHUNK)	--places CHUNK into ITM, as the last element in the ITM list.
DEACT( )	--deactivates the currently active production system, activating the next element on the production system name stack, if one exists.
ACTIVE(PSNAME)	--activates the named production system by pushing it on the production system name stack.
LISTEN( )	--accepts input from the teletype and evaluates it, creating a new chunk at the head of STM with the resultant value as content and HEAR as mark.
SAY(STRING)	--types the string at the teletype. It also creates a new chunk at the hand of STM with STRING as content and SAY as mark.

of the cognitive processor. Chunks are lost or forgotten by displacement from the end of the short, fixed-length list as new chunks are created and placed in STM.

ITM plays the role of contextual memory within the cognitive processor. A copy of those chunks of STM which are of special interest or significance to a given production system (i.e. a chosen move during problem solving) can be transferred to ITM by explicit rule action. This protects the information of those chunks from being lost when the chunks are displaced from STM by further processing. ITM is thus an ordered list of chunk elements which have been copie from STM. New ITM elements are placed at the tail of the ITM list. When integrated into ITM, the new element may be linked to other chunk elements by appropriate semantic relations. ENVIRONMENT, STM, and ITM form a structure of memories which are meant to reflect a depth of processing (Craik and Lockhart, 1972) philosophy. As information is transferred from the ENVIRONMENT, through STM, to ITM the responsible production systems are also transforming the structure and content of that information.

One memory not shown in Figure 3 is Immediate Memory. IM is not shown there due to its transitory nature. During the application of a rule's condition-part, elements of an available memory may satisfy conditions. As each condition is satisfied by a memory element, its condition name is set to reference that element as a new part of IM. These memory references exist only during application of a rule's condition-part, or if the rule fires, also during the execution of its action-part. IM allows conditions to be specified in terms of those aspects of memory satisfying earlier conditions. It also allows actions to be specified in terms of those aspects of memory content which led to their execution. Any new chunks created by rule actions also become part of IM for the remainder of the rule firing. Each rule has its own associated, potential IM.

LTM is also not shown in Figure 2 because it cannot be readily separated from the other memories. LTM is the underlying basis of all the production systems and other memories. LTM contains the universe of symbols, symbol classes, interrelationships between symbols, and basic cognitive processes. The current contents of another memory are instances of LTM contents (implemented as pointers to those contents). The rules of any production system are written in terms of LTM contents. Except for a small set of symbols and condition and action functions, a user must define LTM anew for each production system application. Figure 4 presents a description of the basic condition and action functions which are provided.

The condition-part-satisfying process and the transfer of control between production systems are two characteristics of production system execution by PSMON which bear discussion. During the condition-part-satisfying process, an element of an available memory can satisfy at most one condition of the rule. Besides being a natural convention, this allows the condition part of a rule to ask such questions as whether there are two identical elements in an available memory.

Another aspect of the condition-part-satisfying process is that PSMON will not back up and attempt to resatisfy a condition with a different memory element when a later condition fails to be satisfied. This has an interesting consequence when the failing condition is dependent upon the content of a memory element currently satisfying a prior condition. This is best explained by illustration. Consider rule R.El, shown in Figure 1, as it is applied to the following STM of three elements:

(HEAR::CON)   (SAY::CON)   (HEAR::PAT)

Here and in all presentation of STM to follow, the mark aspect of a chunk is shown before the two colons; the content aspect follows. Applying rule R.El, the first chunk of STM satisfies condition CØ. Condition Cl is then satisfied by the second chunk and rule R.El fires. This rule illustrates the use of an Immediate Memory element (CØ) in a condition. Consider now rule R.El as it is applied to the following STM:

(HEAR::PAT)   (HEAR::CON)   (SAY:CON)

Again, the first chunk satisfies condition CØ. However, condition Cl cannot be satisfied now and rule R.El fails to fire. PSMON will not attempt to satisfy condition CØ with the second chunk, which would result in R.El firing.

This condition-part-satisfying convention both facilitates and simplifies the rule firing process. An operational interpretation is that if there are two or more elements of an available memory which could satisfy a given condition, the first one encountered effectively masks the others from consideration. In the case of STM, this element is the most recently created or attended chunk. In the case of ITM, it is the first (not most recent) chunk saved during a given processing context. Later examples will show that the convention can lead to extra production system rules in some contexts, however.

The ACTIVE and DEACT functions described in Figure 3 allow the action part of a rule to effect a transfer of control between production systems. PSMON maintains a production system name stack. The production system which is named by the top element of the stack is the currently active production system. The ACTIVE function pushes its name argument onto the top of the stack. The DEACT function pops the production system name stack. The stack is also popped when no condition-part of the currently active production system can be satisfied. Transfer of control becomes effective only at the beginning of a system cycle. At that time, condition-parts of the rules of the production system named by the top of the stack are those applied to the memory contents for satisfaction. If the production system name stack is empty at the beginning of any cycle, production system execution ends.

Production System Adaptation

Four action classes present themselves as the natural means for production system adaptation. These are: (1) the deletion, or removal, of a rule; (2) the insertion of a rule; (3) the reordering of existing rules; (4) the alteration of the condition and/or action part of an existing rule. This section describes a production system able to delete and insert rules [action classes (1) and (2)]. Action classes (3) and (4) can be realized by appropriate combinations (compositions) of action classes (1) and (2).

Figure 5 presents the production system ADAPT, together with a description of its associated LTM contents. ADAPT is a general vehicle for production system adaptation. Depending upon STM contents, ADAPT may create and insert a new rule or delete an existing rule from a production system. The production system which undergoes the specified adaptation is the one named by the element below it on the production system name stack.

