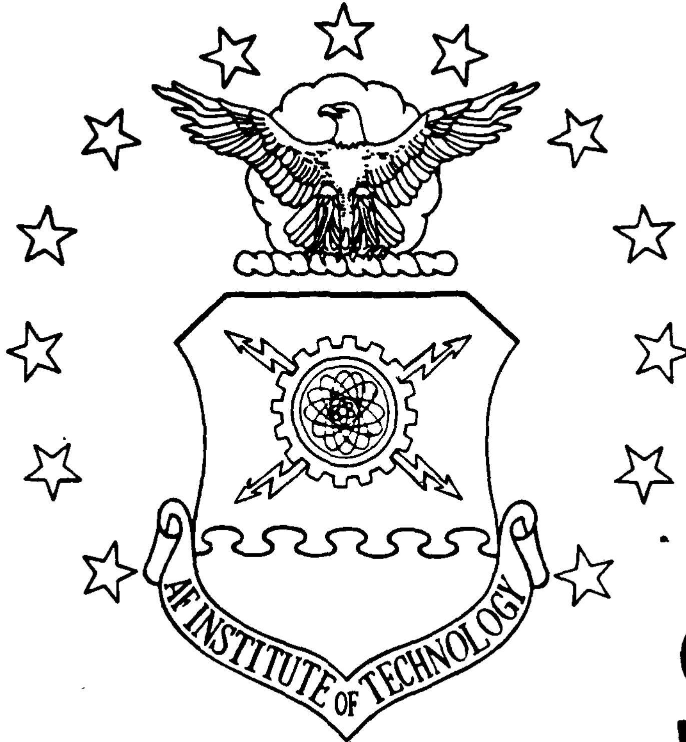
DTIC FILE COPY

(1)

AD-A202 563



DTIC
ELECTE
JAN 23 1989
S H D

SIMULATING RULE-BASED SYSTEMS

THESIS

Nizar Mahmoud Mahaba
Lieutenant Colonel, Egyptian Army

AFIT/GOR/ENS/88D-12

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

89 1 17 144

AFIT/GOR/ENS/88D-12

SIMULATING RULE-BASED SYSTEMS

THESIS

Nizar Mahmoud Mahaba
Lieutenant Colonel, Egyptian Army

AFIT/GOR/ENS/88D-12

DTIC
ELECTE
JAN 23 1989
S D
H

Approved for public release; distribution unlimited

AFIT/GOR/ENS/88D-12

SIMULATING RULE-BASED SYSTEMS

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Operations Research

Nizar Mahmoud Mahaba
Lieutenant Colonel, Egyptian Army

December 1988

Approved for public release; distribution unlimited

Acknowledgements

I wish to thank Major Bruce Morlan, my thesis advisor, for adopting the idea of this research, for the useful discussion of the details of the model, and for facilitating the communication with other faculty members. I also wish to thank Dr. Frank Brown for enriching the work by his technical and grammatical expertise. A word of thanks is also owed to Major Joseph Litko and Major Kenneth Bauer for the help they offered.

Accession For
NTIS GRA&I
DTIC TAB
Unannounced
Justification
By
Distribution/
Availability Codes
Avail and/or
Dist Special
A-1

Table Of Contents

Page

Acknowledgements ii

List of Figures v

List of Tables vi

Abstract vii

I. Introduction 1

II. Rule-Based Systems 7

Basic Architecture 8
Data Memory 12
Rule Memory 14
Control Strategies 16

III. Efficiency of Rule-Based Systems 22

Efficiency of Implementation
Efficiency of Consultation 33
A Comparative Study of Control Strategies
for Expert Systems 38

IV. The Description of the Model 40

Features Presented in the Model 41
Limitations 42
Measure of Effectiveness 43
Model Structure 44
Assertions 51
The Rule Generator 54

V. The Simulated Inference Engines 70

Engine 1 72
Engine 2 73
Engine 3 74
Engine 4 74
Engine 5 74
Engine 6 75
Verification 78

VI. Design of Experiments 80

Type I Experiments: Performance of a Specific RBS 84
Type II Experiments: Comparison of the Matching Effort for Inference Engines . . . 95

VII. Summary and Recommendations 105

Summary
Recommendations 108

Bibliography 111

Appendix: The Computer Programs 112

List of Figures

Figure		Page
1.	Architecture of a Production System Model . . .	10
2.	Summary of the Rete Matcher	32
3.	Summary of Techniques for Improving Efficiency in OPS5	34
4.	The Main components of the Model	47
5.	The relations Among the Initial Assertions . . .	53
6.	The Rule Generator	56
7.	The Condition-Membership Filter	77
8.	Basic Sequence of Experimentation	83
9.	Type 1 Experiments	87
10.	The Effect of Fact Ordering on Monotonic RBSs . .	89
11.	The Effect of Fact Ordering on Non-Monotonic RBSs	89
12.	The Effect of Rule-Ordering on Monotonic RBSs . .	94
13.	Type 2 Experiments	96
14.	The Effect of Controlled-Production Filter on Monotonic RBSs	98
15.	The Effect of Controlled-Production Filter on Non-Monotonic RBSs	98
16.	The Effect of Context-Restricted Filter on Monotonic RBSs	100
17.	The Effect of Context-Restricted Filter on Non-Monotonic RBSs	100
18.	The Effect of Conflict-Resolution Strategy on Monotonic RBSs	103
19.	The Effect of Conflict-Resolution Strategy on Non-Monotonic RBSs	103

List of Tables

Table | Page

1. The Effect of Fact Ordering on Monotonic RBSs (Part 1) 86

2. The Effect of Fact Ordering on Non-Monotonic RBSs (Part 1) 88

3. The Effect of Fact Ordering on Monotonic RBSs (Part 2) 88

4. The Effect of Fact Ordering on Non-Monotonic RBSs (Part 2) 90

5. The Effect of Rule-Ordering on Monotonic RBSs . . 93

6. The Effect of Controlled-Production Filter on Monotonic RBSs 95

7. The Effect of Controlled-Production Filter on Non-Monotonic RBSs 97

8. The Effect of Context-Restricted Filter on Monotonic RBSs 97

9. The Effect of Context-Restricted Filter on Non-Monotonic RBSs 99

10. The Effect of Conflict-Resolution Strategy on Monotonic RBSs 101

11. The Effect of Conflict-Resolution Strategy on Non-Monotonic RBSs 102

AFIT/GOR/ENS/88D-12

Abstract

The purpose of this ~~study~~ thesis is to develop a methodology for evaluating the performance of rule-based systems (RBSs) using a simulation approach. A numerical scheme is used for knowledge representation; facts are represented by integer numbers and the rules and data memories are represented by matrices. The numeric representation can be handled by simplified algorithms that simulate the function of different types of inference engines. Six types of forward-chaining inference engines that vary according to the conflict-resolution strategy and the implementation of filters are simulated and compared. The number of match-tests of the left-hand side of the rules against the data memory is used as a measure of performance to estimate the relative matching effort for each inference engine. Also, a methodology to reduce the matching effort of a RBS by changing the order of the facts in the left-hand side or changing the order of the rules is described.

To simulate RBSs, it is assumed that probabilistic relations among the assertions to a RBS can be identified and specified by the experts or after running the system for some time. The numeric representation and the probabilistic relations provide the environment needed to build a simulation model. A rule-generator program is developed to randomly

generate RBSs with different specifications. RBSs that vary in size (the number of rules), shape (the number of facts in both sides of the rule), and monotonicity (monotonic or non-monotonic) are generated and used in experimentation.

Two types of experiments are performed on the generated RBSs. The first type estimates the reduction ratio in matching effort achieved by rearranging the facts or the rules in a RBS. The second type estimates the reduction ratio in matching effort achieved by implementing two types of filters or changing the conflict-resolution strategies.

SIMULATING RULE-BASED SYSTEMS

I. Introduction

Rule-Based systems (RBSs) are the most used means for building expert systems. Experts tend to express their knowledge for solving problems in terms of conditional rules. In RBSs knowledge is represented as a set of facts and rules. Solutions are inferred by interacting with the users and searching through the knowledge.

RBSs consist of a knowledge base and an inference engine. Different methods and strategies are used for building the knowledge base and designing the inference engine. Some of these methods and strategies perform well when used for certain applications; others are used for building general tools that can be used in different applications. The performance of RBSs can be described in terms of speed or accuracy. Speed refers to the time needed to draw the results, while accuracy refers to the correctness of these results. In expert systems, a subjective aspect of performance may also be considered, viz., the naturalness of the output. Speed can be improved by implementing more efficient algorithms. Accuracy and naturalness do not depend only on the rule set of the RBS, but are also affected by the control strategies of the inference engine.

The speed of an algorithm is usually evaluated by applying the algorithm to different RBSs and comparing the execution time with other algorithms. Accuracy of a specific

RBS is usually evaluated by testing different cases with known results and comparing the output of the RBS with those results. Different methods have been developed to study and evaluate the performance of RBSs.

RBSs are not limited to a small set that can be examined in one study. A study related to the performance of RBSs usually examines an application, an architecture, or a subset of RBSs. When a class of RBSs is investigated, knowledge representation is usually abstracted in order to provide a general representation of the class under investigation. Abstraction leads to simpler representation to the knowledge, and in turn to simplified versions of the inference engines.

This thesis presents a study in the area of the performance of RBSs. The method used in this study belongs to the methods that abstract knowledge representation. As in other studies, this study abstracts knowledge representation in order to investigate a class of RBSs; however the approach is different. In this study, knowledge is represented by numbers and the relations among the facts are described by probability distributions. This suggests that a simulation approach can be used to investigate RBSs and evaluate their performance.

Simulation is a powerful and flexible modeling technique that is widely used to study different systems. Simulation is best used to compare the relative performance of the systems. Simulating RBSs can lead to better understanding of their behaviour and performance under different circumstances.

Problem

The purpose of this study is to show how a simulation approach can be used to design a model that simulates the behaviour of RBSs and to provide an environment to experiment with the model to measure some aspects of the relative performance of RBSs.

The study includes the following steps:

1. Examine the different methods and strategies for building RBSs.

2. Design a numerical scheme for knowledge representation.

3. Define suitable measures for the performance of RBSs.

4. Design a methodology to generate a wide variety of RBSs.

5. Write modules to simulate the function of selected architectures of inference engines.

6. Design experiments to measure the relative performance of RBSs.

Scope

This study is limited to forward-chaining RBSs; backward-chaining RBSs are not attempted. A low level of resolution is used for knowledge representation; facts are composed of one term only, and they should not contain pattern-variables. The left-hand side of the rules can include AND and NOR Boolean operators, but the OR operator is not allowed. This representation is not very restrictive since most knowledge

representation schemes can be converted into this degree of resolution by adding extra rules or adding extra facts in the left-hand side of the rules. Both monotonic and non-monotonic RBSs can be used; the right-hand sides of the rules can contain both adding and deleting actions. Modifying actions and actions that interact with external procedures are not considered. However, the functions of modifying actions can be simulated. Certainty factors or any other uncertainty schemes are not supported by this representation.

No specific restrictions are imposed by this representation for simulating different architectures of inference engines. However, an architecture should be simplified to match the degree of abstraction used for knowledge representation. In addition, any type of application can be handled if it can be suitably represented by the available features.

Assumptions

The following assumptions are made in this study:

1. The RBSs investigated can be represented directly by the features presented in the model, or they can be converted into a suitable form.

2. Probabilistic relations among the facts, used as initial assertions to a RBS, can be defined and specified by the experts initially, or developed after running the system for some time.

3. Matching overhead can be expressed by the number of matching tests, for the conditions in the left-hand side of the rules, against the data memory.

Approach

Features presented. The model is designed to provide a general representation of a wide class of RBSs. The features represented in the model are the basic features found in most forward-chaining RBSs. These features are enough to represent the bahaviour of a small class of RBSs, one term per fact without variables, and at the same time they can approximately represent the behaviour of a wider class.

Model Structure. The purpose of using numeric representation in the knowledge base is to provide a concise representation that can be handled easily and fast. The numeric representation provides a suitable environment to use simple data structures and direct access, which accelerate the inference process. The FORTRAN-77 language is used in this study because of its ability to handle the numeric representation simply and easily, provide a simulation environment, and interact with statistical packages.

Generating RBSs. To investigate a large number of RBSs with different specifications, the model should be provided with a capability to generate RBSs. A rule generator is developed in this study to provide such capability.

Relative performance. Matching constitutes a great part of execution time of RBSs. The selected measure of performance, the number of matching tests, is used to compare

the matching overhead for different RBSs in a way independent of the environment, hardware or software, running the system. Experiments are designed to compare the relative matching overhead for different structures of the same RBS, or different implementations of inference engines.

Organization

The thesis is organized in seven chapters and one appendix. In Chapter II, the components of different architectures of RBSs are described. In Chapter III, a review of some of the work related to the performance of RBSs is presented. The model structure and the rule generator are described in detail in Chapter IV, and the algorithms used to simulate the functions of the selected inference engines are described in Chapter V. In Chapter VI, the experiments used in this study are explained and the results of these experiments are summarized. In the last chapter, a summary of the study and recommendations for future work are presented. The Appendix contains listings of the computer programs and samples of the input and output files of some of these programs.

II. Rule-Based Systems

Rule-based systems (RBSs) are one of many Artificial Intelligence (AI) techniques used for problem solving. Researchers in the AI area have developed different techniques for problem solving based on knowledge representation and manipulation in computer programs. RBSs, logic programming, object-oriented programming, and hybrid-language programming are among the techniques that are widely used for problem solving. RBSs are characterized by using data-sensitive unordered rules rather than sequenced instructions as the basic unit of computation (Brownston and others, 1985: 4). They are sometimes referred to as Production Systems. In this study, the two terms are used as synonyms, though the term rule-based systems has slightly broader definition.

Expert systems constitute a widely used application for RBSs. The term expert systems is used in AI to refer to computer programs that behave like an expert in some, usually narrow, domain of application; an expert system should be capable of explaining its decisions and the underlying reasoning (Bratko, 1987: 314). Experts tend to express their knowledge in terms of a set of situation-action rules, and this suggests that RBSs should be the method of choice for building knowledge-intensive expert systems (Hayes-Roth, 1985: 921).

RBSs are sometimes defined as modularized know-how systems, where know-how is practical problem-solving knowledge. The kinds of information that constitute such

knowledge are defined in (Hayes-Roth, 1985: 921) as follows:

1. Specific inferences that follow from specific observations;
2. Abstraction, generalizations, and categorizations of given data;
3. Necessary and sufficient conditions for achieving some goal;
4. Likeliest places to look for relevant information;
5. Preferred strategies for eliminating uncertainty or minimizing other risks;
6. Likely consequences of hypothetical situations;
7. Probable causes of symptoms.

RBSs have been used successfully to solve problems in different domains. Two of the most famous systems are MYCI for medical diagnosis, and XCON or R1 for the automatic configuration of computers. Applications of RBSs include a variety of problem domains such as classification, diagnosis, monitoring, design, and planning.

Basic Architecture of Production Systems

The production system architecture typically includes three major components:

1. Data memory (working memory)

A data store serves as a global database that contains whatever information is relevant to the specific problem (facts-goals).

2. Rule memory (production memory)

A store for the set of rules (productions) that constitutes the program. Each rule has a left-side condition part which determines the applicability of the rule, and a right-side action part which describes the action to be performed by the rule (Rich, 1983: 31). Rules are usually written in the form:

IF condition THEN action

3. Inference engine (control)

A finite-state machine which executes (fires) rules. It determines which rules are relevant to the current data memory contents and chooses one to apply (Brownston and others, 1985: 7). The inference engine has a cycle consisting of three action states:

1. Match rules: find all of the rules that are satisfied by the current contents of data memory. These rules are the potential candidates for execution and are referred to as the "conflict set".

2. Select rules: apply some selection strategy (conflict resolution) to determine which rules will actually be executed.

3. Executing rules: execute the action part of the selected rules.

Rule-execution (firing) usually changes the contents of data memory. A different set of rules will match in the next cycle. The cycles continue until a stopping condition is satisfied. This control mechanism is referred to as the "recognize-act" cycle. Figure 1 shows the basic architecture of the inference engine.

The strategies that are applied by the inference engine are known as "control strategies". Control strategies are responsible of searching for a solution in the problem's solution space. Two requirements are needed for good strategy: to cause motion in order to lead to a solution, and to be systematic; the latter requirement corresponds to the

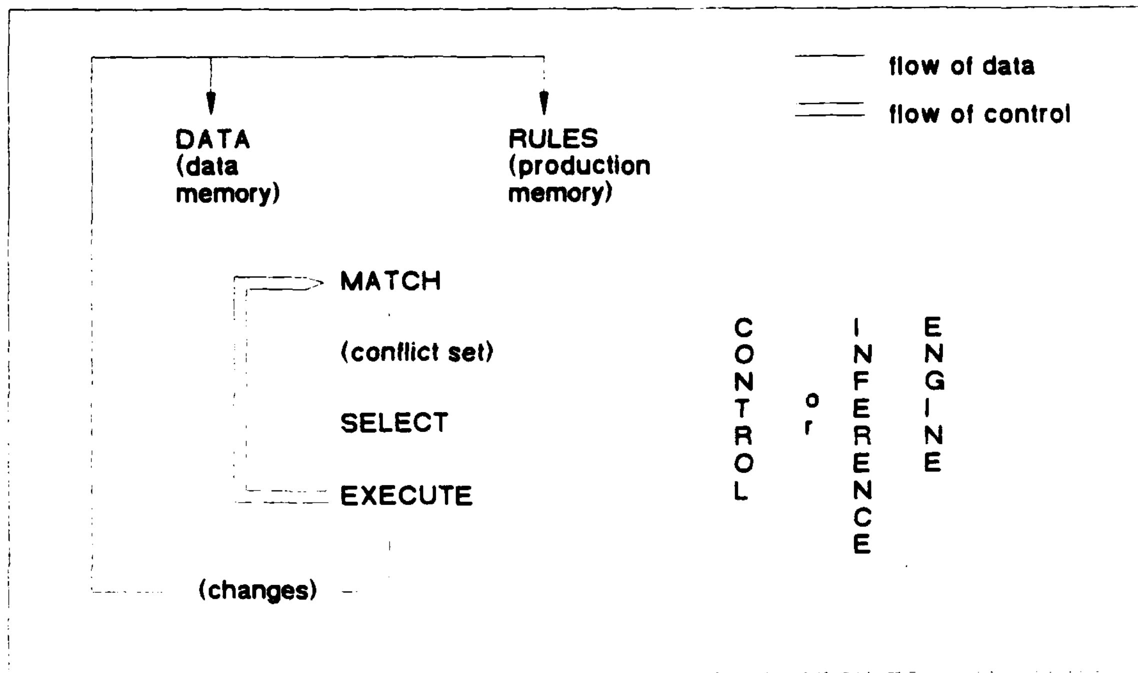


Figure 1. Architecture of a Production System Model.

(Brownston and others, 1985: 6)

need for motion (Rich, 1983: 33-34). Search techniques such as depth-first or breadth-first are systematic strategies. Search is applied to the rules in a production system in one or both of the two directions: forward or backward. The direction used corresponds to the type of reasoning strategy used by the inference engine. According to Rich (Rich, 1983: 56-57) forward and backward reasoning can be viewed as systematic process which can be described as follows:

Forward reasoning (chaining) starts from the initial configuration and begins building a tree of move-sequences that might be solutions. It generates the next level of the tree by finding all the rules whose left sides match the contents of the data memory (the initial configuration), and using their right sides to create a new configuration. The process continues until a configuration that meets a stopping criterion is generated.

