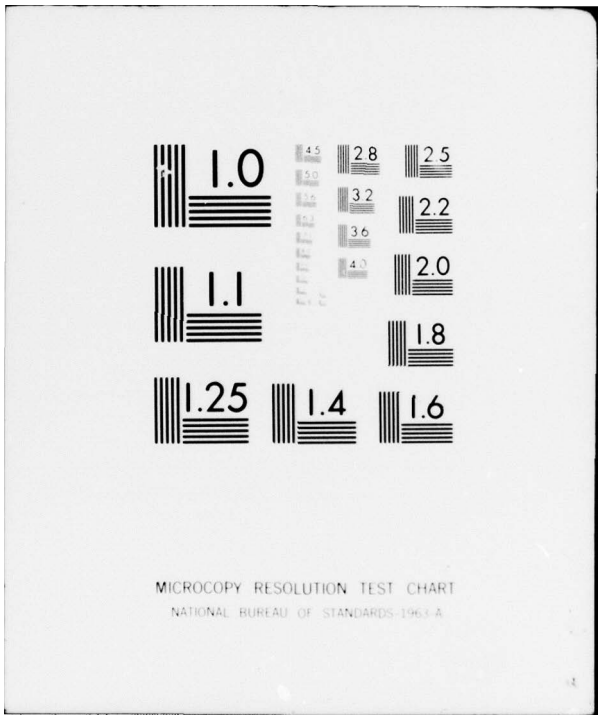
AD-A040 891 RAND CORP SANTA MONICA CALIF F/G 9/2
AN INTRODUCTION TO PRODUCTION SYSTEMS,(U)
NOV 76 D A WATERMAN
UNCLASSIFIED P-5751 NL

1 OF 1
AD A040891

END

DATE FILMED
7-77


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

(2) P.S.

(6) AN INTRODUCTION TO PRODUCTION SYSTEMS

(10) D. A. Waterman

(11) November 1976

(12) 15 p.

(14) P-5751

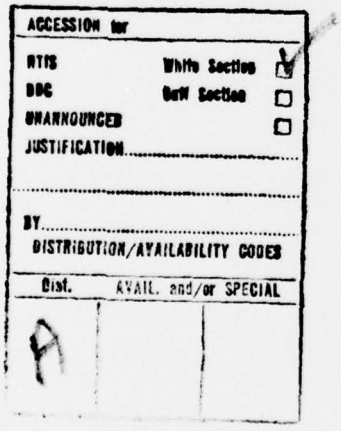
The Rand Paper Series

Papers are issued by The Rand Corporation as a service to its professional staff. Their purpose is to facilitate the exchange of ideas among those who share the author's research interests; Papers are not reports prepared in fulfillment of Rand's contracts or grants. Views expressed in a Paper are the author's own, and are not necessarily shared by Rand or its research sponsors.

The Rand Corporation
Santa Monica, California 90406

ABSTRACT

Production systems provide a simple, uniform way of handling control flow and data management in programs which exhibit intelligent behavior. They are particularly useful for developing programs which can learn from experience, i.e., which can demonstrate adaptive behavior. In this brief introduction to the subject the concept of the production system is defined, and simple examples of production systems are presented. Current applications of production system technology are also discussed.



I. BASIC CONCEPTS

This paper is intended to serve as a brief introduction to production systems, the name given to a class of programs which embody special simplifying constraints with regard to control flow and data management. The basic constraint is that all program statements are of the form "if C then A," i.e., if the conditions C are true then perform the actions A. The constraints lead to system characteristics that facilitate writing programs which exhibit intelligent behavior. The term "production" stems from its use by Post in his symbol manipulation systems (Minsky, 1967). These were systems composed of grammar-like rules for specifying string replacement operations. A typical rule in such a system might have the form AYB -> AZB, meaning that any occurrence of the string Y in the context of A and B would be replaced by the string Z. A production system is actually a generalization of Markov normal Algorithms (Galler and Perlis, 1970), which are collections of ordered string replacement rules. The first applicable rule that is found is applied, and then testing starts over, beginning with the highest priority rule. The cycle continues until no rules are applicable. An example of a simple Markov normal Algorithm which transforms any string composed of a's, and b's into the string "AB" is shown below.

```
Alphabet:     a,b,A,B

Variables:    x

Productions:  1.  aa -> a
              2.  bb -> b
              3.  xa -> ax
              4.   a -> A
              5.   b -> B
```

If the initial string is "bbaba," rules 2,3,2,3,1,4,5 are applied in that order to produce "AB." Notice that the highest priority productions must be applied first to insure that the algorithm will terminate and give the desired result.