ADAPT can only delete the most recently fired rule of a given production system. With each production system, PSMON associates a rule pointer, which is set to reference a rule as it fires. These pointers are available to the

Figure 5

```
*         PRODUCTION SYSTEM WHICH CREATES NEW RULES FROM CURRENT
*         STM CONTENTS MARKED COND AND ACT AND INSERTS THE RULE IN
*         THE PRODUCTION SYSTEM NEXT ON THE PS NAME STACK.
*
R.0:      REMOVE RULE WHICH FIRED LAST FROM ADAPTING PS
          (C0: MARKED(LOC) .AND. IS(REMOVE))
          (C1: NO('MARKED(LOC)'))
                            =>
                                       (MARK(C0,USED))
                                       (REMOVER())
                                       (DEACT())
                                       (DEACT())
ENDRULE
*
R.1:      ADD CONDITION TO THE CONDITION LIST
          (C0: MARKED(CNDL))
          (C1: MARKED(CND))
                            =>
                                       (ADDCNDL(CONTENT(C0),CONTENT(C1)))
                                       (MARK(C1,USED))
ENDRULE
*
R.2:      CREATE CONDITION LIST
          (C0: MARKED(CND))
                            =>
                                       (NEW('C1',CCONDL(CONTENT(C0))))
                                       (MARK(C0,USED))
                                       (MARK(C1,CNDL))
ENDRULE
*
R.3:      ADD ACTION TO THE ACTION LIST
          (C0: MARKED(ACTL))
          (C1: MARKED(ACT))
                            =>
                                       (ADDACTL(CONTENT(C0),CONTENT(C1)))
                                       (MARK(C1,USED))
ENDRULE
*
R.4:      CREATE ACTION LIST
          (C0: MARKED(ACT))
                            =>
                                       (NEW('C1',CACTL(CONTENT(C0))))
                                       (MARK(C0,USED))
                                       (MARK(C1,ACTL))
ENDRULE
*
R.5:      CREATE NEW RULE, INSERT IT, AND DEACTIVATE
          (C0: MARKED(CNDL))
          (C1: MARKED(ACTL))
          (C2: MARKED(LOC))
                            =>
                                       (INSERTR(CRULE(CONTENT(C0),CONTENT(C1)),CC
                                       (MARK('C0,C1,C2',USED))
                                       (DEACT())
                                       (DEACT())
ENDRULE
*
*         CREATE PRODUCTION SYSTEM
*
PS('ADAPT','R.0,R.1,R.2,R.3,R.4,R.5')
```

INSERTR and REMOVER action functions of ADAPT. To delete the rule from the production system named CONCEPT, for example, a rule must fire containing the sequence of actions: NEWLOC (REMOVE); ACTIVE ('CONCEPT'); ACTIVE ('ADAPT'). ADAPT will be the newly activated production system at the start of the next execution cycle. CONCEPT will be below it on the production system name stack, and thus becomes the adapting production system. Rule R.Ø of ADAPT will fire, removing the rule from CONCEPT. Note that the action part of rule R.Ø contains two DEACT action functions. This removes both ADAPT and CONCEPT from the production system name stack.

To create and insert a new rule into an adapting production system, a production system must first create the new rule's conditions and actions, placing them in STM. Rules R.1, R.2, R.3, and R.4 of ADAPT collect these conditions and actions into lists which are suitable for integration into a rule structure. There is an inherent flexibility in how ADAPT may be employed to realize this goal. ADAPT may be activated as each condition or action is created; it may be activated only when all have been created; or it may be used with any strategy between these extremes, some of which may include the use of ITM. Several options will be illustrated in the application examples to follow.

Once the condition and action lists have been completed and a location for rule insertion is determined, rule R.5 creates a rule and inserts it where specified. Four insertion locations are available for ADAPT: TOP or BOTTOM of the adapting production system list; and BEFORE or AFTER the last rule to have fired of the adapting production system. Note that rule R.5 again contains two DEACT actions, removing ADAPT and the name of the adapting production system from the production system name stack after rule insertion.

Several recent reports discuss adaptive production systems, presenting example applications. Waterman (1970) discusses an adaptive production system which improves its performance at draw poker by acquiring bidding heuristics with play experience. Waterman (1974) also reports several adaptive production systems capable of emulating EPAM, acquiring elementary arithmetic capability, and solving a restricted class of sequence extrapolation problems. The adaptive system being discussed here differs primarily in the attempt to modularize the adaptive facility in one production system (ADAPT), to specify some limitations and capabilities, and to describe and illustrate an associated theory of memory use during adaptation. Waterman also employs a production system monitor, PAS-II, with several basic properties that differ from PSMON (Waterman, 1973).

Hedrick (1976) discusses a system for learning production systems from examples. This system compares an existent production system's output with a specification of correct output. If differences exist, the structure of rules which fire is inspected and modified to create the appropriate output. Interesting points are mady by Hedrick regarding rule generalizations and strategies for modification selection. Two applications he discusses are sequence extrapolation and the development of an elementary language processing production system. Hedrick's work constrasts with that discussed here in several ways. That adaptation system is not itself a production system. It does not operate in a memory environment which analogous to that of the human cognitive processor. It has the ability to generate alternative rule structures and to inspect, compare, and evaluate these directly. The adaptation system described here can only understand a production system rule indirectly, through an analysis of the STM contents which result from its firing.

Verbal Learning

EPAM (Feigenbaum, 1963; Feigenbaum and Simon, 1964) and SAL (Hintzman, 1968) are similar computer programs which simulate verbal learning behavior, more specifically, the acquisition of lists of associative pairs of three-letter, nonsense syllables. SAL develops discrimination trees which consist of test nodes and answer nodes. A test node checks for a letter in a specified position of the stimulus; an answer node holds a response. Given a stimulus, the discrimination tree is traversed according to the results of encountered test nodes, until an answer node is reached, the content of which is then reported as the remembered associate. By introducing two parameters to control discrimination tree development, Hintzman reports that many aspects of human verbal learning behavior can be reproduced by the simulation model.

Consider that the stimulus is available to the system through the ENVIRONMENT and that the stimulus is represented as three elements of the form: POSITION * LETTER, where POSITION can be 1, 2, or 3 and LETTER can be any letter of the alphabet. It is assumed that letter positions are noticed in the fixed order 1, 3, 2. Figure 6 presents a list of four paired associates and a discrimination tree which is sufficient to provide consistently correct responses to the stimuli.

The decision structure embodied by any given discrimination tree can be represented by production system in a straightforward manner. Figure 7 presents a production system which is equivalent to the discrimination tree of Figure 6 (rule names and titles have been omitted). The test associated with every positive branch on the path through the discrimination tree from the top test node (root node) to a given answer node corresponds to the conditions tested in the production system rule yielding the same response.