Backward reasoning (chaining) starts from the goal configuration and begins building a tree of move sequences that might be solutions. It generates the next level of the tree by finding all the rules whose right side match the goal configuration, and using the left sides of the rules to generates the nodes of the next level of the tree (subgoals). The process continues until a node matches the goal configuration.

The next three sections show a more detailed description of the components of production systems. The discussion in these sections is based mainly on (Brownston and others, 1985: Ch 7).

Data Memory

Two types of information can be stored in data memory: facts and goals. Facts are used by the rules to make inference, and goals are the final conclusions toward which the problem solving aims. Facts and goals are usually stored in the same memory; however, in some rule-based languages they are stored separately.

Organization. Rule-based languages use different methods to represent data elements. One method is attribute-value pairs that contain knowledge about an object. A similar method is object-attribute-value triples in which the object name is repeated with each attribute. The general LISP list structure is also used by some languages; this representation places little restriction on the form in which knowledge is represented. A different method, used in object-oriented languages, is the frame or schema representation. This method, which allows inheritance of attributes, value types, and values, can greatly facilitate programming without affecting the reasoning style. Semantic networks are also used as a method of data representation. In this scheme, objects are represented by nodes and attributes are represented by arcs.

Goals are also organized in a variety of structures that facilitate the implementation of the specified control strategy. A tree structure often is used when the system seeks its solution through a strategy of divide and conquer, problem refinement, or problem decomposition. The stack

structure is usually used to implement a pure backward-chaining control strategy. When all goals must be processed and they are of equal importance, the queue structure is used. In the case of specifying dynamic priorities to the goals, the agenda structure is used.

Properties. Properties of data elements are used to determine the elements most relevant to the current task; these are examined and processed before the other elements. Three properties of the facts are used for this purpose: recency, certainty, and activation. A recency number is attached to a fact when it is added to (or sometimes deleted from) the data memory. In other implementations, the same recency number is attached to all facts added in the same cycle. In systems that use some kind of probabilistic reasoning, a certainty measure is attached to the facts when added to the working memory. In other systems a measure that relates a data element to other data elements is propagated according to criteria determined either by the architecture or by rules in the system. This measure is called activation and is usually used in cognitive systems.

Goals also have properties that can be updated by the rules, inference engine, or both. Properties that are usually assigned by the rules include: priority, and the expected time and memory-space requirements. Properties that are best assigned by the inference engine include: the method that describes how the goal can be achieved, the cumulative time and space used in accomplishing the goal so far, and the

recency of the goals. Properties that can be specified either way include: the list of goals and subgoals, the set of preconditions and postconditions related to the goal, and the status of the goal (active, achieved , ...).

Rule Memory

At any given time the rule memory may or may not contain all rules that constitute a RBS. In some systems, all rules are continuously active and sensitive to the contents of data memory. In other systems, rules can be grouped into sets that are loaded or removed from rule memory as a group.

The Left-Hand Side (LHS). Forward-chaining systems allow a wide variety of features in the LHS part of rules. This gives the programmer freedom, but complicates the implementation. In backward-chaining, fairly simple features are used for the conditions because they may be established as subgoals. For both types of reasoning, the LHS features can be described according to the following:

1. Types of tests against the data memory:

a. Positive tests: conditions that must be true (present) for the rule to succeed.

b. Negative tests: conditions that must be false (absent) for the rule to succeed.

c. Condition function-calls: a call to a function that serves as a condition element and returns success or failure based on its inputs.

d. Disjunctions: a set of elements, only one of which needs to be in data memory for the rule to succeed.

e. Partial tests: a minimum number of condition elements that must be satisfied for the rule to succeed.

2. Conditions in the range of attribute values:

a. Relational operators: single relations, disjunctions, or conjunctions.

b. Attribute function call: arbitrary function calls to test values.

c. Negative matching against values.

d. Matching components of a list that must be in the attribute list.

e. Partial attribute matching: values must be within some predetermined threshold.

3. Pattern matching with variables

a. Regular variable: matches a single value or a sequence of values to satisfy the value of a single attribute. The value is usually either a scalar or a list.

b. Segment variable: matches a sequence or more than one sequence within a data element to specify the value of a single attribute.

c. Nested pattern: a recursive data structure that matches patterns within patterns.

When a match occurs the system stores the variables and the values that it matches as a binding. Typed variables are rarely used in production system languages.

The Right-Hand Side (RHS). In backward-chaining systems the RHS part has a limited function. It either specifies a conclusion to be drawn or adds data to the data memory. In

forward-chaining systems the RHS part consists of a (usually ordered) set of actions to perform. The types of actions for forward-chaining systems are the following:

1. Changes to data memory: addition, deletion, or modification.

2. Input and output: usually dependent on the operating system implementation language. Some languages that are written in LISP allow the user access to all internal LISP functions.

3. Changes to rule memory: add, remove, or modify rules during execution.

4. Calling user-defined functions: implementation-language functions that are called with the arguments provided by the rules.

Control Strategies

Implementation of control strategies is either totally or partially separated from the rule set. For systems involving total separation, the language provides built-in controls so that the user can concentrate on domain knowledge only. This implementation is faster, easier, and more reliable. However, it is not flexible and may be inefficient when applied to different domains. Systems involving partial separation are more flexible, but adds more responsibility to the user, which means more time and complexity. Three types of control strategies are discussed in this section: conflict resolution, filtering, and metarules.

Conflict Resolution. Conflict-resolution strategies affect all aspects of the performance of RBSs: speed, accuracy and naturalness. In (McDermott and Forgy, 1978: 181-183) the authors defined "sensitivity" and "stability" as two performance requirements that conflict-resolution strategies should meet. Sensitivity is the quickness of response to the dynamically changing demand of the environment, while stability is the system's continuity of behaviour. Some of the principles of conflict-resolution strategies are described below.

Refraction. Refraction requires that rules fire not more than once on the same data. This principle is necessary to prevent looping. Refraction can be implemented by forbidding the firing of identical instantiations on consecutive cycles, or by keeping track of all rule-instantiations that have fired so far.

Data Ordering. Data can be ordered by recency or activation, in order to determine the more sensitive data. Recency of facts or goals can be measured in different ways. Also there is more than one way to select the rule to fire on the basis of recency.

Specificity Order. Specificity favors rules that are special cases of other rules or are more specific according to some measure.

Rule Ordering. Rule ordering is the static ordering of the rule set independent of the way rules are instantiated. The ordering may be specified by the user or may be computed

using some features of the rules.

Arbitrary Choice and Parallel Selection. None of the principles described guarantees that only a single instantiation will remain in the conflict set. An arbitrary choice can be applied to select one instantiation. Some languages use a parallelism principle which fires all the instantiations in one cycle. Parallelism has the disadvantage of the possibility of adding conflicting information.

Some of selection rules that use the principles described in this section are described and evaluated in (McDermott and Forgy, 1978: 184-190). The authors defined three knowledge sources that could be used by any rule: data memory, production memory, and a state memory maintained by the inference engine. A brief discussion of selection-rules is given below:

1. Production order rules, POs:

a. PO1: the relation of dominance totally orders the productions.

b. PO2: productions are divided into groups, each performing a separate task. No relation of dominance is specified among productions related to different tasks. Both methods use production memory as the source of knowledge. However, POe is more selective than PO2.

2. Special case rules, SCs:

a. SC1: uses production memory as its knowledge source. It is sensitive to a special case relationship between the production of instantiations.

b. SC2: uses working memory as its source of knowledge. It is sensitive to a special case relationship between the data of instantiations.

c. SC3: uses both production memory and working memory as its knowledge source. It is similar to SC2, but it takes negated conditions into account.

SCs are only weakly selective; SC2 is the most selective and SC3 is the least.

3. Recency rules, Rs:

Two different measures for the age of a data element are used: the number of cycles that have elapsed since its assertion, or the number of other actions that have been performed since the action that asserted that element. All recency rules use working memory as their knowledge source.

a. R1: measures time in number of actions. It orders instantiations on the basis of the most recent data element contained in each. This rule is highly selective.

b. R2: similar to R1, but it measures time in number of cycles. It is less selective than R1.

c. R3: measures time in number of actions. Among the instantiations, it prefers the one whose least recent element is most recent. It is somewhat more selective than R1.

d. R4: similar to R3, but it measures time in number of cycles. It is less selective than R3.

e. R5: it measures time in number of actions. Unlike the other recency rules, it considers the recency of

all data elements of an instantiation. To order two instantiations, it first compares their most recent elements; if those elements are equally recent, it compares the next most recent elements, and so on. This rule is the most strongly selective of the recency rules.

4. Distinctiveness rules, Ds:

Distinctiveness rules apply the refraction principle discussed before. They use state memory as their knowledge source.

a. D1: tries to prevent any production from firing on consecutive cycles by looking at the most recent firing.

b. D2: looks to the entire history of the system to prevent instantiations from firing twice.

Both D1 and D2 are weakly selective rules.

5. Arbitrary decision rules, AD1:

AD1 selects one instantiation at random.

Filtering. Filtering is a technique used to reduce the number of rules and the number of data elements that participate in the matching process. If no filtering is applied, the entire set of rules will be tested against all data elements in data memory on each recognize-act cycle. Filters must be able to store the information they capture in such a way that accessing and updating is significantly less costly than processing the excluded rules or data elements (McDermott and others,1978: 158). Two types of filtering can be applied: rule filtering and data filtering.

Rule filtering. Three methods that help identify a subset of the rules to match are discussed below: controlled

production, goal restricted, and context restricted. In controlled production, the user writes a program that specifies which subset of the rules to try in the following cycle. In goal filtering, rules are organized into subsystems by the type of goal they help to solve. Context-restricted systems use the information in data memory to group rules into subsystems by their similarity, time of proper application, location in the consultation (goal) tree, or some other feature.

Data Filtering. Two methods that help identify a subset of data elements to be used by the matching algorithm are discussed below: activation filtering and certainty filtering. In the first method, the user specifies an activation threshold, and items below that threshold are not matched. In the second method, the user specifies a minimum certainty, and items that are less certain are not matched.

Metarules. Metarules are rules that determine how to apply other rules. Metarules can be written in the same language as the rules themselves, or in a separate control language. Some of the conflict-resolution principles can be coded quite easily as metarules; examples are recency, specificity, or refraction. Some properties of goals can also be used by metarules. Metarules should not be mixed with normal rules unless they are controlled strictly.

III. Efficiency of Rule-Based Systems

The performance of Rule-Based Systems (RBSs) can be viewed from two sides. The first is the amount of time required to draw the conclusions, and the second is how accurate or reliable these conclusions are. The two sides differ considerably; however, improvement of the reliability of a certain RBS may affect its execution time. In this chapter a review of some of the work in the efficiency of RBSs is presented. The first section presents work in the area of implementation which aims to reduce the execution time, the second section presents a review of an example of how to improve a system's reliability, and the last section presents a combination of the two sides given in one study.

Efficiency of Implementation

The efficiency of RBSs can be improved by making changes in the implementation in two ways: by designing more efficient inference engines, and by writing rule sets in a more efficient way. Reducing the matching effort and limiting the size of the conflict set are among the approaches that can be considered by the RBS designer to improve the performance of the inference engine. The order of rules in the rule set and the number and the order of conditions in the LHS of rules are among the factors that can be handled by the user to improve the performance of a certain RBS.

Five papers are discussed on this section. The first paper presents an implementation of some filtering techniques. Filters were implemented for an actual architecture of a RBS.

The rules investigated in this paper have high LHS resolution, i.e., the LHS of such rules contain more than one term and pattern-variable. However, the architecture of the presented RBS does not include disjunctions in the LHS of the rules. The second paper extends one of the ideas presented in the first paper by using an architecture that allows disjunctions. However, it only uses a low level of resolution for the LHS of rules. The low resolution provides a simpler, more general, but less accurate representation. The third paper also uses low resolution, but in a limited architecture that does not represent a wide variation of RBSs. The fourth paper represents an efficient, but complex matching algorithm. The algorithm exploits the components of data elements in detail and could not be easily abstracted. The final paper in this section presents recommendations to the programmer of a general purpose rule-based language in order to improve processing time by making changes to the rules' structure.

The Effect of Filtering on Efficiency (McDermott and others, 1978: 155-176). Production-system architectures that match all rules in the rule-set against all elements of working memory are inefficient in terms of execution time. Such production systems do not make use of knowledge that could be obtained before or during execution. Three knowledge sources (KSs) can be used to eliminate this inefficiency:

1. The condition-membership KS provides knowledge about the occurrence of condition elements in productions.

2. The memory-support KS provides knowledge about the memory elements that support condition elements.

3. The condition-relationship KS provides knowledge about the relationship among condition elements within each production.

The first KS permits rejection of productions whose condition elements are not supported by elements in working memory. Filters using this KS allows further testing of productions whose condition elements all appear to be supported by memory elements. This does not guarantee that such productions will be satisfied because filters may use only partial knowledge, or variables may be instantiated to different values. The condition-membership KS is static and it is usually established before execution.

The second KS contains the knowledge of which condition-elements are supported by a memory element at the beginning of a cycle. This KS complements the first one. If a filter makes use of both of them, then no matching would be necessary at the beginning of the cycle. However, the knowledge must be updated at the end of each cycle. The ratio of the number of elements added or deleted per cycle to the number of elements in working memory determines how the updating cost compares with the cost of matching.

The third KS can be used to determine whether a production, each of whose condition elements is supported when considered individually, is actually satisfied. It can also be used to determine whether a production is satisfied even though some of its condition elements seem to be unsupported when considered individually. Condition-relationship KS may be used only when the first two KSs are also in use.

Using a production system architecture called PSG, the authors designed two filters. The first one uses condition-membership KS, and the second one uses both the condition-membership and the memory-support KSs. In the first filter, the KS is represented by a discrimination net that indicates which productions contain the same first-named symbol (primary feature) as a way to define similar productions. The net is traversed at the beginning of each cycle to determine the possibly satisfied productions that constitute the conflict set. Three versions of the second filter have been implemented. They differ from one another only in the set of features they use to represent elements. The first implementation uses only the primary feature to represent a condition element. At the beginning of a run three sets of associations are established as follows:

1. Each primary feature is associated with a list of those condition elements it represents.

2. Each condition element is associated with a list of productions containing it.

3. Each production is associated with the number of condition elements contained in it (support measure).

Through this sequence of associations, the support measure is updated whenever a primary feature enters the working memory (decremented by one), or leaves the working memory (incremented by one). If the resulting number is less than or equal to zero, the production is tagged as possibly satisfied. Before each cycle the satisfied productions are put into the conflict set. The other two implementations use

both a primary feature and a list of secondary features to represent condition elements but in two different ways.

The authors used arithmetic formulas to predict the execution cost for the original architecture of PSG and the two modified architectures which augment one of the two filters. They tested the formulas by comparing the actual cost with the predicted cost for different cases that include variation in working memory size, variation in production memory size, and varied production systems with constant working memory size and production memory size. For each case, tests were conducted for the two conflict resolution strategies used in PSG: rule order and recency. They concluded that in spite of inadequacies in the statistical assumptions, the cost formulas do capture the most important dependencies. They added that a large increase in efficiency can be gained by using such simple filtering mechanism.

A Focus of Attention Technique (Whiting and others, 1985: 160-161). A focus of attention technique (SUBEX) has been designed for improving the efficiency of production systems. The basic idea of SUBEX is to build a data structure that allows the inference engine to take advantage of the fact that rule conditions are satisfied incrementally and that preserving the history of past actions can prevent repeated evaluation of the same rule conditions. To evaluate the SUBEX technique, three related implementations of rule-based pattern directed inference systems were studied. The first is a system in which all rules are evaluated against the entire working memory (context base) on each cycle. The second is an

implementation of the condition-membership filter of McDermott and others (McDermott and others, 1978: 160-161). The third implements the subexpression focus technique (SUBEX).

The authors also developed an automated rule base generator which generates arbitrary rule bases with different size, topology, and frequency of appearance of rule types. In addition, an analyzer program was written to examine each generated rule base and report on its characteristics. Finally, a driver program was developed which generates test data sets in a random fashion, runs all three inference engines against the same test data sets and rule set, and reports and summarizes the results.

The software was implemented in Common LISP using a VAX-11/780 computer running the VMS operating system. Differences among the three implementations were kept to the minimum feasible, to facilitate direct comparison using relative CPU execution time. A most-recent-first conflict resolution scheme was used in all inference engines. The condition (antecedent) part of the rules is composed of literals or arbitrary Boolean compositions of literals, while the action part (consequent) is a single positive literal. At the beginning of a test, the rules are read in one at a time from a text file and compiled into tree structures which are accessed by the inference engines.

The SUBEX inference engine builds an index of "reference counts" associated with every Boolean connective in the antecedent part of each rule. The count field of the reference count node represents the number of facts required

to satisfy the Boolean operator which occurs at the top (outermost) level of the antecedent of the rule. This count equals 1 for an OR, and equals N for N-ary AND. The process of constructing reference count nodes continues in a similar fashion for any subexpression of the antecedent, for any subexpression of those subexpressions, and so on. As a fact is inserted into the context base, each reference count node associated with that fact is located and decremented by 1. Any reference count which goes to zero as a result indicates satisfaction of the antecedent component represented by the node. If all nodes that represent a rule go to zero the rule is enabled for firing and is passed on to the conflict-resolution routine.

A series of experiments was performed to compare the relative efficiency of the three inference engines using different rule sizes (50, 200, 800 rules), different shapes (fan-in, fan-out, no-fan), and different antecedent compositions (all ANDs, all ORs, 50% ANDs and 50% ORs, nested ANDs and ORs). The results showed that the SUBE inference method displays a substantial speed advantage over both the other two methods. This advantage is maintained over a wide variation in rule base size and structure, but appears to be most pronounced with large rule bases.

Forward Chaining Versus a Graph Approach (Neapolitan, 1986: 62-69). Production systems in which the set of rules does not contain variable data are used in many expert systems. For such production systems a more efficient

architecture can be implemented by storing the rules in a graph and the true assertions in an assertion list. The assertion list is traversed only once; at each assertion a premise (condition) is triggered in all the rules which have that assertion as a premise. When all premises of a rule trigger, the rule's conclusion is added to the end of the list of assertions. It must be added at the end so that it will eventually be used to make further deductions. Each rule is represented by a special count node with premises coming in and conclusions leaving.

In the regular implementation of forward chaining, the computing time (in the worst case) is a function of N, where N is the number of rules. The actual time is totally unpredictable because it depends on the rule mix and the true assertions. Computing time in the graph method is related only to a linear function of N; this suggests that it should be much more efficient.

Two algorithms were written in PASCAL and implemented in a VAX running under VMS to compare the performance of the two methods. Computing time was measured for a variety of RBSs that vary in the number of rules, maximum number of premises in the rule, and the number of true assertions. It was found that computing time for the regular implementation is usually about 10 times that of the graph method.

The Rete Matching Algorithm (Forgy, Shepard, 1987: 34-40).

One property of production systems is that the number of elements of working memory changed in one cycle is very small compared to the total number of elements in working memory.