A production system is a collection of rules of the form conditions -> actions (Newell and Simon, 1972), where the conditions are statements about the contents of a global data base, and the actions are procedures which may modify the contents of that data base. The conditions and actions are not restricted to string matching and replacement; a condition can be any expression which has a truth value that can be determined from the data base, and an action can include any operation which modifies the data base. When the conditions of a production rule are true the rule can "fire," which means that the actions associated with the true conditions are executed. The activity involved in firing a rule, determining which rules have true conditions, selecting one of them, and executing its actions, is considered one cycle through the system, and can be characterized as a RECOGNIZE-ACT cycle. A production system cycles continuously, halting when the conditions for all production rules are false or a special halting action is executed. Thus production rules fire in a data-dependent fashion, operating quite differently from typical programs which have sequential control or explicit knowledge about where code will next be executed in the program.

II. CLASSIFICATION OF PRODUCTION SYSTEMS

Production systems can be divided into two main classes based on the way the rules make contact with the data base. The conventional production system is condition-driven, that is, the conditions of the rules are compared to the data base and the rules whose conditions match are selected and have their actions executed. This is the scheme used in a number of current systems (Newell and Simon, 1972; Newell, 1973; Waterman, 1975). The other type of production system makes contact with data base through the actions rather than the conditions (Shortliffe et al, 1975; Davis, 1976). These action-driven systems have rules analogous to logical implication statements, i.e., A & B & C -> D, means that *if A, B, and C are true then D is true.* Here the system would try to show (prove) that D is true by first looking for D in the data base and if that fails by showing that A, B, and C are true.

The term production system is somewhat ambiguous. It has been used to refer to both the general framework or language within which production rules can be defined, and to specific sets of production rules written within the framework. Here the term production system architecture will refer to the system one can use to define programs, and production system to a specific program or set of production rules for accomplishing some task.

Condition-driven Production Systems

During the RECOGNIZE step in a condition-driven production system, the conditions of all rules are compared to the data base and those whose conditions are "true" are selected. The set of all rules which

have true conditions during recognition is called the conflict set, and the problem of choosing a single rule from the conflict set (for subsequent execution) is called conflict resolution. During the ACT step of the control cycle all actions associated with the chosen rule are executed. Conflict resolution typically provides a single rule for execution (selection and execution of multiple rules is possible but more complex). Although many different conflict resolution techniques exist, the most common, rule order, involves assigning a priority ordering to the rules when they are first created. Conflict resolution then consists of selecting the highest priority rule from the conflict set. We will call production systems which use the rule order technique for conflict resolution, rule ordered, or just "ordered" production systems.

The operation of this type of production system will be illustrated by showing one which can add two integers. The algorithm, shown below, adds M and N using the intermediate variable COUNT.

```
1.  add(M,N) = COUNT <- 0; read(M,N);
2.  L1 if COUNT = M then return(N);
3.  COUNT <- successor(COUNT);
4.  N <- successor(N);
5.  goto(L1).
```

The ordered production system corresponding to the above algorithm is given below. Here conjunction is indicated by & and variables by x's, thus rule 3 says, in effect, "if the string (COUNT [any item]) is in the data base, and the string (N [any item]) is also in the data base, then change the data base by replacing the item associated with COUNT with the successor of that item, and by replacing the item associated with N with the successor of that item. Thus applying rule 3 to the data base elements (COUNT 0) and (N 1) would change these elements to

(COUNT 1) and (N 2).

```
DATA BASE:  (ADD)

RULES:  1.                                (ADD)  -> del(1)
                                                    put((COUNT 0))
                                                    read()
        2.  (COUNT x1) & (M x1) & (N x2)   -> say(x2)
                                                    stop()
        3.           (COUNT x1) & (N x2)   -> rep(1, x1, succ(x1))
                                                    rep(2, x2, succ(x2))
```