Figure 8 presents three production systems which, in conjunction with ADAPT, are capable of developing a production system representation of a discrimination tree when presented with a list of paired associates. The adapting production system is ANSWER, which initially will respond '?' to any stimulus (rule R.A100). EPAM interacts with the experimenter, asking for new stimuli and correct responses in turn. After ANSWER responds to a stimulus, EPAM asks for the correct response (rule R.E3). If a correct response has been given by ANSWER, no modification is made, a request is made for a new stimulus and ANSWER is reactivated (rule R.E1). Otherwise, EPAM activates LEARN (rule R.E2). LEARN first develops a set of condition chunks based on the current ENVIRONMENT (rules R.L1, R.L1A, R.L2). ADAPT is activated to integrate these conditions into a condition list suitable for rule creation (rule R.L4). When all of the conditions have been integrated into a condition list, no rule of ADAPT can fire. ADAPT deactivates, returning control to LEARN.

When control returns to LEARN, rule R.L3 creates the action chunks which will allow ANSWER to give the correct response, which is the content of the chunk marked HEAR, and to deactivate. The new rule will be placed before the rule of ANSWER which gave the wrong answer leading to modification. Note the transfer of control actions in rule R.L3. LEARN is first popped from the production system name stack, EPAM replacing it as the top element. Then ANSWER and finally ADAPT are pushed onto the stack. Thus ADAPT will be the next active production system and ANSWER will be the production system to be modified. When ADAPT completes its activity with rule R.5 firing, control will return to EPAM as both ANSWER and ADAPT are popped from the production system name stack.

Figure 6

S	R
PAJ	CON
BIF	DEZ
PUJ	LEQ
BOT	FIP

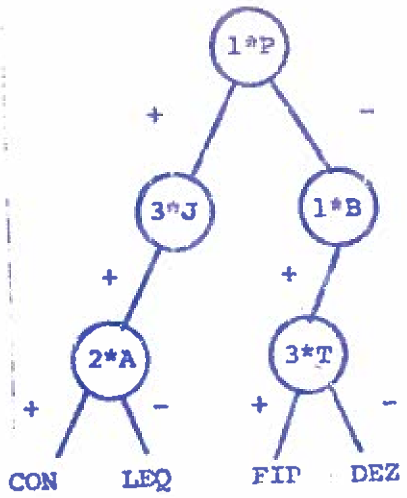


Figure 7

```
(EØ:  is('1*P'))
(El:  is('3*J'))
(E2:  is('2*A'))

                 ==>

                      (SAY('CON'))

ENDRULE

(EØ:  is('1*P'))
(El:  is('3*J'))

                 ==>

                      (SAY('LEQ'))

ENDRULE

(EØ:  is('1*B'))
(El:  is('3*T'))

                 ==>

                      (SAY('FIP'))

ENDRULE

(EØ:  is(1'*B'))

                 ==>

                      (SAY('DEZ'))

ENDRULE
```

Figure 8

```
*
*
*
*       THREE PRODUCTION SYSTEMS, WHICH IN CONJUNCTION WITH ADAPT
*       SIMULATE A VERSION OF EPAM.
*
*
*
R,L1:   GET NEW FEATURE OF CURRENT STIMULUS
        (C0; MARKED(NEW))
        (C1; MARKED(NEW))
        (E0; IS(CONTENT(C0)))
        (E1; IS(CONTENT(C1)))
        (E2; MATCH(ASPECT))
        (C2; MARKED(HEAR))
                         ==>
                                (MARK('E0,E1,E2';TRUE))
                                (MARK('C0,C1',USED))
ENDRULE
*
*
R,L1A:  GET NEW FEATURE OF CURRENT STIMULUS
        (C0; MARKED(NEW))
        (E0; IS(CONTENT(C0)))
        (E1; MATCH(ASPECT))
        (C1; MARKED(HEAR))
                         ==>
                                (MARK('E0,E1';TRUE))
                                (MARK(C0,USED))
ENDRULE
*
*
R,L2:   CREATE CONDITION CHUNK
        (C0; MARKED(TRUE))
        (C1; MARKED(HEAR))
                         ==>
                                (NEWCOND('E',("IS('" CONTENT(C0) "')"))
                                (MARK(C0,USED))
ENDRULE
*
*
R,L3:   CREATE ACTION AND LOC CHUNKS AND ACTIVATE ADAPT
        (C0; MARKED(HEAR))
        (C1; MARKED(CNDL))
                         ==>
                                (MARK(C2,USED))
                                (NEWACT('DEACT()'))
                                (NEWACT(("SAY('" CONTENT(C0) "')")))
                                (NEWLOC(BEFORE))
                                (DEACT())
                                (ACTIVE('ANSWER'))
                                (ACTIVE('ADAPT'))
ENDRULE
*
*
R,L4:   ACTIVATE ADAPT TO CREATE CONDITION LIST
        (C0; MARKED(HEAR))
                         ==>
EVAL =                          (ACTIVE('ADAPT'))
```

```
*
*       EPAM RULES TO INTERACT WITH ENVIRONMENT
*
*
R,E1:   GAVE RIGHT ANSWER SO GET NEW STIMULUS
        (C0; MARKED(HEAR))
        (C1; MARKED(SAY) .AND. IS(CONTENT(C2)))
                         ==>
                                (MARK('C0,C1',USED))
                                (CREATENV())
                                (ACTIVE('ANSWER'))
ENDRULE
*
*
R,E2:   GAVE WRONG ANSWER SO LEARN NEW RULE FOR ANSWER PS
        (C0; MARKED(HEAR))
        (C1; MARKED(SAY))
                         ==>
                                (MARK(C1,USED))
                                (ACTIVE('LEARN'))
ENDRULE
*
*
R,E3:   GET CORRECT ANSWER
        (C0; MARKED(SAY))
                         ==>
                                (LISTEN())
ENDRULE
*
*
R,E4:   GET NEW ENVIRONMENT
        (C0; SUCCEED)
                         ==>
                                (CREATENV())
                                (ACTIVE('ANSWER'))
ENDRULE
*
*       INITIAL ANSWER STRUCTURE, AN EMPTY DISCRIMINATION TREE
*
R,A100: DON'T KNOW ANSWER, SO SAY ?
        (E0; MATCH(ASPECT))
                         ==>
                                (SAY('?'))
                                (MARK(E0,TRUE))
                                (DEACT())
ENDRULE
*
*
*       DEFINE PRODUCTION SYSTEMS
*
PS('ANSWER','R,A100')
PS('EPAM','R,E1,R,E2,R,E3,R,E4')
PS('LEARN','R,L1,R,L1A,R,L2,R,L3,R,L4')
```