Another property is that the condition parts of rules contain many similar or identical patterns. Large production systems usually have from hundreds to more than one thousand elements in working memory at any one time. On a typical cycle, changes made by rule-firing are from 2 to 4, which represents a very small percentage of working memory contents. Since working memory changes little on a typical cycle, much of the information used by the match process in one cycle is still present and can be reused in the next cycle. The basic idea of Rete is to save such information from cycle to cycle, and to update it as necessary to reflect the changes made to the working memory in each cycle. Thus the cost of matching depends primarily on the rate of change to working memory rather than its absolute size. To make use of similarities of patterns in rules, Rete processes the patterns before the system is interpreted, to locate common terms and eliminate as many of them as possible. These patterns are compiled in a special data-flow graph called the Rete network, a tree structured sorting network or index for the productions. The graph serves as a function to map changes in working memory into changes in the set of satisfied productions.

The Rete algorithm has been implemented in several languages in both research-grade and commercial systems. Evidence shows that Rete is the most efficient algorithm developed so far for performing match operations on single processor. Figure 2 shows a summary of the implementation of the Rete algorithm in the OPS5 production-system language.

Efficient Programming in OPS5 (Brownston and others, 1985: Ch 6). OPS5 is a general purpose production-system language. The inference engine of OPS5 is a forward-chaining one, but backward-chaining problem-solving strategies can also be implemented. OPS5 imposes no constraints on the type of application program that can be written. As a result, the programmer must explicitly handle both data representation and flow of control by means of rules. OPS5 uses attribute-value pairs for data representation. Objects are defined by a structure called "element class" which contains the object name and its attributes names. Pattern-variables are allowed in the LHS of rules. Data elements can be added, deleted, or modified as a result of executing the RHS of rules. Programmers can use the BUILD command in the RHS of a rule to add another rule during execution. OPS5 uses the Rete algorithm for the matching process.

Programmers of OPS5 can sometimes make dramatic improvements in the efficiency of a set of rules. Some simple changes such as ordering conditions within a rule, or adding or regrouping attributes within element classes may noticeably reduce the execution time of a production system run. Some possible causes of slowness in the set of rules in decreasing order of effect are specified as follows:

- Changes in working memory are mapped directly into changes in the conflict set.
- Rather than repeatedly testing all conditions in all rules on each recognize-act cycle, the Rete algorithm saves match information in a network.
- Additions to and deletions from working memory cause changes in the network data structures.
- The saved information and the new memory changes determine whether or not there are changes in the set of rules that match.
- When a working memory element is checked against a condition element, the attribute tests of the condition are evaluated in the order in which they are written. If the element fails one of the tests, the remaining tests are not evaluated.
- The conditions of a rule are checked in order from first (top) to last (bottom). If no combination of working memory elements matches an initial sequence of conditions, the remaining conditions are not considered.
- Stored with each condition element are two data structures: a right memory containing the set of working memory elements that the condition matches, and a left memory containing the combinations of working memory elements (and, implicitly, variable bindings) that make it and all preceding conditions match consistently.
- The consistency of matches for a condition element is computed from its right memory and the left memory of the previous condition. Thus, whenever a working memory element is added or removed, all conditions that newly match or no longer match are affected, and all memories of consistent bindings associated with conditions following the affected ones may also change. In other words, if something changes early in a left-hand side, it affects everything that follows it in the left-hand side.
- Identical condition elements and sequences of condition elements in different rules are factored out in the network and matched only once.
- Deleting a working memory element has an additional cost when many other working memory elements match the same conditions that it does. It takes a linear search to find the deleted element in the lists of matching elements for a condition, and the cost propagates to removing the combined matches from later conditions.

Figure 2. A Summary of the Rete Matcher (Brownston and others, 1985: 229)

1. When large numbers of working memory elements match successive conditions in the same rule, finding consistent matches is one of the most expensive processes.
2. Frequent changes to elements that match conditions occurring early in rules are more expensive than changes to those that match later-occurring conditions.
3. When a condition matches a large number of elements, deletion from the set is expensive.

Figure 3 shows a summary of some techniques for speeding up processing. Almost all efficiency techniques involve modifying a rule to exploit a constraint as quickly as possible so that complex processing is performed only when necessary. The principles for improving efficiency are sometimes contradictory. If the principles conflict, estimates of the costs of alternatives may be used to choose the most effective one.

Efficiency of Consultation (Politokis, 1985: 8-30, 36-39)

RBSs are the most common knowledge representation scheme that are used in expert systems. Some AI researchers have developed methods to evaluate expert systems by comparing the human expert's conclusions with the expert system's conclusions. The evaluation process usually uses "cases" with known conclusions to evaluate the expert systems. Evaluation some expert systems such as CASNET, PROSPECTOR, MYCIN, and EXPERT has been performed using the cases approach. Some special programs have been written as an aid in the evaluation process: the TEIRESIAS program for MYCIN, and the SEEK program for EXPERT are among those special programs. In this section a brief review of the SEEK program is presented.

1. Avoid conditions that match many working memory elements.
   a) Add attribute tests that change the condition so that fewer working memory elements match it. Tests may be based on restrictions to data or on the state of processing.
   b) Add new element classes or modify the representation to change the data elements so that fewer of them match the condition.
   c) Represent and enumerate sequences carefully.
2. Avoid big cross-products between conditions.
   a) Order the conditions so the more restrictive ones occur first. This limits the number of consistent matches that are passed on to the next condition.
   b) Write rules with a few big conditions rather than many simple conditions by merging the attributes of related element classes into fewer element classes.
   c) Use **build** to specialize rules.
3. Avoid frequent changes to matched conditions.
   a) Put conditions that match frequently changing elements as far toward the end of the rule as possible.
   b) Avoid excessive changes in control elements.
4. Make matching individual condition elements faster.
   a) Put the most restrictive attribute tests first to speed the match of working memory elements against conditions.
   b) Change the representation of data to speed up matching.
5. Limit the size of the conflict set.
6. Call user-defined functions.

Figure 3. Summary of Techniques for Improving Efficiency in OPS5

(Brownston and others, 1985: 241)

SEEK combines design aids for building expert models with empirical testing and evaluation heuristics; these help in carrying out experiments for the purpose of improving decision-making knowledge. SEEK uses a restricted type of productions written in a tabular form to express the rules. It uses experience, in the form of stored cases with known conclusions, to interactively guide the user in refining the rules of the model. A typical interaction with SEEK involves iterating through three steps: obtain performance of the rules on the stored cases, analyze the rules, and revise the rules.

The performance of the rules is evaluated by matching the expert's conclusion with the model's conclusions in each case. SEEK allows the user to specify the proper way to score the result of the test when ties in certainty occur between the model's conclusion and the expert's conclusion. It also allows the user to determine which rules and cases are to be ignored during the evaluation process. Results can be obtained in two forms: a summary for the performance of the model as a whole, or the performance of a specific rule. Results about the performance of the model shows the number of cases in which the model's conclusions agree or disagree with the expert's conclusions. Results about the performance of a specific rule contains the following:

1. The number of cases in which the rule was satisfied.

2. The number of times the greatest certainty was obtained by the rule and it matched the expert's conclusion.

3. The number of times the greatest certainty in the conclusion was obtained by the rule and the rule did not match

the expert's conclusion.

Interactive assistance for rule refinement is provided during the analysis of the model. Analysis can be performed on the basis of a single case or on all cases. The objective of single-case analysis is to provide the user with an explanation of the model's result for that case. SEEK cites the rules which were used to reach the conclusion and those that partially satisfied (rules that agree with the expert's conclusion which are closest to being satisfied in a misdiagnosed case). In addition, SEEK allows the user to interrogate any conclusion in the model, both final and intermediate results. In all-cases analysis, SEEK performs global analysis and reports the results by numbering and listing partially satisfied rules as potential candidates for generalization, and rules used to reach the model's conclusions as potential candidates for specialization. SEEK applies some heuristics rules, that use the statistics gathered about the rules' performance, for the purpose of suggesting specific experiments about rule refinement. The output of this step is suggestions to generalize some rules by either removing a condition or increasing the confidence, or to specialize other rules by either adding a condition or decreasing the confidence. The suggestions are ordered based on maximum potential gain.

SEEK provides the user with an editing capability to change the rules according to the suggestions. The changes are logged separately from the original rules in the model. The results of the revised version of the model are collected

and analyzed as before. The user can accept or reject each suggestion, and the cycle continues until satisfactory results are obtained.

A Comparative Study of Control Strategies for Expert Systems (Aiello,1983: 1-4). AGE is a collection of tools and partial frameworks for building expert systems. A user of AGE can define production rules about a particular domain, set up a basic structure for a solution space, and then experiment with different control strategies to find one that best fits the problem. "Best fit" can be determined both by subjective and by objective measurements. Subjective measurements include an indication of how natural the knowledge represented and the output of the program seem to the expert. Objective measurements are speed and accuracy in terms of the total number of rules evaluated or comparisons with the expert's conclusions. Three different versions of the PUFF diagnostic expert system were implemented using the AGE building tool. Each version was implemented with an event-driven strategy which has a simple blackboard data structure to store the intermediate and final results. Several levels of rules (knowledge sources KSs) are used in the blackboard data structure. The first level uses the initial assertions (test measurement) as an input and produces actions (events). A user-specified selection method chooses one of those events to use as a focus to be matched against the other levels of KSs. The events drive the order in which KSs are evaluated. The second version uses the model-driven (expectation) control strategy which has the same data structure but instead of

checking all the test data it checks only a limited number of crucial data and makes initial, broad diagnostics. AGE then attempts to substantiate the initial hypothesis with a few more data items of secondary importance. If the diagnosis is still credible, a set of further model-based expectations is generated for corroborating evidence. AGE compares the expectations with input data and partial hypotheses on the blackboard. The third version is a typical goal-driven backward chaining implementation which has only one KS.

The three implementations were compared in terms of speed, accuracy and naturalness. Speed was measured by the number of rules tested, the number of rules executed, the number of input data items referenced, and the number of references to the blackboard or internal data representation in the backchaining strategy. The table below shows the average results for the three strategies far a small sample of actual cases. There was very little deviation from the average for each measurement shown in the table.

Measured Comparison of Three Control Strategies

Strategy	rules tested	rules executed	input data	internal data
event-driven	60.4	12.6	80.4	54.4
model-driven	35.5	14.5	47	48.5
goal-driven	68	14	76	128.8

The results show that the model-driven strategy does less testing and refers less often to data than the other two strategies do. The ability to focus initially on the most likely diagnosis eliminates the need to test rules for other diagnoses. Accuracy was measured as agreement with the doctor's conclusions, based on the statements PUFF is capable of producing. The event-driven and goal-driven strategies are slightly more accurate than the model-driven strategy. Given odd or marginal data, the model driven strategy may produce incomplete interpretations. The naturalness of the output was calculated by counting the number of moves required to reproduce the doctor's order. Output generated by model-driven system always had fewer findings out-of-order. The event-driven system had the next fewest out-of-order items, and the goal-driven system had the most.

IV. The Description of the Model

In Chapter III, a review of some of the work related to the performance of rule-based systems (RBSs) is presented. One approach used in some of this work is to abstract knowledge representation in order to simplify the studying of different implementations of RBSs. The model presented in this study uses a similar approach for the purpose of providing an easily-built, fast, and flexible experimental tool for evaluating RBSs. The model captures the main features of RBSs and presents a general representation that is not dependent on a specific environment.

The basic idea of the model is to represent the knowledge base (rules and facts) in numerical form, and to use prior knowledge about the system to describe probabilistic relations among the assertions. Then a simulation approach can be used to study RBSs and evaluate their performance under different circumstances.

The model is easily built because it uses simple data structures (arrays) in the FORTRAN language. The model is fast because it uses numeric representation for data elements and provides direct access to rules and facts by using their numeric identifications as subscripts. The model is flexible because it allows the use of more than one conflict-resolution strategy and filtering technique without changing the basic data structures.

This chapter describes the basic constructs of the model and the approach used to provide the simulation environment.

The chapter starts by introducing the features represented in the model, the limitation of this representation, and the measure of effectiveness used in comparing the different structures of RBSs. The following section introduces the idea of describing a probabilistic relation among the assertions. The last section contains the details of a random rule-generator procedure used to generate different sets of RBSs.

Features Represented in the Model

The current implementation of the model is restricted to forward-chaining RBSs. Compared with the architectures presented in Chapter II, the model contains the following features:

Data Memory

Organization of Data.

Data are represented by structureless elements with low levels of resolution. Object-attribute-value triples are represented by an individual data element. Attribute-value pairs and lists can be simulated by an equivalent number of data elements.

Properties of Facts.

A recency number can be attached to each fact added to the data memory.

Rule Memory

The Left Hand Side

Types of Tests.

The model represents positive and negative tests directly. Disjunctions and partial tests can be simulated by adding extra rules.

Variables.

Variables are not represented explicitly in the model. They can be simulated by adding

extra rules.

The Right Hand Side. Addition and deletion of data elements are represented in the model. Modification of data elements can be modeled as a combined deletion-addition operation. Actions that call external procedures are not represented in the model.

Control strategies

Conflict Resolution. The model does not impose any restrictions on representing conflict-resolution strategies.

Filtering. The model does not impose any restrictions on representing filtering techniques.

Metarules. The model can represent metarules if they are written in the same regular rule structure.

Limitations

The main limitation of the model is the low resolution of data representation, which does not allow explicit representation of attributes or variables. The same level of resolution was used in other works as described in Chapter III. Low resolution allows much simpler handling of the problem. It accurately represents the object-attribute-value scheme with no variables. For other schemes, results are subject to some degree of inaccuracy. Some conflict-resolution strategies and filtering techniques have to be simplified to match the level of resolution. Simulating RBSs that contain a large number of variables with a large number of possible values for the variables is not practical. However, this author believes that such a degree of

abstraction should be accepted for comparison of results. High-resolution representation is not impossible for the numeric scheme used in the model, but it will add a considerable amount of complexity. Disjunctions in the LHS of rules can be replaced by adding extra rules. Some powerful RBS languages do not use disjunctions in the rule structures. The conclusions drawn from RBSs will not be affected by the replacement, but performance measures may not be exactly the same. Adding this feature to the model is feasible and it will not cause too much complexity.

Backward-chaining inference engines can be built in different architectures that considerably differ from those of forward chaining. In general they use more complex features. A backward-chaining architecture is not attempted for the current implementation of the model. The numeric representation would not impose additional restrictions for such attempts.

Measure of Effectiveness (MOE)

Execution time is the MOE used in most related works for comparing different implementations of RBSs. The same measure can also be used in the present model for the same purpose. This author argues that although execution time is a proper measure, it may be affected by the environment. Different environments may show different statistical results; however, they may show similar general conclusions. For example, an implementation of a hypothetical filter may show 20% improvement in execution time for an inference engine written

in language L1 using computer C1 running under operating system OS1. Another implementation for the same filter may show only 10% improvement if it is written in language L1 using the same computer, C1, running under operating system OS2.

An environment-independent MOE is used in the model, viz., the number of match-tests (NMT) of the conditions in the LHS of rules against the contents of data memory. This measure could be used in experiments that measure the effect of changing some parameters within a specific implementation. In addition, it could be used to compare the matching overhead of different implementations. However, it is not suitable to be used as a measure for the total performance of a RBS.

Model Structure

The model uses numeric representation for the data elements and rules. Facts, taken in any order, are assigned integer numbers. Rules are represented by a collection of numbers that represent data elements on both LHS and RHS of the rule. The array data structure is used to represent data memory, rule memory, and conflict set.

Data Memory. Facts are organized in the NF(i,j) array, where row number is the numeric identification (ID) of the fact. Each fact (i) has two attributes:

1. NF(i,1) is the fact status; it assumes the value 0 if the fact is currently in the data memory, and -1 if not. It is initialized by the value -1 for all facts.

2. NF(i,2) is the recency of the fact; it assumes the

value of an incremental counter that is incremented by 1 whenever a fact is added to (or deleted from) the data memory. It is initialized by the value 0 for all facts.

Data memory is not represented by a separate structure; it is contained in the fact array as a dynamic subset. No search is required to check an attribute of a fact; a direct access is available by using the fact ID.

Rule Memory. Rules are organized in the NR(i,j) array, where row number is the traversing order of the rule. It is used as a primary ID of the rule to facilitate both sequential and direct access to the rule. Each rule (i) has the following information stored in the array:

1. NR(i,1) is the rule number; it is used as a unique secondary ID.

2. NR(i,2) is the number of conditions in the LHS of the rule.

3. NR(i,3) is the number of actions in the RHS of the rule.

4. NR(i,4) --> NR(i,MAXL+3) contain the fact-IDs that constitute the LHS of the rule, where MAXL is the maximum number of conditions a rule can assume in the specific RBS currently represented. A (-) sign is attached with a fact ID if it is a negative condition.

5. NR(i,MAXL+4) --> NR(i,MAXL+MAXR): contain the fact-IDs that constitute the RHS of the rule, where MAXR is the same as MAXL except for the actions. A (-) sign is attached with a fact ID if the required action is deletion.

6. NR(i,MAXL+MAXR+1) is the rule status; it assumes the

value 1 to mark the rule if it is fired, and it is initialized by 0 to all rules.

This structure allows dynamic allocation to the rule set within the array-limits specified by the program. Rule memory is the static subset of NR array that contains the LHS and RHS of the rules.

Conflict Set. Rules that are candidates to fire are collected in the conflict-set array NC(k,j), where row number is the value of an incremental counter that is incremented by one whenever a rule is added to the conflict set. Only the necessary subset of information in stored in the NC array. The subset includes data stored in both the NF array and NR array to identify the LHS of the rule, in addition to the rule primary ID which is used as a pointer to the rule in the rule memory. The data is organized as follows:

1. NC(k,1) is the rule primary ID.
2. NC(k,2) is the rule secondary ID.
3. NC(k,3) --> NC(k,MAXL+2): they contain the recency numbers of the facts in the LHS of the rule. Figure 4 shows the main components of the model and how they communicate with each other.

An Example. To show how the described structure can be applied to RBSs, a simplified example is considered below. This example is adapted from the animal identification RBS described in (Winston, 1984: 182-184).