The actions used in this production system are defined below.

```
     del(n) : deletes the data element that
              matched the nth condition element.
     put(b) : puts b into the data base.
     read() : reads data into the data base.
     say(b) : prints b.
     stop() : stops production system execution.
 rep(n,b,c) : replaces b with c in data element
              that matched the nth condition element.
    succ(n) : returns n+1, the successor of n.
```

A trace of the execution of the production system is given below. It is assumed here that read() brings (N 1) (M 2) into the data base. Note that line 1 of the algorithm corresponds to production rule 1, line 2 to rule 2, and lines 3 and 4 to rule 3. There is nothing in the production system that corresponds directly to the GOTO statement in line 5, since the branching function in production systems is handled by control cycle repetition, which permits unlimited looping, and appropriate data base modification, which in this case makes rule 1 inoperative after it is fired once.

```
DATA BASE:  (ADD)
  rule 1 fires
DATA BASE:  (N 1) (M 2) (COUNT 0)
  rule 3 fires
DATA BASE:  (N 2) (M 2) (COUNT 1)
  rule 3 fires
DATA BASE:  (N 3) (M 2) (COUNT 2)
  rule 2 fires
  3 [is printed and execution stops]
```

Action-driven Production Systems

An action-driven production system is given a premise to "prove," or in effect a question to answer through deductive inference. The actions of rules are examined to find one which could make the premise true. When such a rule is found it is examined to see if all its conditions are true. If they are, the rule is fired; if not, the process continues recursively in an attempt to show that each condition of the rule is true.

An abstract example of this type of production system is shown below. Here data base elements are letters and are considered true if they are in the data base.

```
DATA BASE:  A F

    RULES:  1.  A & B & C -> D
            2.      D & F -> G
            3.      A & J -> G
            4.          B -> C
            5.          F -> B
            6.          L -> J
            7.          G -> H
```

If the goal is to show that H is true, the system first checks the data base to see if H is there. In this case it is not, so the system tries to deduce that H is true using the rules that have H on the right-hand side. The only one applicable is rule 7. The system now attempts to show that G is true, since if G is true then H is also true. Again it checks the data base; G is not there, so it looks for rules that have G on the right-hand side. Conflict resolution for this example is considered to be rule ordered, thus the first (highest priority) rule that is applicable is used. This

is Rule 2, so now the goal is to show (or deduce) that D and F are true. This is accomplished by showing that A is true (from the data base), B is true (from rule 5), and C is true (from rule 4). Since D and F are true, G is true and thus H is true, and the goal has been accomplished. As the applicable rules are fired the appropriate elements are added to the data base. In this case the data base ends up with the elements: H G D C B A F, inserted by rules 5, 4, 1, 2, and 7, in that order.

III. APPLICATIONS OF PRODUCTION SYSTEM TECHNOLOGY

Production system architectures have been used in a number of different systems. An interesting example of a condition-driven architecture is the Meta-DENDRAL system (Buchanan et al, 1972). Meta-DENDRAL is a program designed to formulate rules of mass spectrometry which can be used by Heuristic DENDRAL, a performance program developed for the analysis of molecular structures. The rules learned by meta-DENDRAL are represented as production rules of the form situation -> process, where each situation is a description of a subgraph which represents some class of molecular structures, and each process is an action that will change those structures, such as breaking a bond or moving an atom. Condition testing is based on pattern matching, and all rules with true conditions are executed in some arbitrary order.

MYCIN is an interesting example of an action-driven production system architecture (Shortliffe et al, 1975) The MYCIN program is a production system designed to interact with a physician and advise him regarding antimicrobial therapy selection. The system uses over 200 decision rules to guide its action-directed search for a diagnosis. It not only uses the data base and the rules to validate rule conditions, but also queries the user of the system (the physician) when the information is not in the data base and cannot be deduced from the rules. Thus the user is able to provide the system with current information about the particular case being diagnosed.