Appendix I presents an edited version of the trace produced by the production systems when learning the list of paired associates given in Figure 6. The initial part of the appendix shows all of the rule firing and STM states leading to insertion of the first new rule. This is followed by a sequence of cycles which present the new stimulus, as presented in ENVIRONMENT, the response provided by ANSWER, and the resultant, adapted ANSWER. One can see several instances of stimulus generalization (marked by asterisks). The production system shown in Figure 7 is realized after the fifth stimulus (a repeat of the first) is presented. Note rule R.A1 is a redundant rule with respect to the given list (it will never fire), but remains as a remnant of the early stages of learning.

Concept Formation

The ability to develop decision structures which correctly classify objects as instances or not instances of a concept is a related form of human adaptive behavior. The methods by which humans develop these conceptual decision structures have received considerable experimental and theoretical attention (Bruner, Goodnow, & Austin, 1956; Hunt, Marin & Stone, 1966; Bourne, Eckstrand & Dominowski, 1971; Levine, 1975). Figure 9 presents the set of conceptual rules which have been most studied. The name of the conceptual rule, an instance of the concept, and the condition-part(s) of the one or two production system rules necessary to embody the concepts decision structure are given.

Appendix II presents the production system CL which, in conjunction with ADAPT, is capable of developing production system representations of these conceptual decision structures. The adapting production system is named CONCEPT, which initially answers NO to all objects (rule R.CØ). Production system rules are created, modified, or removed based upon the objects presented to the system in ENVIRONMENT and upon their associated feedbacks.

Several aspects of the developmental strategy employed by CL reflect or predict important experimental results concerning human concept formation. The rules in CONCEPT at any time can be considered to be the current working hypothesis. When CONCEPT correctly classifies the current stimulus, no

Figure 9

Level I

```
Affirmation
RED
(EØ : is('RED'))
                  = =>
```

```
Negation
not RED
(EØ : NO(" is('RED')"))
                        = =>
```

Level II

```
Conjunction
RED and SQUARE
(EØ : is ('RED'))
(E1 : is ('SQUARE'))
                  = =>
```

```
Alternative Denial
either not RED or not SQUARE
(EØ : NO("is ('RED')"))
                        = =>
(EØ : NO("is(' SQUARE ')"))
                        = =>
```

```
Inclusive Disjunction
RED or SQUARE or both
(EØ : is('RED'))
                  = =>
(EØ : is('SQUARE'))
                  = =>
```

```
Joint Denial
neither RED nor SQUARE
(FØ : NO("is('RED')"))
(E1 : NO("is('SQUARE')"))
                        = =>
```

```
Implication
if RED then SQUARE
(EØ : NO("is('RED')"))
                  = =>
```

```
Exclusion
RED and not SQUARE
(EØ : is ('RED'))
(E1 : NO("is('SQUARE')"))
                        = =>
```

Level III

```
Bicondition
RED  if and only if SQUARE
(EØ : is('RED'))
(E1 : is('SQUARE'))
                  = =>
(EØ : NO('RED')"))
(E1 : NO("is('SQUARE')"))
                  = =>
```

```
Exclusive Disjunction
RED or SQUARE but not both
(EØ : is ('RED'))
(E1 : NO("is('SQUARE')"))
                        = =>
(EØ : is('SQUARE'))
(E1 : NO("is('RED')"))
                        = =>
```

adaptation is made (rules R.L1, R.L2A, R.L2B, R.L2C). CL uses ITM to remember those object features which have previously been chosen as bases for the condition of a conceptual rule. Only an object feature not currently in ITM may be chosen as basis for the condition of a rule which has one condition. CL thus does not repeat prior hypotheses (Levine, 1975). Rules with two conditions (conjunctive rules) are formed only when all features of a current object have been previously used in rules with one condition. Since only one rule is formed at a time, those decision structures based upon two rules (disjunctive concepts) require, in general, more development activity than a simple affirmative concept. These last two aspects of the strategy predict a ise in concept attainment difficulty with increasing concept level. This is consistent with experimental results (Bourne et al, 1971).

CL develops CONCEPT by placing rules which say YES (indicating an object is a concept instance) ahead of the default rule R.CØ, which provides all of the NO responses. When CONCEPT says NO, but the ensuing feedback indicates that the object is concept instance, a rule is added at the top of CONCEPT. This rule has as its sole condition a feature of the current object not yet in ITM, if such a feature exists (rules R.L3A, R.L3B, R.L3C). These rules illustrate the need for multiple rules which is caused by the condition satisfying convention. Otherwise, a conjunctive rule is formed, with two conditions, each based upon a feature of the current object (rule R.L3D).

If, on the other hand, CONCEPT classifies the object as a concept instance but the feedback indicates otherwise, the rule producing the wrong response may be altered or removed. Rule alteration is affected by ADAPT through insertion of a new rule after the offending rule and then removal of the offending rule. Rule alteration is realized by rule replacement. Rules R.L4B and R.L4C prepareto replace the offending rule with one having a sole negative condition based upon a feature of the current object. This feature must not yet be in ITM. When a negative condition is satisfied by an available memory, no element of the memory becomes part of IM. Thus, a rule based upon a sole negative condition must supply a record of what missing feature led to its firing through explicit rule action to facilitate future rule alteration.

If all features of the current object are already in ITM, the offending rule is replaced by a conjunctive rule of its positive condition and a new negative condition based upon a feature of the current object (R.L4D) or of two negative conditions (R.L4E). If the offending rule is a conjunctive rule, it is removed from CONCEPT, without replacement (R.L4A).