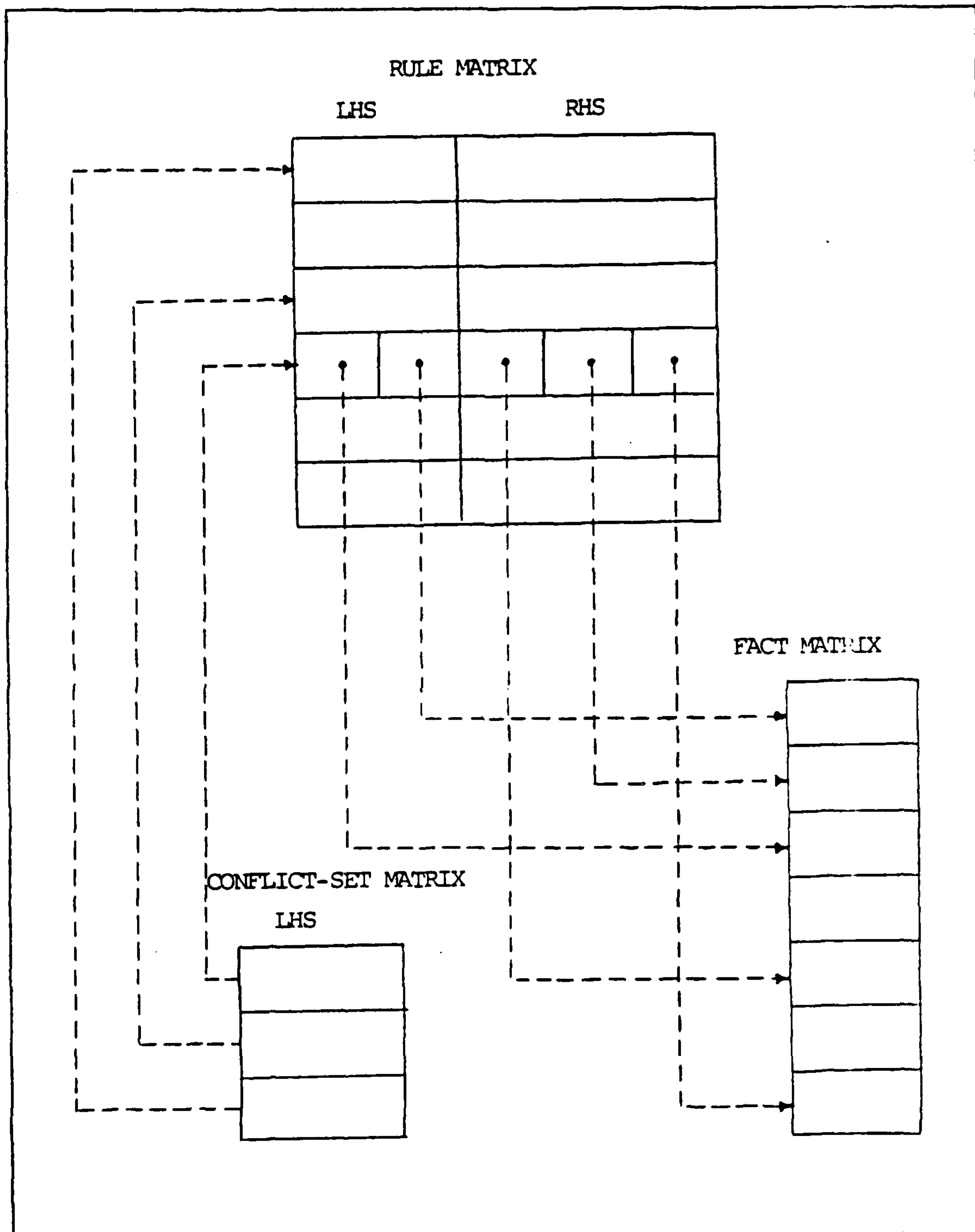


Figure 4. The Main Components of the Model

Rule 1:

IF the animal has hair

THEN the animal is a mammal

Rule 2:

IF the animal has feathers

THEN the animal is a bird

Rule 3:

IF the animal eats meat

THEN the animal is a carnivore

Rule 4:

IF the animal is a mammal

AND the animal is a carnivore

AND the animal has a tawny color

THEN the animal is a cheetah

Rule 5:

IF the animal is a bird

AND the animal does NOT fly

AND the animal has long legs

AND animal has a long neck

THEN the animal is an ostrich

An arbitrary numeric mapping to the facts may be as follows:

1 --> the animal has hair.

2 --> the animal has feathers.

3 --> the animal eats meat.

4 --> the animal has a tawny color.

5 --> the animal has long legs.

6 --> the animal has a long neck.

7 --> the animal flies.

8 --> the animal is a mammal.

9 --> the animal is a bird.

10 --> the animal is a carnivore.

11 --> the animal is a cheetah.

12 --> the animal is an ostrich.

The rules can then be written as follows:

Rule 1: IF 1 THEN 8

Rule 2: IF 2 THEN 9

RUL 3: IF 3 THEN 10

Rule 4: IF 8, 10, 4 THEN 11

Rule 5: IF 9, -7, 5, 6 THEN 12

Assuming that the rules are written in ascending order the NR matrix will be as follows:

Rule order	Rule number	Length LHS	Length RHS	Conditions				Actions	Rule status
1	1	1	1	1	-	-	-	8	0
2	2	1	1	2	-	-	-	9	0
3	3	1	1	3	-	-	-	10	0
4	4	3	1	8	10	4	-	11	0
5	5	4	1	9	-7	5	6	12	0

Assuming facts 1, 3 and 4 are the initial true assertions the NF matrix will be as shown in the next page. After the first iteration of the recognize-and-act cycle both rule 1 and rule 3 eligible to fire. The NC matrix will be as shown in the next page.

The Fact Matrix

Fact number	Fact status	Fact recency
1	1	1
2	-1	0
3	1	2
4	1	3
5	-1	0
6	-1	0
7	-1	0
8	-1	0
9	-1	0
10	-1	0
11	-1	0
12	-1	0

The Conflict-Set Matrix

Serial number	Rule order	Rule number	Recency of the LHS			
1	1	1	1	-	-	-
2	3	3	2	-	-	-

Assuming rule 1 is selected to fire then row 8 in the NF matrix will be (1 4) and the rule status of rule 1 will be set to 1. Matrices are updated to keep track of the changes that happen to the system until a stopping criterion is reached.

Assertions

Facts that are added to the data memory are known as assertions. Forward-chaining production systems start with initial assertions inserted by the user to specify which facts are known to be true. The production system then updates the contents of the data memory by firing rules until a stopping condition is satisfied. The number of facts added or deleted depends completely on the rule mix and the initial assertions, so that it is totally unpredictable. Therefore, performance is best evaluated by a posteriori testing (Neapolitan, 1986: 64).

Production systems are deterministic systems; for given initial assertions, the results will be always the same provided no changes are applied to the rule memory or the conflict-resolution strategy. However, the input to a production system (initial assertions) is not deterministic. Inputs, though they vary from one run to another, are usually a subset of a limited domain. In some problem domains the structure and the frequency of inputs can be predicted by the experts, estimated from statistics of similar systems, or estimated after running the system for some time. This posterior knowledge may be in the form of prior probabilities, conditional probabilities, or frequency figures.

As an example, suppose a RBS is constructed for a diagnostic system in which the results of three tests are the input to the system. Experts in this domain estimate prior and conditional probabilities of the results of the tests as follows:

1. Test I has 2 possible results with probabilities P(I-1), P(I-2).

2. Test II has 3 possible results conditioned on the results of the first test as follows:

P(II-1/I-1), P(II-1/I-2), P(II-2/I-1), P(II-2/I-2), P(II-3/I-1), P(II-3/I-2).

3. Test III has 2 possible results conditioned on the results of both the first and the second tests as follows:

P(III-1/I-1,II-1), P(III-1/I-1,II-2), P(III-1/I-1,II-3),
P(III-1/I-2,II-1), P(III-1/I-2,II-2), P(III-1/I-2,II-3),
P(III-2/I-1,II-1), P(III-2/I-1,II-2), P(III-2/I-1,II-3),
P(III-2/I-2,II-1), P(III-2/I-2,II-2), P(III-2/I-1,II-3).

These relations are best represented by a tree structure as shown in Figure 5, where nodes are the possible results of tests and arcs are the paths from one test to another. Nodes are assigned sequential integer numbers that represent the possible initial assertions to the system. The input to the system is a 3-tuple assertion, for example (1, 4, 10) or (2, 7, 11).

This hypothetical RBS can be simulated by randomly selecting the input from the specified probability figures. The number of matching tests (NMT) for an input can be calculated for a certain architecture. Performing a suitable amount of runs, performance can be estimated by the average value of NMT.

The given example shows a case where the number of assertions is fixed for each run and the relation among the

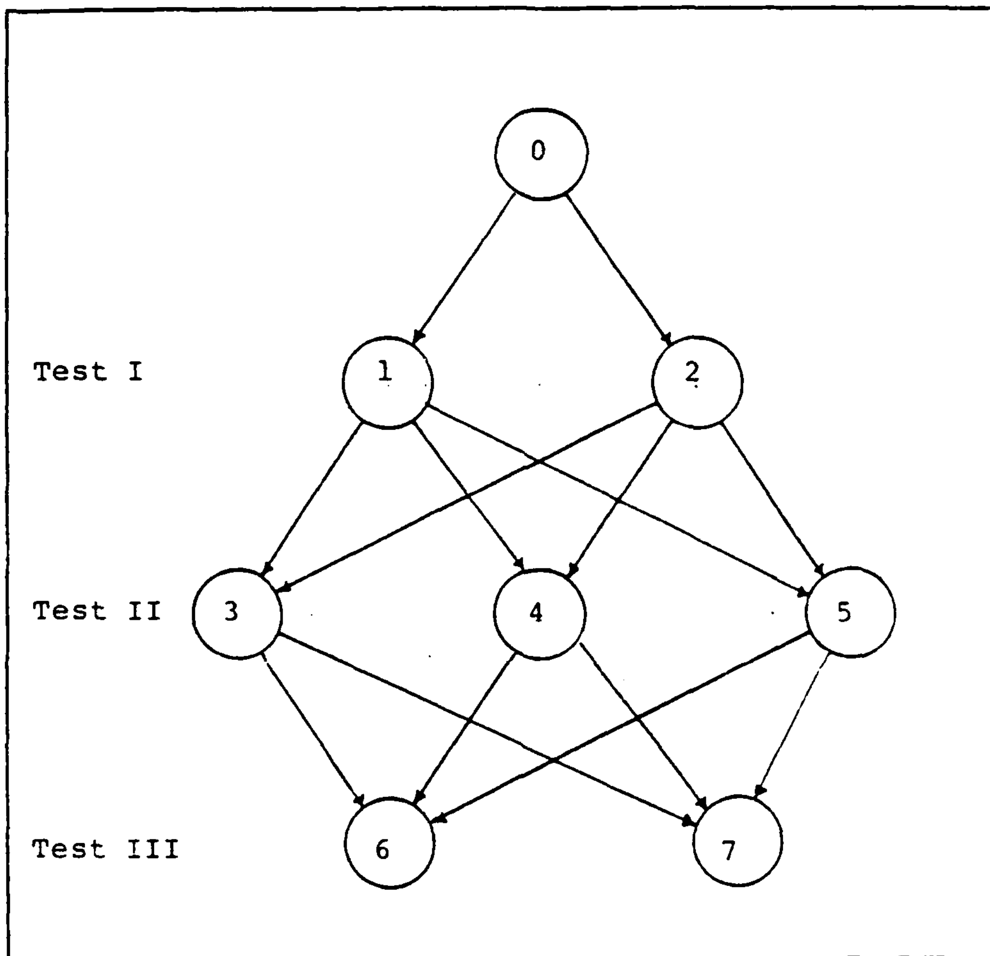


Figure 5. The Relations Among the Initial Assertions

assertions are identified by conditional probabilities. Assertions can be identified in different ways according to the nature of the system. The number of assertions may be a random number or the assretions may be divided into independent groups. All in all, if the relation among the assertions can be stochastically identified, it is always possible to simulate the input to the RBS.

The Rule Generator

The features of the present model described so far provides a methodology to simulate a specific RBS. To use this model for experimentation, a large number of RBSs with different features should be tested. It is not practical to look for already developed RBSs or to write new RBSs. The best way is to provide a method to generate RBSs. RBSs are not described only by the number of rules; other criteria can also be used to describe them. However, since RBSs are problem-dependent they are not restricted to fixed patterns. RBSs depend also on the capabilities of the language used and on the programmer using this language. To generate RBSs some criteria should be selected to describe a relatively wide spectrum of RBSs. In (Whiting and others, 1985: 215-220), as described in Chapter III, the authors discuss a rule generator program that randomly generates different structures according to parameters that define size, shape, and LHS composition. A similar approach is used in this study to automatically generate variations of RBSs that are suitable to the features represented in the model.

A rule generator should generate consistent RBSs. In (Nguyen, 1987: 4-8), the author described six types of inconsistency that may occur in RBSs. These types will be discussed at the end of this section. A rule generator should also be flexible and cover a wide spectrum of RBSs. These requirements should be satisfactorily implemented in a rule generator in order to provide credibility in the results of the experiments applied to the generated RBSs. The rule generator (RG) developed in this study is a complement to the model; the rest of this section contains a detailed description to the RG.

Basic Idea. The basic idea of the R is to divide the RBS into levels of rules; the LHS of a level is built totally or partially from the RHS of the preceding level. Facts defined as initial assertions may be also used to build the LHS of the rules. The LHS of the first-level rules are totally built from the initial assertions. The RHSs of the rules are built from new facts and they can contain facts from the RHSs of rules in the same level. Figure 6 illustrates the basic idea of the rule generator.

Parameters. A set of parameters that describe RBSs are used by RG; they are:

1. Size: size measures the number of rules that compose a RBS. Because RG divides rules into levels, the number of rules in each level needs to be specified as well. The numbers of rules in the different levels need not be identical. Three sets of rules are chosen in this study: 100 rules in three levels (30, 35, 35), 500 rules in 4 levels

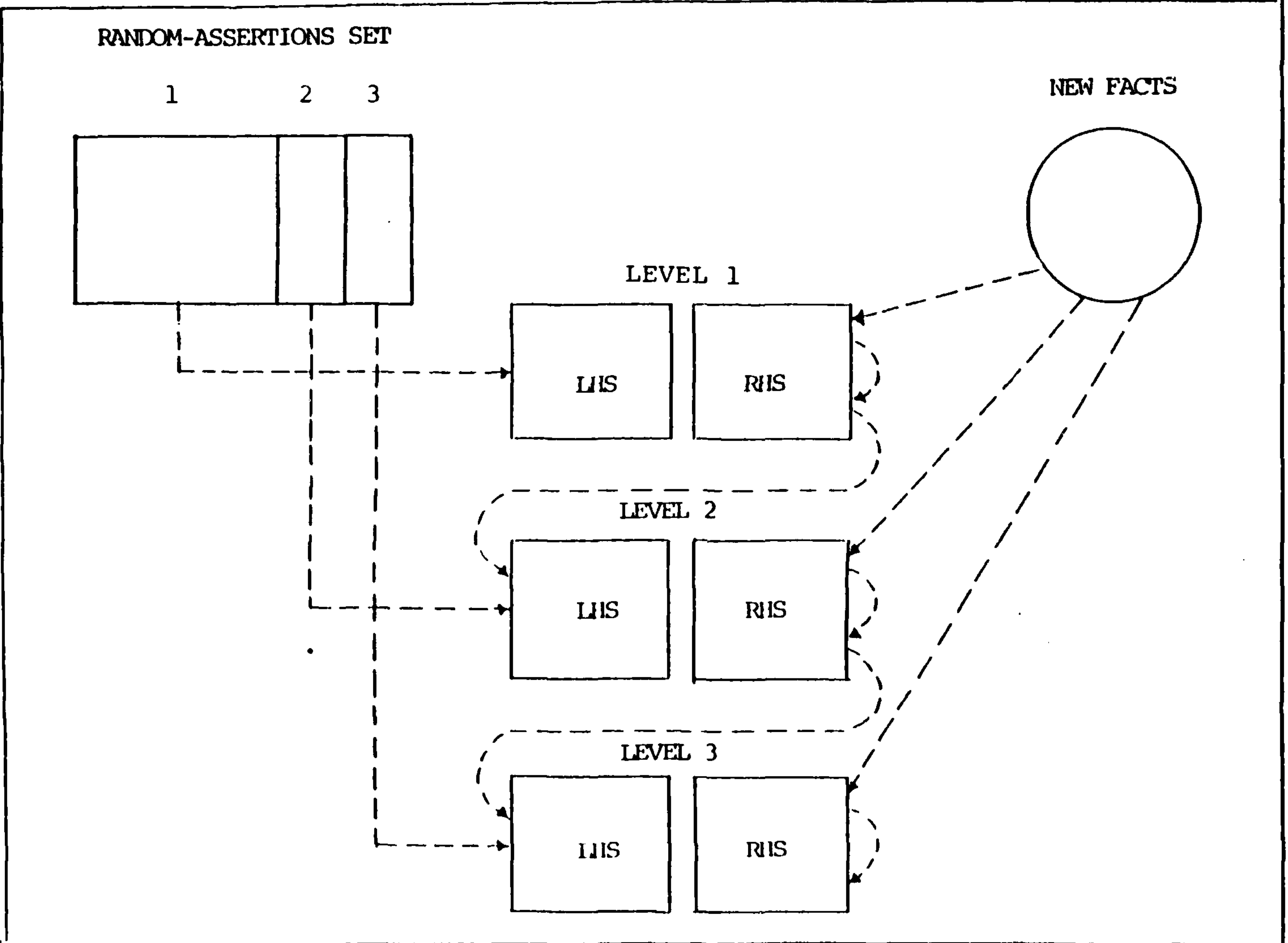


Figure 6. The Rule Generator

(125-rules each), and 1000 rules in 5 levels (200-rules each).

2. Shape: the shape parameter defines the relation between the number of facts in both sides of rules. In fan-out shape the RHS is larger than the LHS. In fan-in shape the LHS is larger than the RHS. In no-fan shape both sides are almost equal. The RG allows the user to specify the lower and upper bounds of each side for the whole RBS. The actual number in each rule is selected randomly within the specified bounds. In this study the bounds are selected as follows:

	LHS		RHS	
	Lower bound	Upper bound	Lower bound	Upper bound
Fan-out	2	2	3	4
No-fan	2	3	2	3
Fan-in	3	4	2	2

3. Negated conditions: the LHS of rules may include negated conditions by a specified percentage. This process is applied only to the initial assertions. Conditions selected from the preceding level are negated in the subsequent level if they were identified previously as deleting actions. The first condition in the first-level rules are not tested for negation, and the first condition in the subsequent levels is selected only from adding actions. This feature does not permit the RBS to have rules with all conditions negated, which is unlikely in actual RBSs.

4. Repeated actions: in actual RBSs, it is likely to have some of the actions repeated in more than one rule. This feature is permitted in RG by specifying a percentage of

actions that can be repeated. Repeated actions are selected randomly from actions in the same level.

5. Deleting actions: non-monotonic RBSs allow facts to be deleted from the data memory. This feature is permitted in R by specifying a percentage of deleting actions. Only repeated actions are allowed to be identified as deleting actions.

Initial Assertions. In the previous section a probabilistic scheme for the relation among the initial assertions is described. This scheme and two other schemes are used to help generate RBSs in addition to their principal use in experimentation. The proposed schemes will be called conditional schemes, independent schemes, and grouped schemes. These schemes are not the only ways to describe probabilistic relations among the initial assertions, but they are proposed in this study to demonstrate how this model can utilize any probabilistic scheme.

Conditional Scheme. The example described in the previous section is a typical conditional scheme. This scheme is programmed by making use of two matrices: the first matrix contains pointers to the facts branched from the current fact, and the second matrix contains the cumulative probability figures of those branches. The following two tables show an example of the two matrices which are called the ASR-matrix and the APP-matrix respectively.

ASR matrix

Row			
0	1	2	-
1	3	4	5
2	3	5	-
3	6	7	8
4	6	7	8
5	7	8	-

APP matrix

Row			
0	.4	1	-
1	.2	.5	1
2	.6	1	-
3	.3	.6	1
4	.1	.9	1
5	.7	1	-

The matrices are accessed directly starting from row 0 which branches to either row 1 or row 2 according to uniform random draw. A second random draw decides the next row, and so on. In this example, the 3-tuple assertion (1, 4, 7) would be generated by the random-number sequence (.35, .46, .87). Another random-number sequence (.95, .71, .01) would generate the assertion sequence (2, 5, 7).

Independent Scheme. In an independent scheme, the initial assertions are divided into independent subsets. Each subset contains assertions having a common probability distribution. Only one assertion from each subset is selected randomly. The ASR and APP matrices for an example similar to the one presented in the conditional scheme will be as follows:

ASR matrix

Row			
1	1	2	-
2	3	4	5
3	6	7	8

APP matrix

Row			
1	.4	1	-
2	.3	.7	1
3	.2	.8	1

The matrices are accessed sequentially in this case to determine the assertions. For example the random number sequence (.25, .06, .11) generates the assertion sequence (1, 3, 6).