A system which uses both condition-driven and action-driven rules is RITA (Anderson and Gillogly, 1976). The RITA system is designed

for writing computer programs called agents which intelligently interface the user to the outside computer world. RITA's production system control structure provides the degree of simplicity and modularity needed to make program organization straightforward and program modification relatively easy. The system is human engineered, i.e., the programs or RITA agents have an English-like syntax which makes them easy to write and almost self-documenting. The language primitives in RITA permit the user to interact with other computer systems, even to the extent of initiating and monitering several jobs in parallel on external systems.

Within RITA, agents can be created which are entirely condition-driven (also called pattern-driven), entirely action-driven (also called goal-driven), or are some combination of both. The condition-driven production rules are called RULES, the action-driven ones are called GOALS, and they both operate on a data base composed of objects with associated attributes and values. An example of a simple RITA agent that deduces the prompt characters for all ARPAnet hosts in the data base is shown below.

[DATA BASE]

```
OBJECT computer<1>:
        name             IS  "rand-unix",
        type             IS  "PDP-11",
        operating-system IS  "UNIX",
        access-link      IS  "ARPAnet";

OBJECT computer<2>:
        name             IS  "sri-ai",
        type             IS  "PDP-10",
        operating-system IS  "TENEX",
        access-link      IS  "ARPAnet",
        prompt-character IS  "@";
```

```
                    [RULE SET]

RULE 1:

    IF: THERE IS a computer WHOSE access-link IS "ARPAnet"
                AND WHOSE prompt-character IS NOT KNOWN

  THEN: DEDUCE the prompt-character OF the computer
        & SEND concat (the name OF the computer, "uses a prompt of ",
                the prompt-character OF the computer) TO user;

GOAL 1:

    IF: THERE IS a computer WHOSE type is "PDP-11"
                AND WHOSE operating-system IS "UNIX"
                AND WHOSE access-link IS "ARPAnet"
                AND WHOSE prompt-character IS NOT KNOWN

  THEN: SET the prompt-character OF the computer TO "%";
```

When this agent is executed Rule 1 fires, because both its premises are true, initiating a deduction for the prompt-character of computer <1>. Since GOAL 1 is applicable, it is used in the deduction and automatically updates the data base. Then the second action of RULE 1 is executed and the sentence "rand-unix uses a prompt of %" is printed.

In conclusion, production systems are an interesting form of program organization for a number of reasons. First, they provide a parsimonious way of modeling human cognition, i.e., the production system data base can be compared to human short term memory, and the production rules to human long term memory. Second, production rules tend to represent independent components of behavior and thus the creation and addition of new production rules can be incremental, a feature which facilitates modeling learning processes (Waterman, 1970, 1975). Third, when a large body of knowledge is represented in rule form, as in MYCIN, it becomes easier to explain, justify, and analyze the rationale used by the program to reach its decisions. Finally, the simplicity of the RECOGNIZE-ACT

control structure (no branching or block structure) facilitates automatic program creation, debugging, and verification.

REFERENCES

Anderson, R. H., and Gillogly, J. J., Rand intelligent terminal agent (RITA): Design philosophy. Rand Report R-1809-ARPA, February, 1976.

Buchanan, B. G., Feigenbaum, E. A., and Sridharan, N. S., Heuristic theory formation: data interpretation and rule formation. In Machine Intelligence 7, Edinburgh University Press, 1972.

Davis, Randall, Applications of meta level knowledge to the construction, maintenance and use of large knowledge bases, Report No. STAN-CS-76-552, Computer Science Department, Stanford University, 1976.

Galler, B., and Perlis, A., A View of Programming Languages, Addison-Wesley, 1970.

Minsky, M., Computation: Finite and Infinite Machines, Prentice-Hall, 1967.

Newell, A., and Simon, H. A., Human Problem Solving, Prentice-Hall, 1972.

Newell, A., Production systems: Models of control structures. In W. C. Chase (ed.), Visual Information Processing, 1973, pp. 463-526.

Shortliffe, E. H., Davis, R., Buchanan, B., Axline, S., Green, C., and Cohen, S., Computer-based consultations in clinical therapeutics: exploration and rule acquisition capabilities of the MYCIN system. Computers and Biomedical Research, Vol. 8, 1975, pp. 303-320.

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

Waterman, D. A., Adaptive production systems. 4th IJCAI Conference Proceedings, September, 1975, pp. 296-303.