The system's performance is greatly affected by the sequence of objects presented and the chosen noticing order of an object's features, as listed in ENVIRONMENT. Though the model is capable of representing and developing all of the conceptual rule structures of Figure 9, it is surely incomplete in its representation of human behavior on the task. The system requires a means to eliminate rules based upon CONCEPT complexity. One likely heuristic method would be to allow only several rules to ever exist in CONCEPT, since all conceptual structures require at most two production system rules. Another missing aspect is how to allow the system to decide when it has become lost and to appropriately regroup its efforts. Behavior protocols are needed to allow further specification of CL which could account for aspects of human performance beyond the general characteristics it presently predicts. Yet, the current model illustrates the principles of production system adaptation for this processing context.

Problem Solving and Einstelling

Newell and Simon (1972) discuss the means-ends analysis approach to problem solving. Their work and other recent studies (Greeno, 1973; Simon and Reed, 1976; Atwood and Polson, 1976) confirm the role of this strategy in human problem solving behavior. Simply stated, the strategy selects the move which eliminates as much as possible of an existing difference between a current problem state and the desired goal state. The strategy will always select a move which leads directly to a goal state. If no such move exists, a subset of possible moves which are relevant to reducing the difference are evaluated and compared. The move which results in the least remaining difference is the move chosen. The state resulting from this move becomes the new current state and the move selection process begins again. This cyclic process continues until either the goal state is reached and success is signalled or no applicable move is relevant, in which case the strategy reconsiders unchosen moves from prior states.

Appendix II presents two production systems, SOLVE1 and SOLVE2, which together can solve a class of simple water jug problems by the means-ends analysis approach. These water jug problems consist of four jugs: A, B, C, and GOAL. The capacities of jugs A, B, and C differ with each problem, as specified in a problem statement. The capacity of the GOAL jug is constant, say at 100. Each problem statement specifies an amount of water to be placed in the GOAL jug, which starts empty. Two classes of move operators are available to realize this goal state: (1) pour into GOAL a full A, B, or C jug; (2) bail out of GOAL a full A,B, or C jug. A solution to a given problem consists of a sequence of move operators which produces the goal state in the GOAL jug. One further restriction is that no jug may be used more than three times in any solution. For example, consider the following problem:

A	B	C
31	5	27

GOAL
48

This problem has the solution: pour in A, pour in C, bail out B, bail out B.

To solve these problems, SOLVE1 is first activated. SOLVE1 initially perceives the goal state and creates a new internal state, which is minus the desired goal state (rule R.W9). SOLVE1 and SOLVE2 maintain a current state, in the chunk marked STATE OR STATE, which actually is the difference between the current amount of water in GOAL and the desired goal state. The system then begins to consider moves to solve the problem. If jug A, B or C euqals STATE the in or out move is generated to complete the solution (rules R.W1A, R.W1B). ANSWER is activated to report the solution sequence.

If no move is available which can complete the solution, the three relevant (in or out) moves are generated for consideration (rules R.W2A, R.W2B). If any move leads to a state which is equal to the capacity of jug A, B, or C, that move is selected (rule R.W3). Otherwise, moves to prior states and moves which lead away from the goal state are rejected by the "acceptability heuristic" (rules (R.W4A, R.W4B, R.W4C, R.W5). A move which leads away from the goal state is one which produces a state with a greater difference value than the current state. If, after move elimination, any moves remain under consideration, the one leading to the state with the least remaining difference is selected (rules R.W6A, R.W6B, R.W7). When a move is selected, it is placed in ITM, and a new current state is established in STM. The move selection process then begins.

If, after elimination of moves to prior and lesser states, no moves remian under consideration, then SOLVE1 returns to the initial state to begin the search with a different move. Such behavior has been shown to occur frequently in human problem solving protocols (Atwood & Polson, 1976). If all moves have been tried from the initial state, then SOLVE2 is activated (rule R.W8). SOLVE2 relaxes some of the move selection criteria employed by SOLVE1. It will choose any move leading to a better state, if that state is a new state, one not yet in ITM (rule R.W12). If no such move exists, it will select a move that leads away from the goal state, if that move leads to a new state as yet not in ITM (rules R.W22 and R.W32). Thus, SOLVE2 eliminates the acceptability heuristic, accepting moves which lead to lesser states. If all moves from a current state lead to states already in ITM, the best is chosen anyway (rules R.W42A, R.W42B, R.W52). Move generation and selection is cyclic, as in SOLVE1.

Consider the following problem:

A	B	C
35	9	31

GOAL
48

This problem requires the successful problem solver to move to a new state which is "unacceptable" in order to find the following solution: pour in A; pour in C; pour out B; pour out B.

Even if experimentation upheld this division into easy and hard problems based upon the applicability of the acceptability heuristic, the model would not reflect subject's behavior when given a series of such problems, some of which had a common solution. Luchins and Luchins (1950) have shown that subjects develop a tendency to produce the common solution when possible, even when more direct solutions exist.

The tendency to develop such a mental set or Einstelling is added to the problem solving model through the use of the production system CONTEXT, which is also presented in Appendix III. This production system can be activated between the presentation of problems. Between problems, ITM is automatically cleared of all problem solving search residue, leaving only the set of previous solutions as memory context between problems. CONTEXT looks in ITM to determine if two recent solutions are the same. If so, conditions are created and ADAPT is activated to form a condition list (rule R.WL4). This condition list is saved in ITM, while construction of the action list proceeds by creating actions corresponding to steps in the common solution (rules R.WL3, R.WL1). Finally the new rule is inserted at the top of SOLVE1 (rule R.WL2).

This new rule will fire first when SOLVE1 is activated and will carry out the common solution sequence. If this produces the goal state, ANSWER is activated, which will report the repeated solution rule (R.WA0). If the common solution does not apply, SOLVE1 will revert to the means-ends search strategy.

Conclusion

Three forms of adaptive behavior have been discussed. A production system implementation of each has been presented. Each implementation illustrates the role of the general adaptation production system ADAPT. ADAPT is a general utility process for production system adaptation. General guiding principles for the control of such adaption is a current research topic.

References

1. Atwood, M. and Polson, P., "A process model for water jug problems," Cognitive Psychology, 1976, p. 191-216.

2. Bourne, L., Eckstrand, B., Dominowski, R., The Psychology of Thinking, Prentice Hall: Englewood Cliffs, N.J., 1971.

3. Bruner, J., Goodnow, J., Austin, G., A Study of Thinking, John Wiley : New York.

4. Craik, F. and Lockhart, R., "Levels of processing: a framework for memory research," Journal of Verbal Learning and Verbal Behavior, 1972, 11, 671-684.