Grouped Scheme. In a grouped scheme, instead of selecting one assertion from each subset as in the independent scheme, a fixed number of assertions (group) is selected. The members of the group are selected randomly with equal probability and the sampling is performed without replacement to prevent repetition. Only one matrix is needed in this scheme. The first column in the matrix contains the first fact in each subset, the second column contains the last fact, and the third column contains the group size. The matrix shown below represents a grouped-scheme example.

Row			
1	1	10	4
2	11	26	5
3	27	38	3
4	39	50	3

An example of a randomly generated assertion sequence would be (1, 3, 9, 5, 25, 15, 13, 19, 21, 27, 37, 30, 47, 39, 41).

Procedure. To generate a RBS, the RG program should determine which facts are used to build the LHS of the rules as a first step. The LHS of the first-level rules is built from the initial assertions. Assume facts 1 --> 8 are the possible initial assertions and they have the probabilistic

relation described in the first example in the previous subsection. If a fan-out RBS is to be generated, then the LHS of the rules will consist of two conditions. To build a rule that can fire from a randomly-generated assertion sequence, the facts chosen to compose the LHS must be consistent with the probabilistic relation among the facts. A rule with a LHS consisting of facts (1, 2) or facts (2, 4) will never fire. In the first case, facts 1 and 2 are mutually exclusive, they can not be true at the same time. In the second case there is no branch from fact 2 to fact 4. If the LHS are generated completely randomly, many nonsense rules will be generated. The best way to build the LHS of the rules is to use a randomly generated assertion sequence which is guaranteed to be consistent. For example, the assertion sequence (1, 4, 7) can be used to generate several consistent LHSs such as: (1 ,4), (1, -7) or (7, -4).

The procedure used in RG is first to generate a number of assertions from a probabilistic relation among the initial assertions, then to select some of them randomly to build the LHSs of the first-level rules. The conditions that compose a LHS are selected randomly from facts within the assertion. Conditions, except the first one, may be negated according to the specified negation percentage.

The RHSs of first-level rules are composed of facts that start from where the initial assertions end. For the given example the RHS starts from fact 9. If the RHS of the first level is composed of three actions, then they will be the facts (9, 10, 11). If a non-zero repetition percentage is

specified, facts can be repeated in more than one rule. The procedure used in R is to generate some rules at the beginning of each level without repetition, to give a chance for adding new facts before repeating them. The number chosen for unrepeated rules are 2, 3, and 4 for fan-out, no-fan, and fan-in respectively. For the given example, the RHS of the second rule may be the facts (12, 13, 14, 15), and the RHS of the third rule may be (9, 16, 17, 13) where facts 9 and 13 are repeated actions. If a non-zero delete percentage is specified, some of the repeated actions may be identified as deleting actions. For example, the RHS of the third rule may become (-9, 16, 17, 13) where fact 9 will be deleted if rule 3 is fired and the fact already exists in the data memory.

The LHS of the second-level rules consists of the RHS of rules from the first level. First, a rule is selected randomly from the first level, then a number of actions, equal to the number of conditions needed for the second-level rule, is selected randomly from the RHS of the selected first-level rule. For example, assume rule 2 is randomly selected to build rule 40 in the second level. If the number of conditions in rule 40 is 2, then facts (14, 12) may be selected for the LHS of the rule. Facts are copied from the RHS to the LHS with the same signs (+ or -). For example, if rule 3 is selected instead of rule 2, the RHS of rule 40 may be (-9, 17). This procedure is sufficient for the fan-out shape where the LHS is always greater than the RHS. In the case of no-fan and fan-in shapes, extra initial assertions, similar to the first-level assertions, need to be assigned to the

subsequent levels. To illustrate this situation, assume that the data of the grouped-scheme example presented previously are used to generate a fan-in RBS composed of three levels. The first two rows of data are assigned to the first level, the third row is assigned to the second level, and the fourth row is assigned to the third level. Examples of first-level rule are:

Rule 1: IF 2, 4, -15, 19 THEN 51, 52

Rule 2: IF 16, -3, 10, -1 THEN 53, 54

Rule 3: IF 14, -24, 4 THEN 55, -51

The RHS of rule 1 starts from fact 51 because the last initial assertion allowed is 50.

Assume that the second level starts with rule 31, which consists of four conditions, and rule 3 is randomly selected to build the LHS of rule 31. The first two conditions of rule 31 will be 55 and -51. The other two conditions are selected randomly from the facts (27 --> 38) assigned to the second level. As mentioned before, rules are built from previously generated random assertions to guarantee the consistency of the rules. Any random selection from grouped-scheme data will be consistent; however, the same procedure used for the other schemes is applied also to the grouped scheme just to unify the procedure. To illustrate the procedure used to generate random assertions for all levels, consider the same example of grouped scheme. To generate random assertions for a RBS, four extra parameters are identified; they are:

1. NA1, the total number of assertions for the first level.

2. NA2, the total number of assertions for each of the other levels.

3. NTA1, the number of true assertions for the first level.

4. NTA2, the number of true assertions for each of the other levels.

The procedure assumes that levels other than the first one have identical specifications. This assumption is used to simplify the procedure and reduce the amount of input data. The value of these parameters, for the case given in the example, can be determined from the given data; they are 26, 12, 9, and 3 respectively. Each random assertion will consist of 1 cells. The first 9 cells are used to build the first-level rules, the next 3 cells are assigned to the second level, and the last three cells are assigned to the third level. The process of building the LHS of the rules is performed in two steps: first the locations of cells are selected randomly, then the value of the facts in the selected cells are copied into the LHS of the rule. If negation is permitted, a negation test is applied to each fact separately. The two conditions needed for rule 30 in the example are determined first by selecting two cells randomly from the candidate cells 10, 11, and 12. Assume cells 10 and 12 are selected and the random assertion is the one given in the example, then the conditions will be 27 and 30. The final structure of LHS of rule 31 will be the facts (55, -51, 27, 30). The RHS of rule 31 will start with the fact next to the last condition added to the first level.

Any of the three proposed probability schemes can be used to generate RBSs of any size and any shape. Fan-out RBSs need few initial assertions to fire a reasonable number of rules in each level. No-fan RBSs need more initial assertions, and fan-in RBSs need the most. The number of initial assertions should also increase with the increase in size (the number of rules in the rule set). A separate program is used to generate random assertions from each scheme. The probabilistic relations are specified in an input file to the program. The output of the program is a file containing a set of random assertions. The RG program uses the random assertions file as an input. Also, a file containing the specifications of the required RBS is needed as an input to the RG program. A rule specification file contains the following data items:

1. NRULE, the size.
2. NLVL, number of levels.
3. LVL(1), LVL(2), ..., number of rules in each level.
4. MINL and MAXL, the lower and upper bounds of the LHS.
5. MINR and MAXR, the lower and upper bounds of the RHS.
6. PN, negation percentage.
7. PR, repetition percentage.
8. PD, deletion percentage.

It also contains the NA1, NA2, NTA1, and NTA2 parameters. Listings of the programs and samples of the input and output are shown in the Appendix.

Consistency of the Generated RBSs. As mentioned early in this section, six types of inconsistency in RBSs were

described in (Nguyen, 1987: 4-8); they are the following:

1. Redundant rules:
   Two rules or rule chains are redundant if they succeed in the same situation and have the same conclusions.
2. Conflicting rules:
   Two rules or rule chains are conflicting if they succeed in the same situation but with conflicting conclusions.
3. Subsumed rules:
   Two rules or rule chains are in subsumption if they have the same conclusions, but one contains additional constraints on the situations in which it will succeed.
4. Unnecessary conditions:
   Two rules contain unnecessary conditions if the rules have the same conclusions, a condition in one rule is a negation of a condition in the other rule, and all other conditions in the two rules are equivalent.
5. Unreachable conditions:
   If there is no match for a condition, it is said to be unreachable.
6. Circular rules:
   A set of rules is circular if the chaining of those rules in the set forms a cycle.

The following paragraphs discuss the possibility of generating each of the six types of inconsistency in the RBSs generated by the RG program.

1. Redundant rules: generating rules of equivalent LHSs is fairly probable for RBSs with small LHS, especially for the first-level rules. If the repetition percentage is high then generating RHSs with at least one common action is frequent and may occur to those rules having equivalent LHSs. The probability of generating redundant rules is higher for the first few rules of each level. However, because RG does not allow repetition in the first few rules, this probability is reduced.

2. Conflicting rules: recognizing conflicting rules in monotonic RBSs requires the identification of conflicting

facts. For example, facts 17 and 24 may be defined as conflicting facts that cannot be true at the same time. There is no need to complicate the generated RBSs by identifying such relations. In non-monotonic RBSs, two rules are considered as conflicting rules if they have an equivalent LHS and one of them adds a fact and the other deletes the same fact. This situation may occur in the generated RBSs if the LHS is small and both the repetition and deletion percentages are high.

3. Subsumed rules: generating two rules with equivalent RHSs is fairly probable for RBSs with small RHS, and generating two rules with LHSs differing only by the addition of one condition in one of them is frequent in RBSs with small LHSs. Generating rules with both cases would occur if both the LHS and the RHS are small. However, if either or both sides is long enough, the probability of generating subsumed rules is very small.

4. Unnecessary conditions: The causes that lead to unnecessary conditions are quite similar to the causes of subsumed rules. By the same argument, generating rules with unnecessary conditions is very rare if either or both sides is long enough.

5. Unreachable rules: the conditions of the rules are generated either from initial assertions that can be true or from the RHS of rules that have a positive probability to fire. So, all conditions are reachable.

6. Circular rules: two rules are circular if the LHS of

the first rule is equivalent to the RHS of the second rule, and in the same time the RHS of the first rule is equivalent to the LHS of the second rule. The RG program may generate the first part of this condition, especially for RBSs with small LHSs, but it never generates the second part at the same time because it composes the LHS from the preceding level while the RHS is composed either from new facts or from repeated facts at the same level.

In summary, the probability of generating the six types of inconsistency can be classified as low for types 1 and 2, very low for types 3 and 4, and zero for types 5 and 6. Eliminating inconsistent rules completely can be achieved by controlling the LHS to prevent generating equivalent LHSs. Such a process will slow down the RG especially for large RBSs. Types 1 and 2 can be greatly reduced if the LHS of the rules consist of two or more facts. Types 3 and 4 can be almost eliminated if both sides of the rules consist of two or more facts. The specifications of the RBSs selected for experimentation include at least two facts in both sides, as mentioned at the beginning of this section, in order to generate highly consistent RBSs.

Usage. The developed RG program can generate highly consistent RBSs in different specifications that cover a wide spectrum of RBSs handled by the current implementation of the model. The choice of the parameters of a RBS and the number of initial assertions for each level should be well balanced in order to generate a good RBS. Understanding the relations

among the parameters requires practicing with the RG for a while, and preparing the input files may consume some time. However, the execution time of the programs is unnoticeable even for a 1000-rule RBS.

V- The Simulated Inference Engines

In Chapter IV a detailed description of the representation of a knowledge base (data memory, rule memory, initial assertions) used in the model is presented. Also the method used to generate a wide spectrum of RBSs is explained. In this chapter, a description to the different types of inference engines simulated in this study is presented. Conflict resolution and filtering are the main features of inference engines as explained in Chapter II. Six types of forward-chaining inference engines are simulated. Each inference engine is characterized by a conflict-resolution strategy and a filter technique (if any). The six types are the following:

1. Engine 1: production order (PO1) conflict-resolution strategy and no filtering.

2. Engine 2: lexical-order recency (R5) conflict-resolution strategy and no filtering.

3. Engine 3: most recent (R1) conflict-resolution strategy and no filtering.

4. Engine 4: least recent (R3) conflict-resolution strategy and no filtering.

5. Engine 5: production order (PO1) conflict resolution-strategy and rule filtering (controlled productions)

6. Engine 6: most recent (R1) conflict-resolution strategy and rule filtering (context restricted).

In all engines the distinctiveness conflict-resolution strategy is applied to prevent rules from firing twice. In

engines 2, 3 and 4, an arbitrary decision is applied if necessary to break ties. For each engine, two versions of the computer program that simulates each engine are written. The first version is used for the detailed results of one run. The second version is used to collect statistics for a specified number of runs. The rule memory file (the output file of the R program or a representation of a specific RBS) is used as an input file to the required program. In version 1 the initial assertions are entered interactively, while in version 2 a file that contains the initial assertions for the specified number of runs is used as another input file. The output of version 1 is a file containing the firing rules and the value of the number of match-tests (NMT). In version 2, two output files are produced: the performance file and the statistics file. The performance file contains the values of the NMT for each run and their average value; the statistics file contains statistics about the number of times each fact is added to the data memory, and the number of times each rule fires.

At the beginning of each program the rule memory file is read and stored in a matrix. A fact matrix is established and initialized according to the initial assertions. All programs include two main processes which are called: check a rule, and fire a rule. The details of the two processes are as follows:

1. Check a rule:

The purpose of this process is to decide whether a rule is eligible to fire or not. No search is needed to accomplish this process. Each condition in The LHS of the rule is

checked directly by inspecting the status of the condition in the fact matrix. The result of the check depends also on the nature of the condition (positive or negative). The NMT is represented by a counter which is incremented by one after each check irrespective of the result of the check. If a condition fails the test, no further checking is needed. The rule is eligible to fire if all its conditions pass the test.

2. Fire a rule:

When a rule is selected to fire, the rule status flag is set to mark the rule. The status of the actions in the RHS of the rule, the first column in the fact matrix, is modified according to the nature of the action (add or delete a fact). After firing the rule another cycle starts from the beginning of the rule matrix. The procedure of each program is described in the following sections, and the listings of the programs are shown in the Appendix.

Engine 1

Engine 1 is the simplest forward-chaining inference engine. A straight-forward procedure is used according to the following steps:

1. Read the first rule.

2. IF the rule status flag is set

   THEN go to 6

   ELSE go to 3.

3. Check the rule.

4. IF a condition fails
   THEN IF the rule is not the last one

      THEN go to 6

ELSE stop.

ELSE go to 5.

5. Fire the rule.

6. Read the next rule,

   go to 2.

Engine 2

In engine 2, the conflict-set matrix described in Chapter IV is used to store the recency of the conditions in the LHS of the rules eligible to fire. Another matrix is established to store the same values after sorting them for each rule in descending order. To facilitate the selection of a rule according to R5 policy, the recency of the sorted facts are converted into one equivalent number, the lexical value, which is a function of both the order and the value of the recency of all the facts in the rule. The lexical value of all the rules in the conflict set is stored in an array, then the rule having the maximum lexical value is selected to fire. The procedure can be summarized in the following steps:

1. Read the first rule.

2. IF the rule status flag is set

   THEN go to 7

   ELSE go to 3.

3. Check the rule

4. IF a condition fails

   THEN IF the rule is not the last one
        THEN go to 7

        ELSE IF the conflict set is empty

THEN stop

ELSE go to 5

ELSE add the rule to the conflict set,

IF the rule is not the last one

THEN go to 7

ELSE go to 5.

5. Sort the LHS of the rules in the conflict set, calculate the lexical number for the rules, select a rule to fire.
6. Fire the rule.
7. Read the next rule, go to 2.

<u>Engine 3</u>

Engine 3 uses the same procedure as engine 2 except for step 5. After sorting the LHS of the rules in the conflict set, the rule having the maximum first condition recency number (most recent of the most recent) is selected.

<u>Engine 4</u>

Engine 4 uses the same procedure as engine 2 except for step 5. Sorting is applied in ascending instead of descending order, then the rule having the maximum first condition (most recent of the least recent) is selected.

<u>Engine 5</u>

The procedure of engine 5 is similar to the procedure of engine 1. However, instead of checking all rules in the rule matrix in every cycle, the rules are divided into groups

taken one at a time until all the rules eligible to fire in a group fire. The procedure works as a filter that allows a specific subset of the rule memory to be checked in each cycle. The formation of the groups should be entered to the program as input data. In this program, it is assumed that the rules in the rule memory file are ordered such that it is enough to specify the order of the rule in the end of each group. For the RBSs generated by RG, each level of rules can be considered as a group. If a specified RBS is used it should be ordered first.

Engine 6

Engine 6 is an implementation to the condition-membership filter (McDermott and others, 1978: 160-161) as described in Chapter III. The procedure used in this study is similar to the one discussed in (Whiting and others, 1985: 215-220) because the degree of abstraction used in both studies is quite similar in spite the fact that the data structure is different. The authors described their procedure, which they called the RF (Rule Focus) inference engine as follows:

> RF does not evaluate every rule on every cycle. Instead, it evaluates only the unfired rules that contain in their antecedent (LHS) the last fact entered in the context base (data memory). To locate these rules it utilizes an index, built when the rule set is first read, which consists of all facts which appear in the rule base (rule memory), and for each fact a list of all rules whose antecedent contains that fact. As a fact enters the context base, the fact list is searched to find it, and the corresponding rule list for that fact is traversed ..

In this study the index is organized as the fact matrix where it can be accessed directly, rather than sequentially,

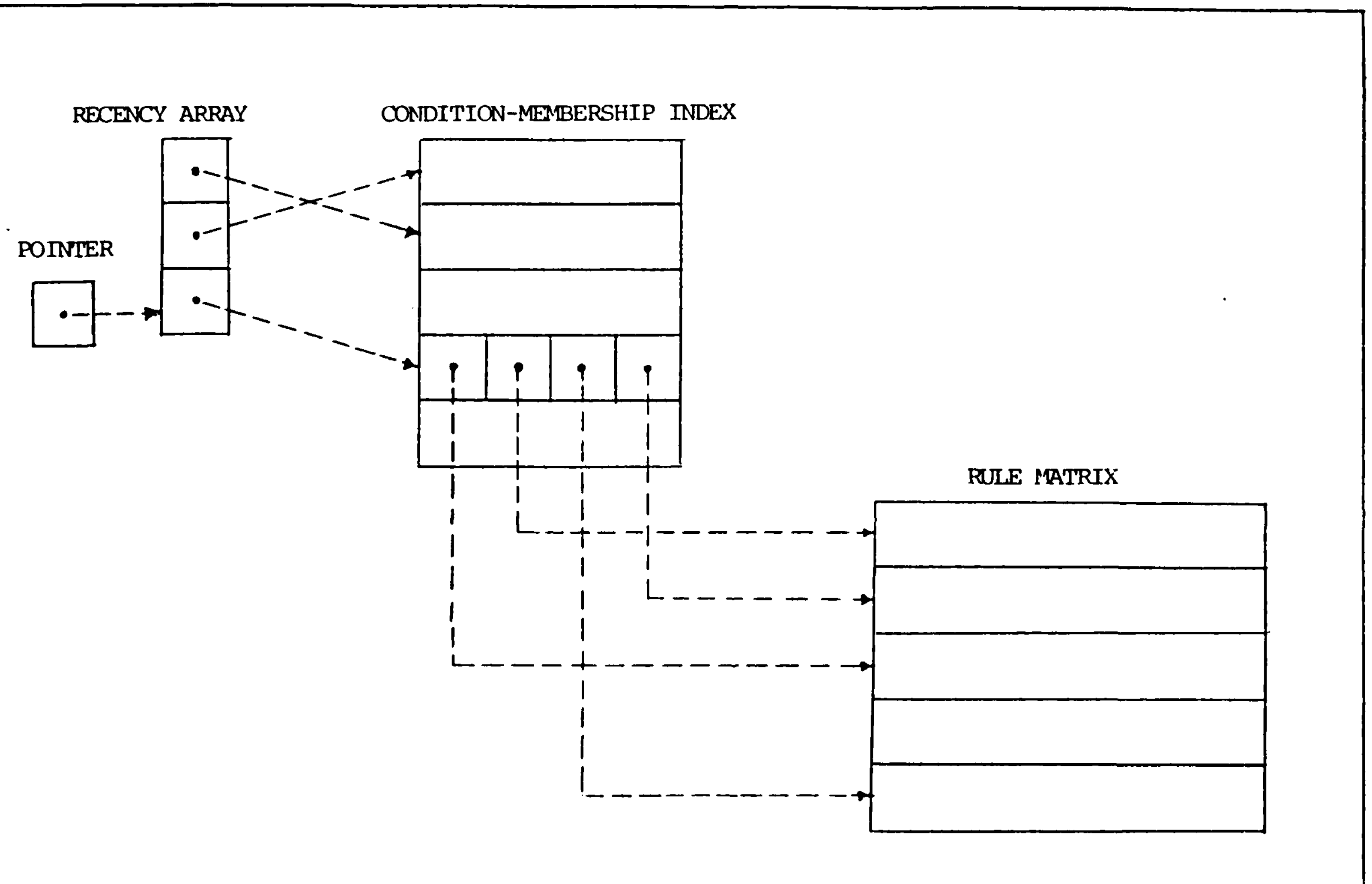


Figure 7. The Condition-Membership Filter

by specifying the fact number. The first cell of each row is used as a pointer to the first empty place in the row. The index is constructed by reading the rule matrix sequentially and storing the rule order in all rows of the index corresponding to the conditions in the LHS of the rule. To implement the R1 policy, the most recent fact should be determined. The direct approach is to search for this fact through the fact matrix. This approach may involve searching more than once to find the next most recent fact if no rule related to the most recent fact is eligible to fire. An alternative approach used in this study, avoids such repeated search. A separate array is established to save, in sequential order, the value of the facts added to or deleted from the data memory in sequential order. In addition, a pointer is created to maintain the location of the last fact added to this array, i.e., the most recent fact. The next most recent fact can simply be determined by decrementing this pointer by one. Figure 7 illustrates the approach used for implementing the condition-membership filter.