5. Farley, A., "PSMON, a production system monitor," Technical Report CS-76-1, Univ. of Oregon, Computer Science Dept., 1976.

6. Feigenbaum, E. "The simulation of verbal learning behavior," Proc. WJCC, 1961, p. 121-132.

7. Feigenbaum, E. and Simon, H., "A theory of the serial position effect," British Journal of Psychology, 1962, 53, 30 7-320.

8. Greeno, J., "The structure of memory and the process of solving problems," in Solso (ed), Contemporary Issues in Cognitive Psychology, Winston: Washington, D.C., 1973.

9. Hedrick, C., "Learning Production Systems from Examples," Artificial Intelligence, 1976, 7, p. 21-50.

10. Hintzman, D., "Explorations with a discrimination net model for paired-associate learning," Journal of Mathematical Psychology, 1968, 5, p. 123-162.

11. Hunt, E., Marin, J., Stone, P., Experiments in Induction, Wiley: New York, 1966.

12. Keele, S., Attention and Human Performance, Goodyear: Pacific Palisades, CA, 1973.

13. Levine, M., A Cognitive Theory of Learning, L. Erlbaum: Hillsdale, N.J., 1975.

14. Lilly, J., "Programming and meta-programming the human biocomputer," 1969.

15. Luchins, A., "Mechanization in problem solving," Psychological monographs, 1942, 54, #6 (whole #248).

16. Newell, A., "Production Systems: Models of Control Structures," in Chase (ed.), Visual Information Processing, Academic: New York, 1973.

17. Newell, A. and Simon, H., Human Problem Solving, Prentice Hall: Englewood Cliffs, N.J., 1972.

18. Simon, H. and Reed, S., "Modeling strategy shifts in a problem solving task," Cognitive Psychology, 1976, p. 86-97.

19. Waterman, D., "Generalization techniques for automating the learning of heuristics," Artificial Intelligence, 1970, 1, 121-170.

20. Waterman, D., "Adaptive Production Systems," CIP report, Carnegie-Mellon University, 1973.

21. Waterman, D., "PAS-II Reference Mannual, " Psychology Dept., Carnegie-Mellon University, 1973.

```
INITIAL STM
 STM:  () () () () () () ()

ACTIVATE EPAM                                        APPENDIX I

R.E4 FIRES       GET NEW ENVIRONMENT

CURRENT ENVIRONMENT
1•P
3•J
2•A

 STM:  () () () () () () ()

ACTIVATE ANSWER

R.A1ØØ FIRES     DON'T KNOW ANSWER, SO SAY ?

 STM:  (SAY :: ?) (TRUE :: 1•P) () () () () ()

DEACTIVATE ANSWER

ACTIVATE EPAM

R.E3 FIRES       GET CORRECT ANSWER

 STM:  (HEAR :: CON) (SAY :: ?) (TRUE :: 1•P) () () ()
          ()

R.E2 FIRES       GAVE WRONG ANSWER SO LEARN NEW RULE FOR ANSWER PS

 STM:  (USED :: ?) (HEAR :: CON) (TRUE :: 1•P) () () ()
          ()

ACTIVATE LEARN

R.L2 FIRES       CREATE CONDITION CHUNK

 STM:  (CND :: (EØ: IS('1•P'))) (HEAR :: CON) (USED :: 1•P)
          (USED :: ?) () () ()

R.L4 FIRES       ACTIVATE ADAPT TO CREATE CONDITION LIST

 STM:  (HEAR :: CON) (CND :: (EØ: IS('1•P'))) (USED :: 1•P)
          (USED :: ?) () () ()

ACTIVATE ADAPT

R.2 FIRES        CREATE CONDITION LIST

 STM:  (CNDL ::  CONDITION LIST ) (USED :: (EØ: IS('1•P')))
          (HEAR :: CON) (USED :: 1•P) (USED :: ?) () ()
```

```
CURRENT ENVIRONMENT

1*B
2*F
3*I

SAY:     ?


CURRENT STATE OF ADAPTING AS 'ANSWER'

R,A1:  NEW RULE
        (E0: IS('1*P') )
                               ==>
                                              (SAY('CON'))
                                              (DEACT())
ENDRULE

R,A2:  NEW RULE
        (E0: IS('1*B') )
                               ==>
                                              (SAY('DEE'))
                                              (DEACT())
ENDRULE

R,A100:          DON'T KNOW ANSWER, SO SAY ?
        (E0: MATCH(ASPECT) )
                               ==>
                                              (SAY('?'))
                                              (MARK(E0,TRUE))
                                              (DEACT())
ENDRULE


CURRENT ENVIRONMENT

1*P
3*J
2*U

SAY:  CON    ****
```

```
CURRENT STATE OF ADAPTING PS 'ANSWER'

R.A3:  NEW RULE
        (E1: IS('1*P') )
        (E0: IS('3*J') )
                         =>
                                  (SAY('LEQ'))
                                  (DEACT())
ENDRULE


R.A1:  NEW RULE
        (E0: IS('1*P') )
                         =>
                                  (SAY('CON'))
                                  (DEACT())
ENDRULE


R.A4:  NEW RULE
        (E1: IS('1*B') )
        (E0: IS('3*T') )
                         =>
                                  (SAY('FIP'))
                                  (DEACT())
ENDRULE


R.A2:  NEW RULE
        (E0: IS('1*B') )
                         =>
                                  (SAY('DEZ'))
                                  (DEACT())
ENDRULE


R.A100:            DON'T KNOW ANSWER, SO SAY ?
        (E0: MATCH(ASPECT) )
                         =>
                                  (SAY('?'))
                                  (MARK(E0,TRUE))
                                  (DEACT())
ENDRULE



  CURRENT ENVIRONMENT

  1*P

  3*J

  2*A


  SAY:   LEQ    ****
```

APPENDIX II

```
*
*          PRODUCTION SYSTEM FILE FOR ADAPTIVE PS
*          IMPLEMENTATION OF CONCEPT FORMATION
*
*
*          INITIAL 'CONCEPT' PRODUCTION SYSTEM
*
*
R,C0:      ANSWER NO, NOT AN EXAMPLE OF CONCEPT
           (E0: MATCH('FEATURE'))
           (E1: MATCH('FEATURE'))
           (E2: MATCH('FEATURE'))
                       ==>
                                      (SAY(NO))
                                      (DEACT())
ENDRULE
*
*
*          RULES WHICH IN CONJUNCTION WITH ADAPT
*          CREATE A CONCEPT RULE STRUCTURE IN
*          THE PRODUCTION SYSTEM 'CONCEPT'.
*
*
R,L0:      GET FEEDBACK ON STIMULUS
           (C0: MARKED(SAY))
           (C1: NO('MARKED(HEAR)'))
                       ==>
                                      (LISTEN())
ENDRULE
*
*
R,L00:     CREATE ACTIONS AND ACTIVATE ADAPT FOR RULE INSERTION
           (C0: MARKED(LOC))
           (C1: MARKED(CNOL))
                       ==>
                                      (NEWACT('DEACT()'))
                                      (NEWACT('SAY(YES)'))
                                      (ACTIVE('CONCEPT'))
                                      (ACTIVE('ADAPT'))
ENDRULE
*
*
R,L000:    REMOVE OLD VERSION OF ALTERED RULE
           (C0: MARKED(LOC) .AND. IS(REMOVE))
                       ==>
                                      (ACTIVE('CONCEPT'))
                                      (ACTIVE('ADAPT'))
ENDRULE
R,L1:      PREPARE TO ACCEPT NEW ENVIRONMENT AS ANSWER CORRECT
           (C0: MARKED(HEAR) .AND. IS(NO))
           (C1: MARKED(SAY) .AND. IS(NO))
           (C2: MARKED(NEW))
           (C3: MARKED(NEW))
           (C4: MARKED(NEW))
                       ==>
                                      (MARK('C0,C1,C2,C3,C4',USED))
ENDRULE
*
```

```
*
R.L4A:  PREPARE TO REMOVE RULE LEADING TO INAPPROPRIATE YES
        (CØ: MARKED(HEAR) .AND. IS(NO))
        (C1: MARKED(SAY) .AND. IS(YES))
        (C2: MARKED(NEW))
        (C3: MARKED(NEW))
                          *=>
                                   (NEWCOND('E',("IS('" CONTENT(C2) "')")))
                                   (NEWCOND('E',("NO('" 'IS("' CONTENT(E1) '"
                                   (NEWLOC(REMOVE))
                                   (MARK('CØ,C1,C2,C3',USED))
                                   (ACTIVE('CONCEPT'))
                                   (ACTIVE('ADAPT'))
ENDRULE
*
*
R.L4B:  PREPARE TO REPLACE RULE LEADING TO INAPPROPRIATE YES
        (CØ: MARKED(NEW))
        (EØ: IS(CONTENT(CØ)))
        (E1: MATCH('FEATURE'))
        (IØ: NO('IS(CONTENT(E1))'))
                          *=>
                                   (NEWCOND('E',("NO('" 'IS("' CONTENT(E1) '"
                                   (NEWLOC(REMOVE))
                                   (NEWLOC(AFTER))
                                   (MARK('CØ,EØ,E1',USED))
                                   (ACTIVE('ADAPT'))
ENDRULE
*
*
R.L4C:  PREPARE TO REPLACE RULE LEADING TO INAPPROPRIATE YES
        (CØ: MARKED(NEW))
        (EØ: IS(CONTENT(CØ)))
        (E1: MATCH('FEATURE'))
        (E2: MATCH('FEATURE'))
        (IØ: NO('IS(CONTENT(E2))'))
                          *=>
                                   (NEWCOND('E',("NO('" 'IS("' CONTENT(E2) '
                                   (NEWLOC(REMOVE))
                                   (NEWLOC(AFTER))
                                   (MARK('CØ,EØ,E1',USED))
                                   (ACTIVE('ADAPT'))
ENDRULE
R.L4E:  PREPARE TO ALTER RULE LEADING TO INAPPROPRIATE YES
        (CØ: MARKED(HEAR) .AND. IS(NO))
        (C1: MARKED(SAY) .AND. IS(YES))
        (C2: MARKED(NEW))
        (EØ: IS(CONTENT(C2)))
        (E1: MATCH('FEATURE'))
                          *=>
                                   (NEWCOND('E',("IS('" CONTENT(C2) "')")))
                                   (NEWCOND('E',("NO('" 'IS("' CONTENT(E1) '
                                   (NEWLOC(REMOVE))
                                   (NEWLOC(AFTER))
                                   (MARK('CØ,C1,C2,EØ,E1',USED))
                                   (ACTIVE('ADAPT'))
ENDRULE
*
*
```