When the most recent fact is determined, only the rules related to this fact, as specified in the index, are checked in sequential order. The first rule eligible to fire from those rules is selected. If no such rule is found, the next most recent fact is determined and the same procedure is applied. The procedure can be summarized in the following steps:

1. Find the most recent fact.

2. Read the first rule related to the most recent fact.

3. IF the rule status flag is set

   THEN go to 7

   ELSE go to 4.

5. IF a condition fails

   THEN IF the rule is not the last one in the index row

        THEN go to 7

        ELSE IF the fact is the least recent fact

             THEN stop

             ELSE find the next most recent fact,

                  go to 7.

   ELSE go to 6.

6. Fire the rule.

7. Read the next rule related to the current fact,

   go to 1.

Verification

To verify the set of programs that simulate the different inference engines, two tests are applied. In the first test, an actual small RBS consisting of 15 rules, the animal-identification RBS (Winston, 1984: 182-184), is translated into the numeric representation used in the model. Several runs with different initial assertions are applied to each inference engine to verify that the correct results are produced. In the second test, several generated RBSs with different specifications are applied to all engines. The results are compared to verify that the same results are

produced by all engines. In monotonic RBSs, all the engines produce the same results, but the order of firing may change according to the conflict-resolution policy. In non-monotonic RBSs, the conflict-resolution policy may affect the results in addition to the firing order. In some cases, the rule-ordering policy fires a few rules more or less than the recency policy for the same assertions. This difference is a result of the conflict-resolution policy and the nature of the RBS itself; it does not indicate any type of error in the programs.

VI. Design of Experiments

The representation of rule-based systems (RBSs) used in the model provides a convenient environment for experimentation. Experiments can be performed on actual or generated RBSs for the purpose of comparing their performance. Experiments can be used to compare different implementations for the inference engine or the structure of the rule set. Two types of experiments are performed in this study: the first provides a methodology to enhance the performance of a specific RBS by changing the order of the conditions of the LHS of the rules or changing the order of the rules in the rule set, and the second compares the matching effort for different implementations for the inference engine. In both types, the number of match-tests (NMT) of the conditions of the LHS of the rules against the contents of the data memory is used as the measure of effectiveness.

The rule generator (RG) program is used to generate the RBSs used in the experimentation. The independent-probability scheme described in Chapter IV is used to describe the probabilistic relations among the assertions of the generated RBSs. The initial assertions are divided into independent subsets of four assertions each. An arbitrary probability distribution is assigned to each subset. One data file containing 200 facts in 50 subsets and their probability distributions is used as an input file to the independent-scheme program. Only the part of the file needed to generate random assertions for a specific RBS is read, stored, and

used in the independent-scheme program. The number of initial assertions used for the different sizes and shapes selected to generate the RBSs used in experimentations is as shown in the following table:

shape	size	number of levels	number of initial assertions level 1	other levels	total
Fan-out	100	3	4	-	4
	500	4	6	-	6
	1000	5	8	-	8
No-fan	100	3	5	1	7
	500	4	10	2	16
	1000	5	16	3	28
Fan-in	100	3	7	3	13
	500	4	20	5	35
	1000	5	25	6	49

The negation probability for facts in the LHS of the rules and the repetition probability for facts in the RHS of the rules selected for the generated RBSs are 0.2 and 0.4 respectively. For non-monotonic RBSs the selected deletion probability is 0.4. It should be noted that these probabilities do not represent the actual percentages in the generated RBSs because of the restrictions applied in the RG program in order to generate consistent RBSs as explained in Chapter IV. The actual percentages are less than the specified probabilities, but they cannot be accurately estimated and there is no need to do that. These probabilities are used only to help generate RBSs that look

like actual RBSs. For each of the nine RBSs defined in the table, two versions of RBSs' specifications files are written: the first includes the monotonic version and the second includes the non-monotonic version.

Three sets of random assertions are generated for each RBS used in experimentation. The first set is used as input to the RG program to be used in generating the RBS. The second and third sets are used as random samples from the assertions population of the simulated RBS. These sets of assertions will be called set 0, set 1, and set 2 respectively. Figure 8 illustrates the basic sequence of experimentation.

The statistic used in the experiments is the reduction ratio in matching effort. For example, if the estimated mean of NMT of system 1 is X, and the estimated mean of NMT of system 2 is Y where X is greater than Y, then the statistic is (X - Y)/X.

Experiments are applied for 18 design points which represent the intersections of the following three factors:

1. Shape (3 levels: fan-out (FO), no-fan (NF), and fan-in (FI)).

2. Size (3 levels: 100, 500, and 1000 rules).

3. Monotonicity (2 levels: monotonic and non-monotonic).

One Thousand replications in two independent runs of size 500 each are applied for each design point using the generated random assertions set 1 and set 2. The following sections

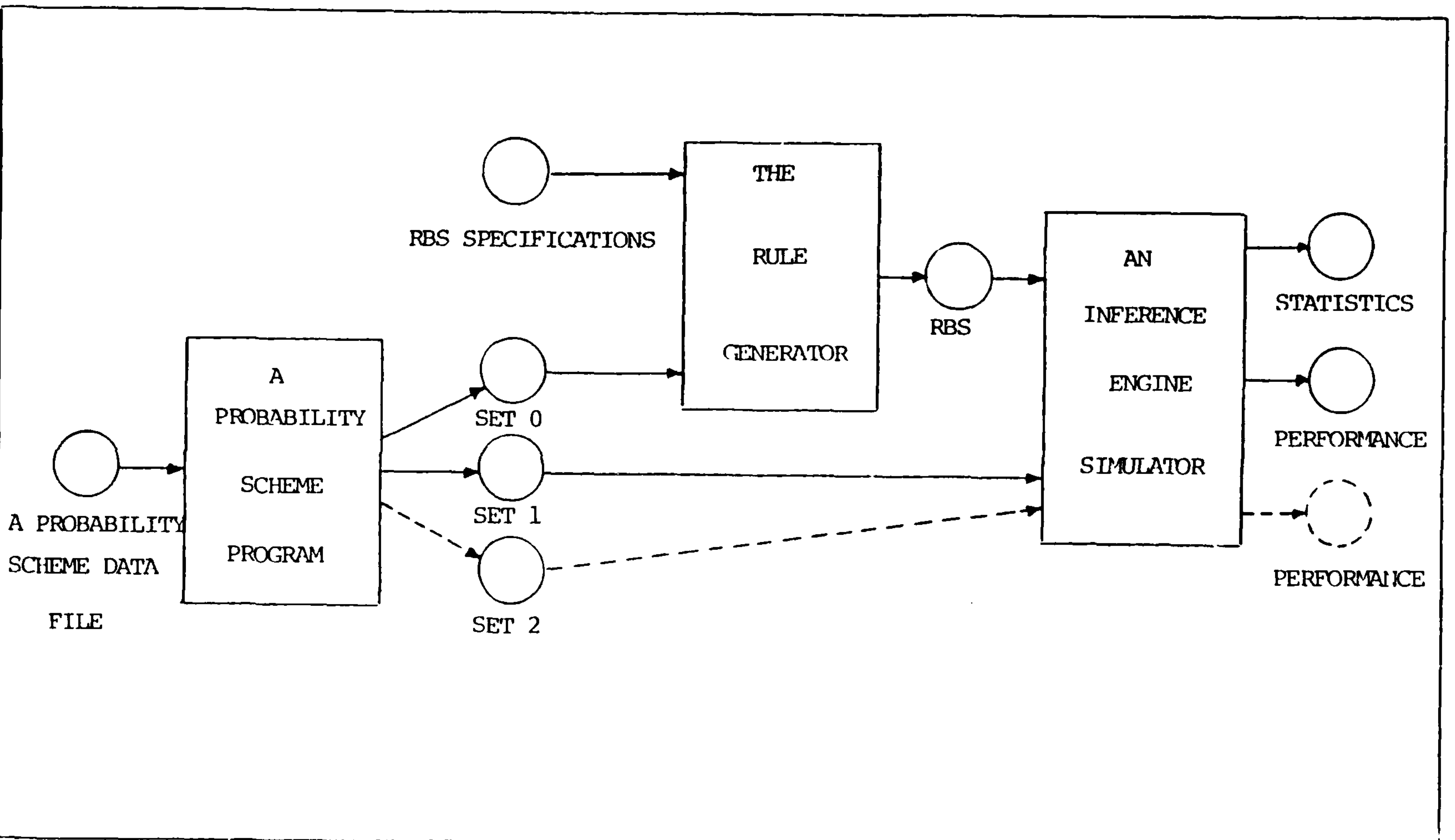


Figure 8. Basic Experimentation Sequence

describe the experiments and present a summary of their results.

Type 1 Experiments: Performance of a Specific RBS

Experiment 1: The Order of the Conditions in the LHS. As mentioned in Chapter III, the execution time of a RBS can be reduced by rearranging the conditions in the LHS of the rules such that the most restricted conditions are tested first. If a rule is going to fail the matching test, then it is better to fail after one test instead of two or more tests. This rule seems simple; however, the question is how to identify the most restricted conditions. It is unlikely to identify such conditions for all the rules in the rule set only by the prior information about the system. One measure that can identify the degree of restriction of the conditions is the number of times a condition passes the matching test (the less often a fact passes the test the more restricted it will be). Another measure is the number of times a fact is added to the data memory (the less often a fact is added to the data memory the more restricted it will be). Given the probabilistic relations among the initial assertions, the model is able to simulate the system and estimate the degree of restriction of all the facts that comprise the RBS using either measure. Then, the LHS of the rules can be sorted in ascending order according to the frequency of the measure used. As mentioned in Chapter V, version 2 of each program that simulates an inference engine, the simulator, collects statistics about the performance of the RBS during the specified number of runs and stores the data in a file. The

number of times each fact is added to the data memory is stored in the statistics file. A computer program, the fact-ordering program shown in the Appendix, is written to sort the conditions. The program uses both the statistics file and the RBS file as inputs and produces an output file containing the newly-structured RBS.

The experiment is applied twice: the first experiment is applied for the 18 design points of the factor space using engine 1, which uses the production order conflict-resolution policy with no filtering; and the second is applied for the 100-rules RBSs using all the six engines. For the given RBS and inference engine, the experiment is performed according to following steps:

1. Run the simulator using set 1 of the random assertions and the original structure of the RBS and collect the statistics (run 1).

2. Rearrange the RBS.

3. Run the simulator using set 1 and the new structure of the RBS (run 2).

4. Run the simulator using set 2 of the random assertions and the original RBS (run 3).

5. Run the simulator using set 2 and the new structure of the RBS (run 4).

6. Estimate the reduction ratio between run 1 and run 2.

7. Estimate the reduction ratio between run 3 and run 4.

The reason for applying the second replication to the new structure generated from the first replication is to find if the new structure will perform better for any random set or

only for the random set that generates the structure. The table below summarizes the procedure of the experiment and Figure 9 illustrates the sequence of the experiment.

	Set 1	Set 2
Original RBS	Run 1	Run 3
New RBS	Run 2	Run 4

Results. Tables 1 and 2 show the results of the first part of the experiment, which is applied to engine 1 for the 18 design points. Tables 3 and 4 show the results of the second part of the experiment, which is applied to all the engines for the 100-rules size only. The reduction percentage shown in the tables is the average of the reduction ratio of the two random sets multiplied by 100. Figures 10 and 11 depict the results of the first part of the experiment.

Table 1 The Effect of Fact Ordering on Monotonic RBSs (Part 1)

Shape	Size	Reduction Percentage
fan-out	100	4.820
	500	6.055
	1000	6.925
no-fan	100	6.320
	500	7.250
	1000	5.435
fan-in	100	3.845
	500	6.665
	1000	6.015

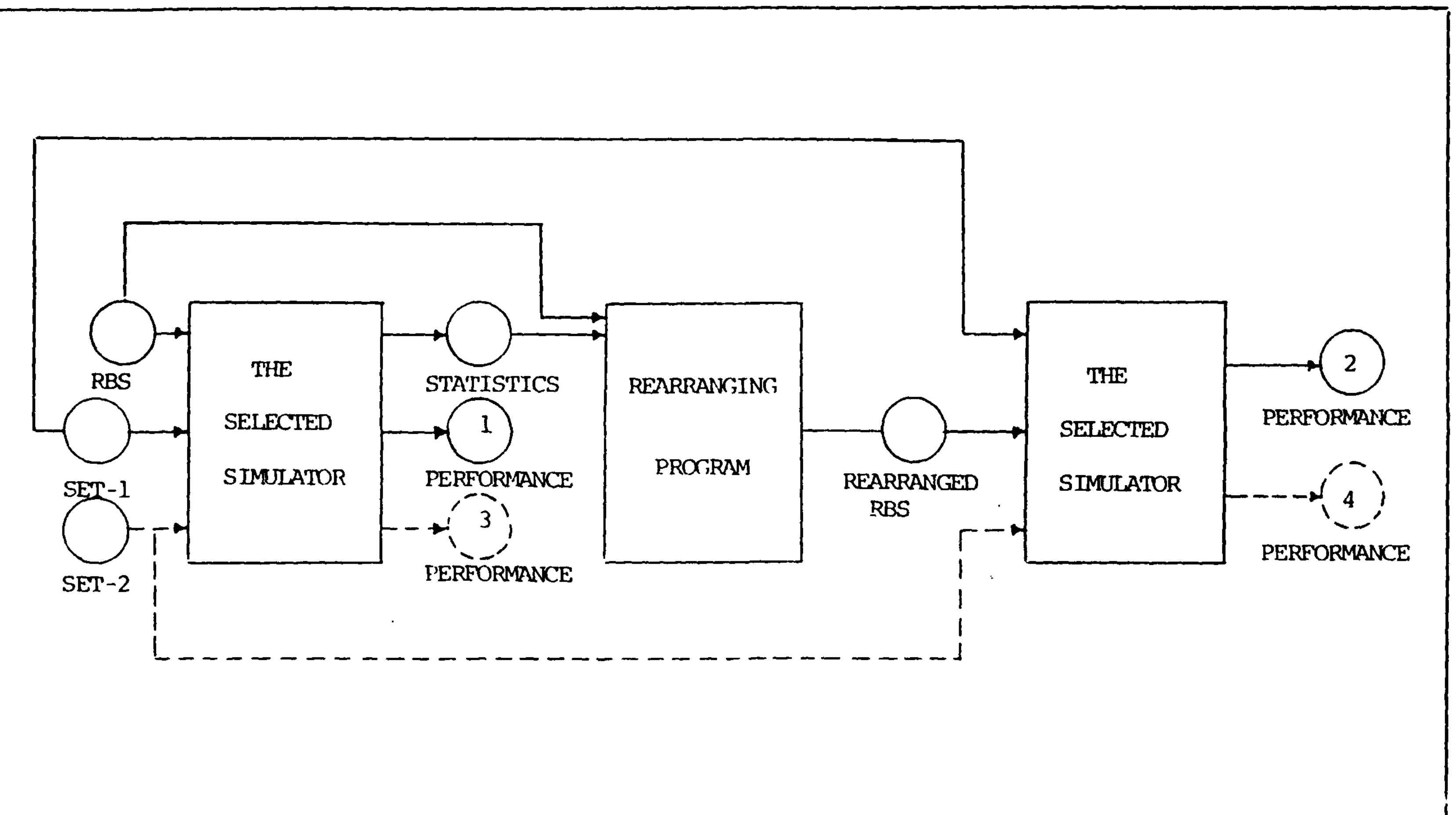


Figure 9. Type-1 Experiments

Table 2 The Effect of Fact ordering on Non-Monotonic RBSs (Part 1)

Shape	Size	Reduction Percentage
fan-out	100	2.925
	500	3.255
	1000	3.980
no-fan	100	6.225
	500	8.540
	1000	9.290
fan-in	100	9.490
	500	11.175
	1000	10.770

Table 3 The Effect of Fact Ordering on Monotonic RBSs (Part 2)

Shape	Engine	Reduction Percentage
fan-out	1	4.820
	2	3.700
	3	3.745
	4	3.505
	5	4.730
	6	12.445
no-fan	1	6.320
	2	5.480
	3	5.805
	4	5.520
	5	7.595
	6	15.745
fan-in	1	3.845
	2	2.805
	3	2.805
	4	2.835
	5	3.560
	6	4.015