APPENDIX III

```
*
*
*       PRODUCTION SYSTEM FILE FOR THE SOLUTION
*       OF WATER JUG PROBLEMS BY THE MEANS - ENDS
*       ANALYSIS APPROACH.
*
*
R.WØ:   REMEMBER CHOSEN MOVE AND SET UP NEW STATE
        (CØ: MARKED(CHOICE) .AND. GETDIF('T1'))
                         ==>
                                 (SAVE(CØ))
                                 (MARK(CØ,OLD))
                                 (NEW('C1',T1))
                                 (MARK(C1,STATE))
ENDRULE
*
*
R.W1A:  GENERATE INMOVE TO COMPLETE SOLUTION
        (CØ: MARKED(STATE) .AND. ISLT(Ø))
        (EØ: MATCH('JUG DIGITS , T1') .AND. EQ((CONTENT(CØ) + T1),Ø))
                         ==>
                                 (NEW('C1',GENINMOVE(ISJUG,T1,CONTENT(C
                                 (MARK(C1,CHOICE))
                                 (MARK(CØ,STATE,))
                                 (DEACT())
                                 (ACTIVE('ANSWER'))
ENDRULE
*
*
R.W1B:  GENERATE OUTMOVE TO COMPLETE SOLUTION
        (CØ: MARKED(STATE) .AND. ISGT(Ø))
        (EØ: MATCH('JUG DIGITS , T1') .AND. EQ((CONTENT(CØ) - T1),Ø))
                         ==>
                                 (NEW('C1',GENOUTMOVE(ISJUG,T1,CONTENT(
                                 (MARK(C1,CHOICE))
                                 (MARK(CØ,STATE,))
                                 (DEACT())
                                 (ACTIVE('ANSWER'))
ENDRULE
*
*
R.W2A:  GENERATE IN-MOVES AS STATE LESS THAN ZERO
        (EØ: MATCH('JUGA DIGITS , T1'))
        (E1: MATCH('JUGB DIGITS , T2'))
        (E2: MATCH('JUGC DIGITS , T3'))
        (CØ: MARKED(STATE) .AND. ISLT(Ø))
                         ==>
                                 (NEW('C1',GENINMOVE('A',T1,CONTENT(CØ)
                                 (NEW('C2',GENINMOVE('B',T2,CONTENT(CØ)
                                 (NEW('C3',GENINMOVE('C',T3,CONTENT(CØ)
                                 (MARK(CØ,STATE,))
                                 (MARK('C1,C2,C3',MOVE))
ENDRULE
*
*
R.W2B:  GENERATE OUT-MOVES AS STATE GREATER THAN ZERO
        (EØ: MATCH('JUGA DIGITS , T1'))
        (E1: MATCH('JUGB DIGITS , T2'))
        (E2: MATCH('JUGC DIGITS , T3'))
        (CØ: MARKED(STATE) .AND. ISGT(Ø))
```

```
*
*
R.W6B:  REJECT LESSER MOVE
        (CØ: MARKED(MOVE) .AND. GETDIF('T1'))
        (C1: MARKED(MOVE) .AND. WORSE(GETDIF(),T1))
                        ==>
                                   (MARK(C1,OLD))
ENDRULE
*
*
R.W7:   SELECT A MOVE
        (CØ: MARKED(MOVE))
                        ==>
                                   (MARK(CØ,CHOICE))
ENDRULE
*
*
R.W8:   HAVE TRIED ALL INITIAL MOVES: BEGIN PHASE TWO
        (CØ: MARKED(STATE.))
        (EØ: MATCH('GOAL DIGITS . T1') .AND. EQ(-T1,CONTENT(CØ)))
                        ==>
                                   (MARK(CØ,STATE))
                                   (DEACT())
                                   (ACTIVE('SOLVE2'))
ENDRULE
*
*
R.W9:   START FROM INITIAL STATE
        (EØ: MATCH('GOAL DIGITS . T1'))
                        ==>
                                   (NEW('CØ',('-' T1)))
                                   (MARK(CØ,STATE))
ENDRULE
*
*
*       RULES WHICH ALLOW THE SYSTEM TO RELAX ITS MOVE
*       SELECTION CRITERION IN PHASE TWO OF PROBLEM
*       SOLVING WHEN INITIAL PHASE FAILS TO FIND ANSWER.
*
*
*
R.W12:  CHOOSE ACCEPTABLE MOVE TO NEW STATE
        (CØ: MARKED(STATE.))
        (C1: MARKED(MOVE) .AND. GETDIF('T1') .AND. WORSE(CONTENT(CØ),
        (IØ: MARKED(CHOICE) .AND. NO('EQ(GETDIF(),T1)'))
                        ==>
                                   (MARK(C1,CHOICE))
ENDRULE
*
*
R.W22:  MARK ACCEPTABLE MOVE AS LEADING TO SEEN STATE
        (CØ: MARKED(STATE.))
        (C1: MARKED(MOVE) .AND. WORSE(CONTENT(CØ),GETDIF()))
                        ==>
                                   (MARK(C1,OLD))
ENDRULE
*
*
R.W32:  CHOOSE PREVIOUSLY UNACCEPTABLE MOVE TO NEW STATE
        (CØ: MARKED(MOVE) .AND. GETDIF('T1'))
```

```
                          ==>
                                      (NEW('C3',(GETMOVE(CONTENT(I0)) '!' CONT
                                      (NEW('C4',GETSTATE(CONTENT(I0))))
                                      (MARK('C0,C1',OLD))
                                      (MARK(C3,ANS))
                                      (MARK(C4,STATE))
ENDRULE
*
*
R,A2:   SAY ANSWER AS STRING IS COMPLETE
        (C0: MARKED(ANS))
                          ==>
                                      (SAY(CONTENT(C0)))
                                      (SAVE(C3))
                                      (DEACT())
ENDRULE
*
*
R,A3:   BEGIN ANSWER BUILDING PROCESS
        (C0: MARKED(CHOICE) .AND. EQ(GETDIF(),0))
                          ==>
                                      (NEW('C1', GETMOVE(CONTENT(C0))))
                                      (NEW('C2', GETSTATE(CONTENT(C0))))
                                      (MARK(C1,ANS))
                                      (MARK(C2,STATE))
ENDRULE
*
*
*       RULES WHICH ALLOW THE SYSTEM TO NOTE REPEATING
*       SOLUTIONS AND TO BUILD NEW RULES TO TRY THESE
*       SOLUTIONS PRIOR TO A GENERAL MEANS-ENDS SEARCH.
*
*
*
R,L1:   CREATE ANOTHER_MOVE STEP FOR THE SOLUTION RULE
        (C0: MARKED(ACTL))
        (C1: MARKED(ANS) .AND. GETSTEP('T1'))
                          ==>
                                      (NEWACT(GENMOVE(T1,'GETDIF(,C1)')))
                                      (ACTIVE('ADAPT'))
ENDRULE
*
*
R,L2:   GATHER ACTION AND CONDITION LISTS IN STM FOR RULE CREATION
        (C0: MARKED(ACTL))
        (,?: MARKED(CNDL))
                          ==>
                                      (DEACT())
                                      (ACTIVE('SOLVE1'))
                                      (ACTIVE('ADAPT'))
ENDRULE
*
*
R,L3:   SAVE CONDITION_LIST AND BEGIN ACTION CONSTRUCTION
        (C2: MARKED(CNDL))
        (I0: MARKED(ANS))
        (I1: MARKED(ANS) .AND. ISSAME(CONTENT(I0)) .AND. GETSTEP('T1'
                          ==>
                                      (SAVE(C0))
                                      (MARK('C0,I0',USED))
                                      (NEWACT('MARK(C0,USED)'))
```