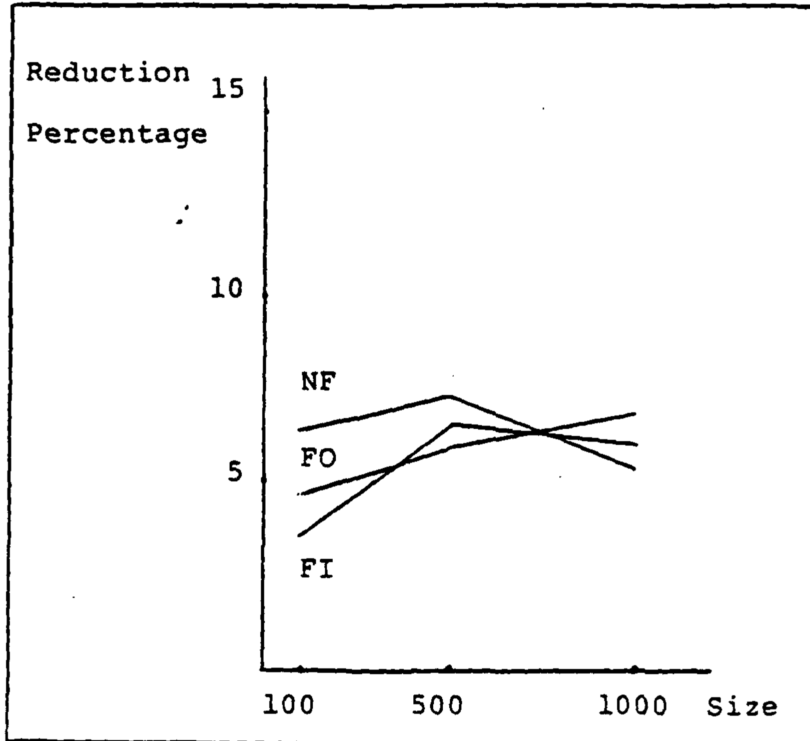


Figure 10. The Effect of Fact Ordering on Monotonic RBSs

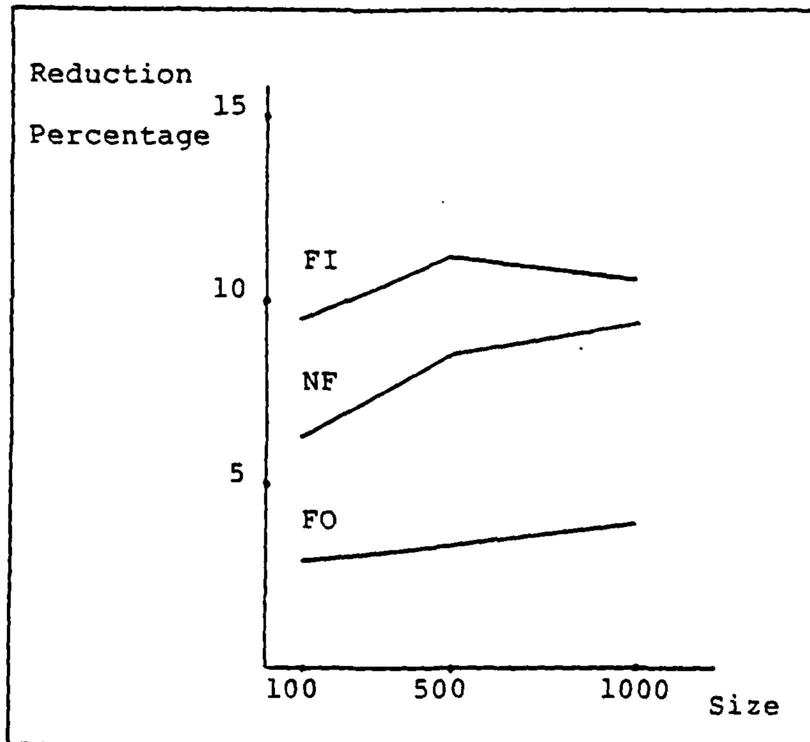


Figure 11. The Effect of Fact Ordering on Non—Monotonic RBSs

Table 4 The Effect of Fact Ordering on Non-Monotonic RBSs (Part 2)

Shape	Engine	Reduction Percentage
fan-out	1	2.925
	2	2.202
	3	2.204
	4	2.270
	5	2.315
	6	7.530
no-fan	1	6.225
	2	7.120
	3	7.225
	4	7.360
	5	7.700
	6	13.070
fan-in	1	9.490
	2	10.465
	3	10.455
	4	10.475
	5	9.860
	6	8.600

Comments. Three factors that affect the results of this experiment can be identified. The first factor is the length of the LHS of the rules; fact ordering is expected to be more efficient for RBSs with larger LHSs. The second factor is the amount of information that can be obtained about the degree of restriction of the facts; more information can be obtained if more rules fire at each run. The last factor is the degree of "association" of the facts; if facts appear together in the RHS of a rule and in the LHS of another rule, then there is high probability that they have the same degree of restriction. The first factor suggests that more reduction can be obtained from fan-in RBSs, while the second factor suggests that more information may help fact ordering for the short LHSs of fan-out RBSs. The third factor is related to

the nature of the RBS; the procedure used in the RG program implies high degree of association in order to generate consistent RBSs. In non-monotonic RBSs, more distinction among the facts can be obtained as a result of allowing the deletion of some facts from the data memory.

The results of the first part of the experiment show that the reduction percentage increases slightly as size increases, then it becomes almost constant. Shape does not affect fact ordering significantly in monotonic RBSs, while fan-in shape provides the highest reduction percentage for non-monotonic RBSs. Significant reduction is obtained for large-size fan-in and no-fan shapes. A more significant reduction is expected for actual RBSs having less degree of association than the ones generated by the RG program.

The results of the second part of the experiment show that all engines provide close reduction percentage with one exception. Engine 6, which includes a context-restricted filter, provides higher reduction percentage in both fan-out and no-fan shapes. Engine 6 tests smaller numbers of rules that have more potential to fire and thus save a great amount of unnecessary tests. Assume that the matching cost can be divided into two parts X and Y, where X represents the cost of the unnecessary tests and Y represents the cost of testing the rules that have more potential to fire. The results of the experiment shows that Y is much smaller than X ($Y < .1 X$). Fact ordering affects the X-part more than the Y-part. After ordering the facts it can be assumed that the X-part reduced

by an amount A and the Y-part is reduced by an amount B. The results show that A is greater than B; however, the ratio A/B is much less than the ratio X/Y. The reduction ratio for engine 6 will be B/Y, while for the other engines it will be (A+B)/(X+Y) . It is obvious that the reduction ratio for engine 6 would be the greatest especially for fan-out and no-fan shapes where the total matching cost, X+Y , is high.

Experiment 2: The Order of the Rules. The order of the rules in the rule memory affects the execution time when a production-order conflict-resolution policy is applied (engine 1). Matching effort can be reduced if the rules that fire more frequently are at the beginning of the rule set. The frequency of firing can be measured by the number of times each rule fires. An experiment similar to the one described previously is applied to engine 1. The frequency of rules' firing is collected in the statistics file. Another program, the rule-ordering program shown in the Appendix, sorts the rules in the rule set in descending order according to the firing frequency. Two replications of the experiment are applied in the same way described in experiment 1. This experiment is only applied to monotonic RBSs since changing the order of the rules affects the results in non-monotonic RBSs.

Results. Table 5 shows the results of the experiment for the 18 design points and Figure 12 depicts the results. The reduction percentage shown in the table is the average of the reduction ratios of the two random sets multiplied by 100.

Table 5 The Effect of Rule Ordering on Monotonic RBSs

Shape	Size	Reduction Percentage
fan-out	100	38.735
	500	43.165
	1000	46.095
no-fan	100	37.080
	500	54.795
	1000	47.855
fan-in	100	21.630
	500	23.135
	1000	25.860

Comments. Two factors that affect the results of this experiment can be identified. The first factor is the rules-firing percentage; both fan-out and no-fan RBSs have high rule-firing percentage, while fan-in RBSs have the least. For the RBSs tested in the experiment, the firing percentage for the three shapes are 10-15 for fan-out, 6-10 for no-fan, and less than 2 for fan-in. The second factor is the rules' firing frequency distribution which is problem dependent. The first factor suggests that more reduction can be obtained from fan-out RBSs.

The results show that reduction percentage, in most cases, increase slightly as size increases. Both fan-out and no-fan shapes provide high and almost equal reduction percentage, which is about twice the reduction percentage of fan-in shape.

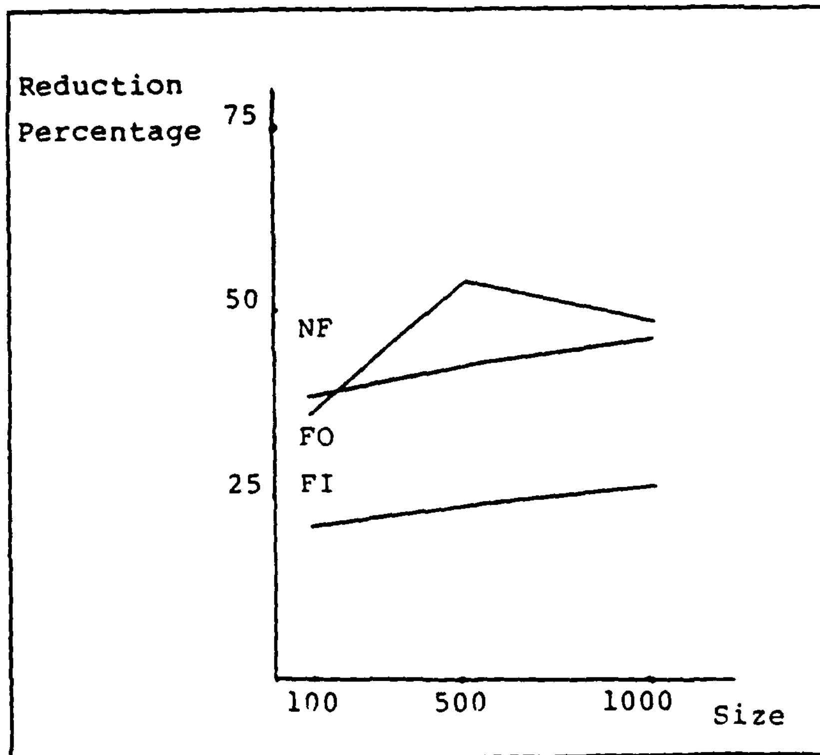


Figure 12. The Effect of Rule Ordering on Monotonic RBSs

Type 2 Experiments: Comparison of the Matching Effort for the Inference Engines.

Experiment 3: The Effect of Filtering. Two types of rule filtering are investigated in this study: filter 1, a controlled-production filter, that is added to engine 1 to produce engine 5; and filter 2, and a context-restricted filter, that is added to engine 3 to produce engine 6. The reduction ratio in the matching effort is estimated for each filter on all the design points in the factor space using both the random sets 1 and 2 for each point. Figure 13 illustrates the sequence of type 2 experiments.

Results. Tables 6 and 7 show the results of filter 1 and tables 8 and 9 show the results of filter 2. Figures 14 to 17 depict the results. The reduction percentage shown in the tables is the average of the reduction ratios of the two random sets multiplied by 100.

Table 6 The Effect of Controlled-Production Filter on Monotonic RBSs

Shape	Size	Reduction Percentage
fan-out	100	58.035
	500	76.125
	1000	81.060
no-fan	100	52.030
	500	73.475
	1000	75.715
fan-in	100	13.935
	500	23.080
	1000	25.495

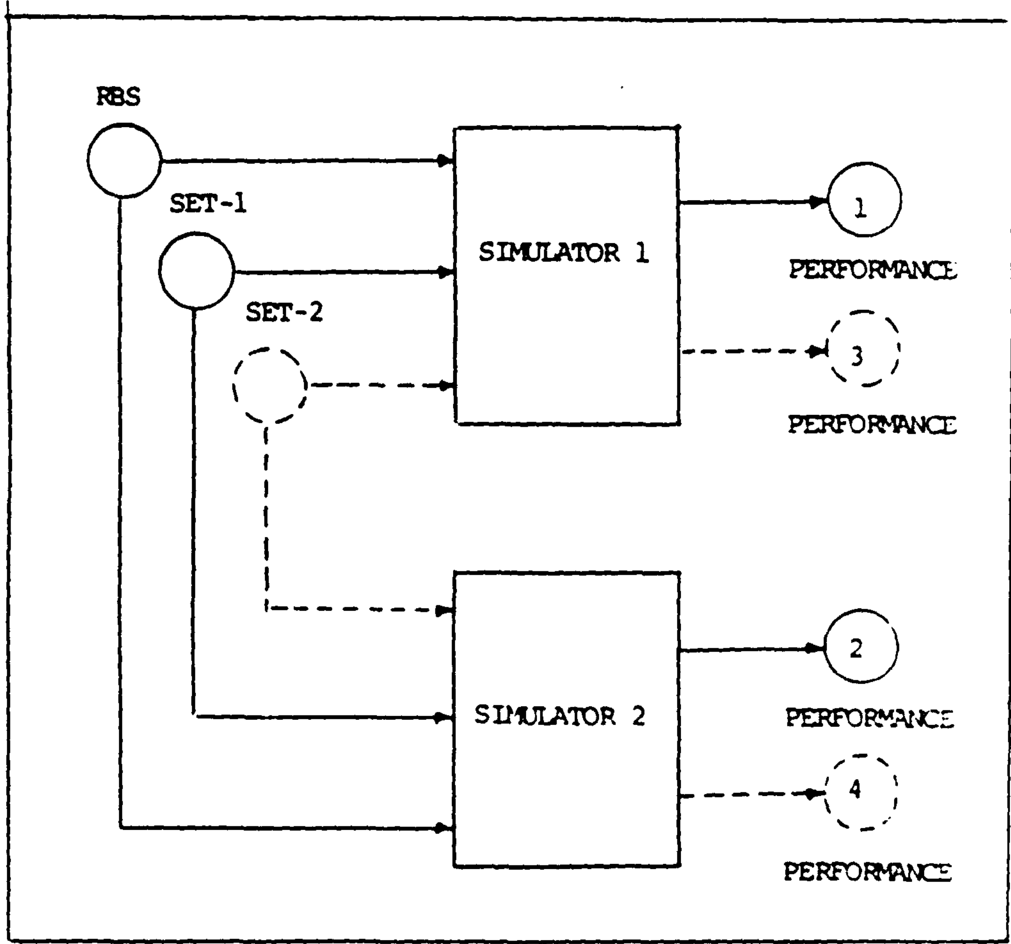


Figure 13. Type-2 Experiments

Table 7 The Effect of Controlled-Production Filter on Non-Monotonic RBSs

Shape	Size	Reduction Percentage
fan-out	100	59.240
	500	74.120
	1000	80.690
no-fan	100	49.540
	500	71.225
	1000	74.180
fan-in	100	15.415
	500	27.055
	1000	27.130

Table 8 The Effect of Context-Restricted Filter on Monotonic RBSs

Shape	Size	Reduction Percentage
fan-out	100	90.605
	500	94.450
	1000	96.460
no-fan	100	83.785
	500	91.525
	1000	93.240
fan-in	100	46.045
	500	61.300
	1000	64.665

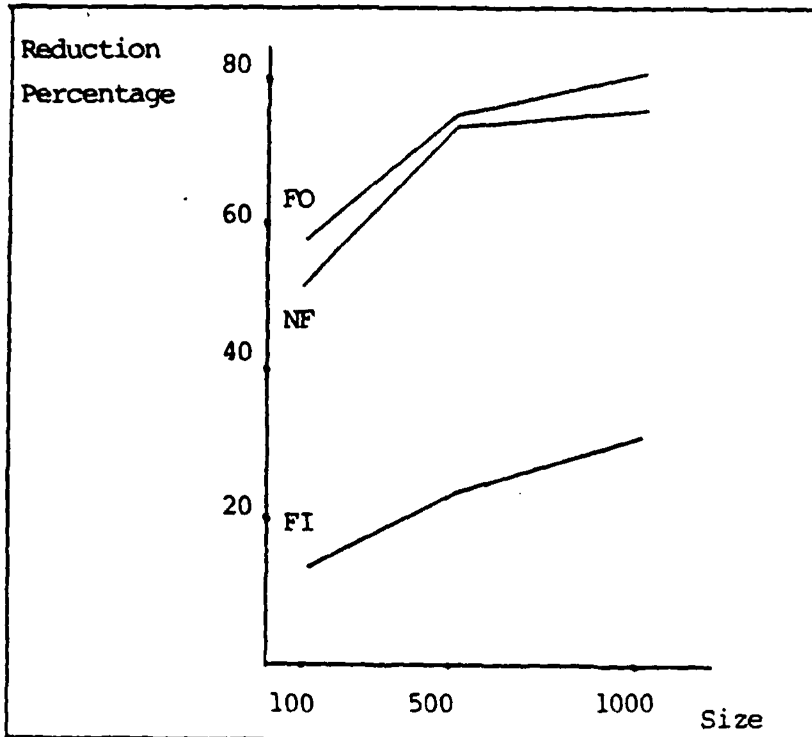


Figure 14. The Effect of Controlled-Productions Filter on Monotonic RBSs

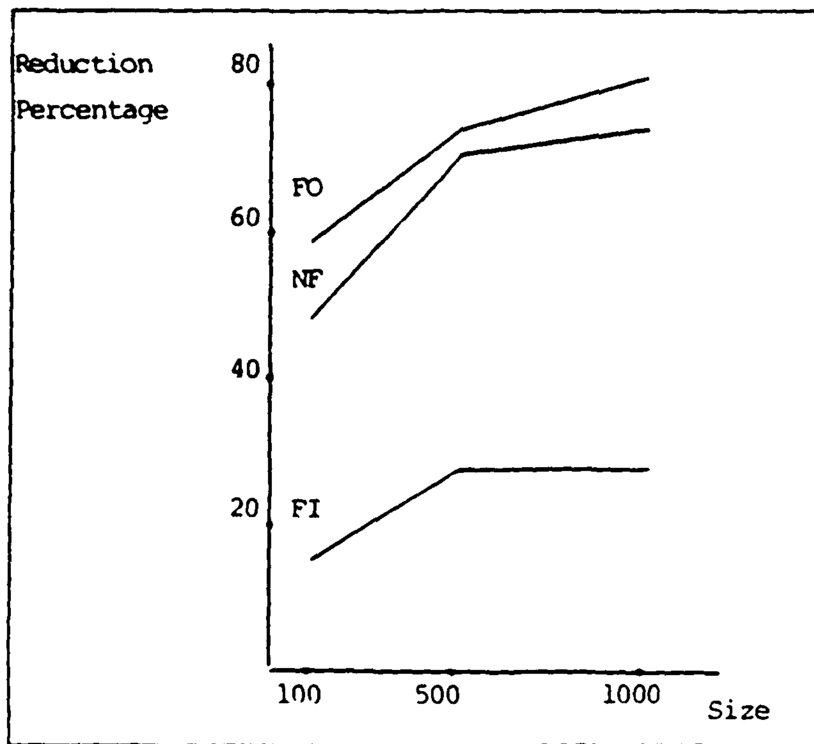


Figure 15. The Effect of Controlled-Productions Filter on Non-Monotonic RBSs

Table 9 The Effect of Context-Restricted Filter on Non-Monotonic RBSs

Shape	Size	Reduction Percentage
fan-out	100	92.710
	500	95.350
	1000	97.385
no-fan	100	85.975
	500	93.080
	1000	94.350
fan-in	100	49.885
	500	64.570
	1000	67.00

Comments. No specific factors can apparently be identified as the factors that have the main effects on the results of the experiment. However, the results show that fan-out and no-fan shapes provide better reduction percentage, which indicates that both filters work more efficiently with higher matching effort. The reduction percentage increases with the increase in size in both filters, but the increasing rate is lower for filter 2 in most cases. Monotonicity has no significant effect on the results for the first filter, while non-monotonic RBSs provide slightly higher reduction for the second filter.

Experiment 4: The Effect of Conflict-Resolution Strategy. Speed is not the only factor that decides which conflict-resolution strategy should be applied. Production-order strategy is expected to be faster than any other strategy that collects the rules eligible to fire in a

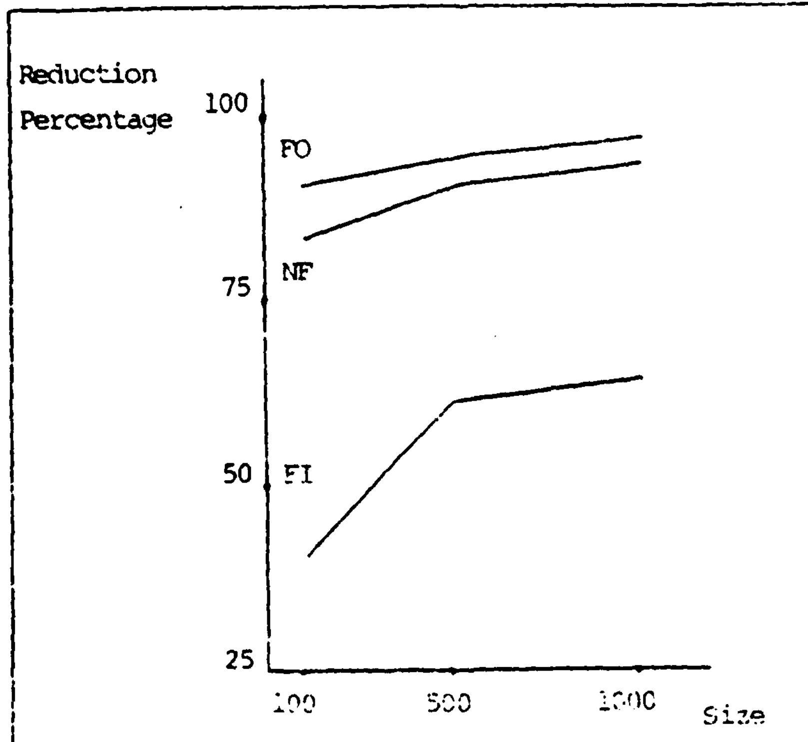


Figure 16. The Effect of Context-Restricted Filter on Monotonic RBSs

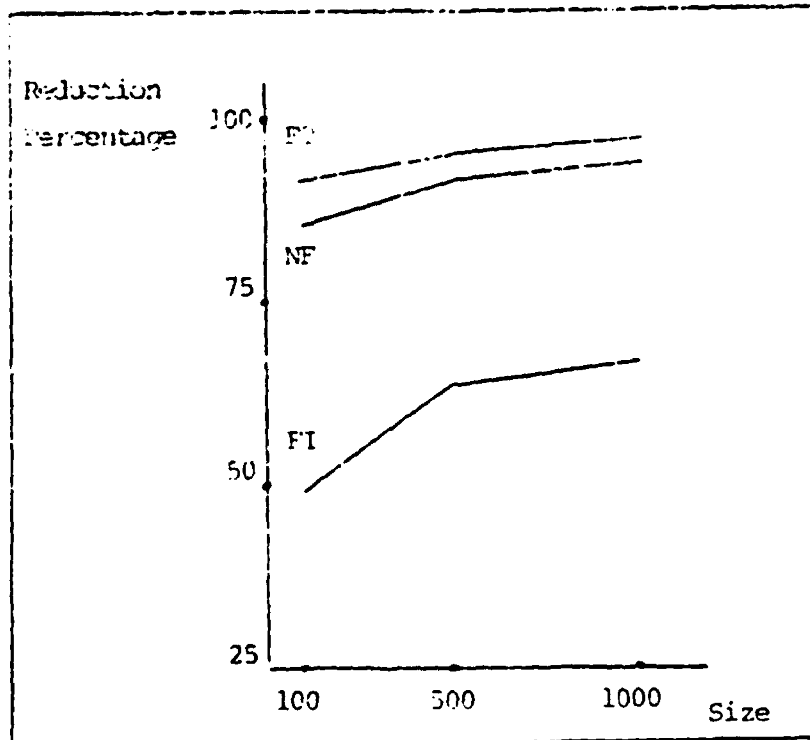


Figure 17. The Effect of Context-Resrtricted Filter on Non-Monotonic RBSs

conflict set. The purpose of this experiment is to compare the matching effort between the PO1 strategy applied in engine 1 and one of the recency strategies, viz., R1 applied in engine 3. The reduction ratio in the matching effort is estimated for each filter on all the design points in the factor space using both the random sets 1 and 2 for each point.

Results. Tables 10 and 11 shows the results of the experiment and Figures 18 and 19 depicts the results. The reduction percentage shown in the tables is the average of the reduction ratios of the two random sets multiplied by 100.

Table 10 The Effect of Conflict-Resolution Strategy on Monotonic RBSs

Shape	Size	Reduction Percentage
fan-out	100	47.425
	500	45.530
	1000	46.180
no-fan	100	45.525
	500	47.010
	1000	52.325
fan-in	100	40.085
	500	62.590
	1000	70.700

Comments. No specific factors can apparently be identified as the factors that affect the results of the experiment. However, the results show that fan-in shape provides better reduction percentage than the other two shapes. Reduction percentage increases considerably with the

Table 11 The Effect of Conflict-Resolution Strategy on Non-Monotonic RBSs

Shape	Size	Reduction Percentage
fan-out	100	49.000
	500	53.160
	1000	54.520
no-fan	100	49.675
	500	53.475
	1000	59.065
fan-in	100	41.050
	500	62.840
	1000	71.505

increase in size for fan-in shape, while it increases slightly for no-fan shape and it is almost constant for fan-out shape. Non-monotonic RBSs provide higher reduction percentage for fan-out and no-fan shapes, while the effect of monotonicity almost diminish in fan-in shape.

Conclusion

Experiments are performed on randomly generated RBSs that vary in three main factors: size, shape, and monotonicity. The rule-generator program generates highly consistent RBSs as discussed in Chapter IV. Two runs are applied in all experiments with large sample size in order to obtain accurate estimates with low variations. The relative performance of RBSs is evaluated by an independent measure of performance to avoid the effect of the environment. Although matching is not the only component of the cost of running RBSs, it does constitute the major part of it. In type 1 experiments, matching is the only factor that can be used to enhance the

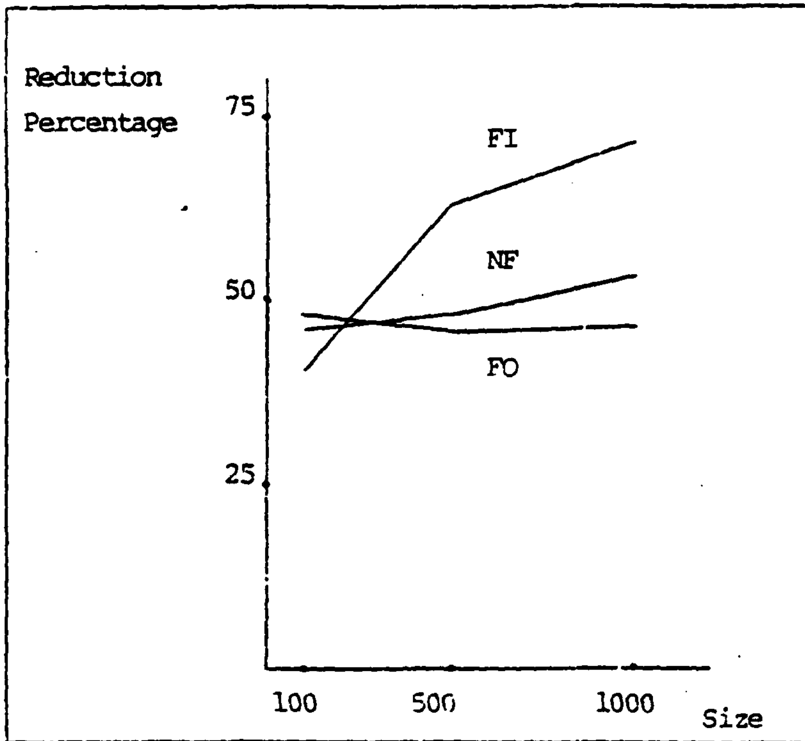


Figure 18. The Effect of Conflict-Resolution Strategy on Monotonic RBSs

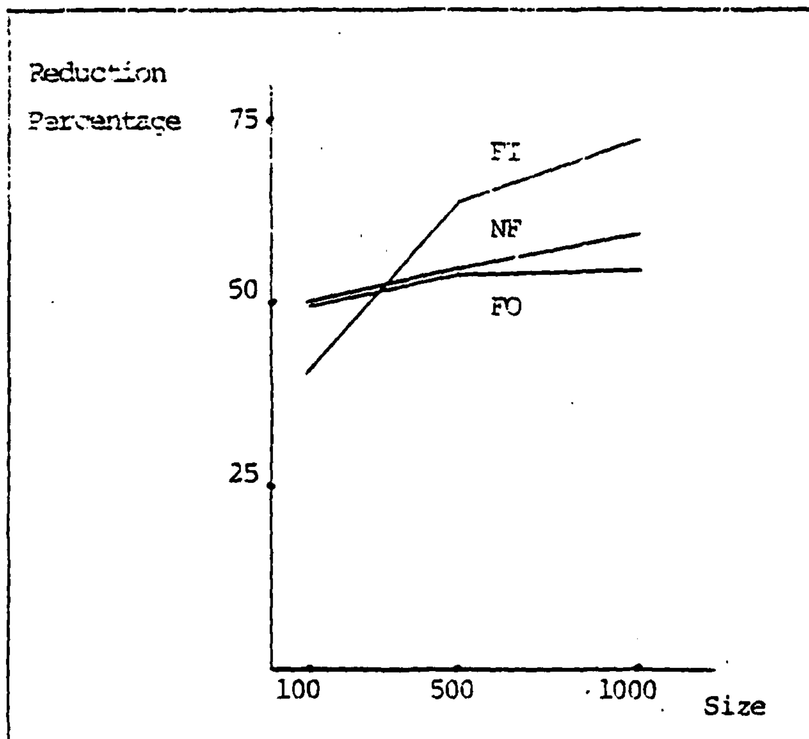


Figure 19. The Effect of Conflict-Resolution Strategy on Non-Monotonic RBSs

performance of RBSs since the other factors are constant. In type 2 experiments, other contributing factors such as the cost of filters and the cost conflict-resolution strategies are not considered in this study. The three factors analyzed in the experiments are not the only factors that may affect the performance of RBSs. Other factors such as the number of levels, the number of rules in each level, the negation percentage, the repetition percentage, and the number of initial assertions may have impact on the performance of RBSs.

VII. Summary and Recommendations

Summary

Rule-based systems (RBSs) are a problem-solving technique that is widely used in expert systems and other Artificial Intelligence applications. The speed of RBSs is one of the performance criteria often investigated in studies. This study presented a simulation approach to evaluate the most important factor of RBSs' speed, viz., the matching effort. The simulation environment is achieved by designing a numerical scheme to represent the knowledge base and defining probabilistic relations among the initial assertions to the RBSs. The numerical representation can be used to model most of the features of RBSs either directly or indirectly by resolving the structure or the function into simpler form. The components of the knowledge base are represented by an array data structure that is accessed both directly and sequentially in a simple and fast way. Six types of forward-chaining inference engines, characterized by conflict-resolution strategy and the filtering technique are simulated and investigated. In addition a rule-generator program is developed to provide a fast way to generate a wide spectrum of RBSs for experimentation. The rule-generator program is built with tight procedures to control the construction of RBSs in order to provide a high degree of consistency for the generated RBSs. The probabilistic relations among the initial assertions are used to help generate the RBSs and to generate random-assertions sets to be used in experimentation. The

numeric representation of the knowledge base, the simulated inference engines, and the rule generator, compose the simulation model used in this study to perform experiments on RBSs to evaluate their relative matching efforts. The matching efforts are measured by the number of match-tests of the rules against the data memory. This measure has the advantage of being independent of the environment, hardware or software, running the RBSs. The relative performance of different structures of RBSs is evaluated by comparing their matching efforts. All experiments were performed for two randomly-generated assertion sets of size 500 each to provide accuracy and reduce the variation in the estimate of the reduction ratio.

Experiments are designed and applied to a variety of RBSs generated by the rule generator program. Three of the factors that characterize the RBSs are analyzed, viz., size, shape and monotonicity. The six inference engines simulated in this study are used in the experiments accordingly. Two types of experiments are designed to estimate the relative performance of RBSs. The reduction ratio in matching effort is the statistic used in experiments to evaluate the RBSs; these RBSs vary in the structure of the rule set or the features of their inference engine. In the first type of experiments, a methodology to enhance the performance of a specific RBS is described and evaluated in two different experiments. In the first experiment, the facts in the LHS of the rules are rearranged according to their degree of restriction which is estimated by the number of times each

fact is added to the data memory when the RBS is run a large number of times. In the second experiment, rules are ordered according to their firing frequency, which is also estimated by running the RBS a large number of times. The fact; ordering procedure enhances the performance of RBSs in all the cases studied. The achieved savings in matching effort varied from 2% to 15%. The rule-ordering procedure is applied to monotonic RBSs that use the production order conflict-resolution strategy; the achieved savings varied from 21% to 55%.

In the second type of experiments, the effect of filtering and conflict-resolution strategies on matching effort is studied in two experiments. Two types of filters are evaluated in the first experiment. The first filter is a controlled-production filter applied to inference engines that use the production order conflict-resolution strategy; the second filter is a context-restricted filter applied to inference engines that use a recency conflict-resolution strategy. The first filter achieved from 14% to 81% saving in matching effort, while the second filter achieved from 46% to 97% saving. In the second experiment, the production order conflict-resolution strategy is compared with the recency conflict-resolution strategy. The first strategy achieved savings from 40% to 72% of the matching effort of the second strategy.

Recommendations

The current implementation of the model used in this study does not cover all features of RBSs. The model contains only the basic features of forward-chaining RBSs. Extensions to the current model can be applied in two directions: the first is to extend the model as an experimental tool, and the second is to use the model as a tool for building actual RBSs.

As an experimental tool, the direct extension to the current implementation is to apply the developed methodology to actual RBSs that can be represented suitably by the features available in the current implementation of the model. A possible extension is to use the model to compare the total effort of different structures of RBSs by estimating the cost of the other functions of inference engines beside the matching cost. In (McDermott and others, 1978: 163-165, 172-175) the authors discuss mathematical formulations for estimating the total cost. The formulations include matching cost, action cost, and filter cost. The current model can be modified to provide estimates of the parameters of these equations. Another extension is to use a more complex parameter to estimate the matching effort. In actual RBSs, data memory is not accessed directly because of the complex representation of data elements. A search is usually applied to check the existence of a certain fact in the data memory. If the search time, for a specific implementation of data memory, can be estimated as a function of data-memory size, then it can be used as a better estimate to the matching

effort.

The features represented in the model simulates the function of actual, but simple, inference engines. A simple extension is to construct a file containing the facts and their integer-number mapping and to display the facts that constitute the rules selected to fire. A more complex extension is to write code that reads rules written in natural-language form and build the rule matrix automatically, given that the rules are written in a form that can be handled directly by the model.

More complex features could be added to the model to provide representations for other features or classes of RBSs. Backward-chaining RBSs can be handled by the numeric representation, but it would require more complex data structures. Disjunctions can be represented by simple extensions to the rule matrix and the algorithms that simulate the inference engines, but the rule generator and the rearrangement of the LHS of the rules will be more complicated. Variables can be represented if they can assume only a limited number of values. One way is to give the facts that represent variables a range of values and to extend the rule status flag in the rule matrix into a vector that keeps track of the instantiations of the rule. In addition, major changes to the algorithms should be applied to simulate the binding process. Uncertainty schemes are usually applied to backward-chaining RBSs; however, some forward-chaining tools, EXPERT for example, provide uncertainty schemes. The numeric representation can handle uncertainty schemes with simple

extensions to the current representation and algorithms.

In general, adding more features will limit the use of the model in experimentation because of the difficulty in generating consistent RBSs. Specific RBSs will need to be designed in order to run the experiments. However, more specific structures can be tested and more accurate performance measures can be obtained. Adding more features will enhance the performance of the numeric representation as a RBSs' building tool that can be more efficient than other building tools but in limited and simple applications.

Bibliography

Aiello, Nelleko. "A Comparative Study of Control Strategies For Expert Systems: AGE Implementation of Three Variations of PUFF," The National Conference on Artificial Intelligence. 1-4. Washington, D.C: IAAA,1983.

Bratko, Ivan. PROLOG Programming For Artificial Intelligence. Wokingham, England: Adison-Wesley Publishing Company, 1987.

Brownston, Lee and others. Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming. Reading, Massachusetts: Adison-Wesley Publishing Company, 1985.

Forgy, C. L. and Susan J. Shepard. "Rete: a Fast Match Algorithm," AI Expert, 34-40 (January 1987).

Hayes-Roth, Frederick. "Rule-Based Systems," Communications of The ACM, 28: 921-932 (September 1985).

McDermott, J and C. Forgy. "Production System Conflict-Resolution Strategies," Pattern-Directed Inference Systems. Edited by D. A. Waterman and F. Hayes-Roth. New York: Academic Press, 1978.

McDermott, J and others. "The Efficiency of Certain Production System Implementations," Pattern-Directed Inference Systems. Edited by D. A. Waterman and F. Hayes-Roth. New York: Academic Press, 1978.

Neapolitan, Richard E. "Forward-Chaining Versus a Graph Approach as The Inference Engine in Expert Systems," SPIE, Applications of Artificial Intelligence III, 635: 62-69 (1986).

Nguyen, Tin A. "Verifying Consistency of Production Systems," IEEE Forth Conference on Artificial Intelligence Applications, 4-8. Kissimee, Florida (1987).

Politakis, Peter G. Empirical Analysis For Expert Systems. Boston: Pitman Advanced Publishing Program, 1985.

Rich, Elaine. Artificial Intelligence. New York: McGraw-Hill Book Company, 1983.

Whiting, K. W. and others. "SUBEX: A Focus of Attention Technique For Rule-Based Inference," IEEE Second Conference on Artificial Intelligence Applications, 215-220. Miami-Beach, Florida (1985).

Winston, Patrick Henry. Artificial Intelligence. Reading, Massachusetts: Adison-Wesley Publishing Company, 1984.

Appendix:The Computer Programs

Contents

	Page
List of Variables	114
The Conditional-Scheme Program	118
The Independent-Scheme Program	120
The Grouped-Scheme Program	122
The Rule-Generator Program	124
Sample 1: Fan-Out Random-Assertions Set	129
Sample 2: Fan-Out Monotonic RBS	130
Sample 3: Fan-Out Non-Monotonic RBS	131
Sample 4: No-Fan Random-Assertions Set	132
Sample 5: No-Fan Monotonic RBS	133
Sample 6: No-Fan Non-Monotonic RBS	134
Sample 7: Fan-In Random-Assertions Set	135
Sample 8: Fan-In Monotonic RBS	136
Sample 9: Fan-In Non-Monotonic RBS	137
Engine 1 Simulator (Version 2)	138
Engine 2 Simulator (Version 2)	141
Engine 3 Simulator (Version 2)	146
Engine 4 Simulator (Version 2)	147
Engine 5 Simulator (Version 2)	148
Engine 6 Simulator (Version 2)	150
Fact-Ordering Program	154
Rule-Ordering Program	157

List of Variables

This part of the Appendix contains a list of the variables used in all programs. The variables are classified as: two-dimensional arrays, one-dimensional array, and variables. The variables in each category is written in alphabetical order.

Two-Dimensional arrays.

ASR: the initial assertions.

ASRT: the random assertions.

APP: the probabilistic relation among the initial assertions.

INDX: the condition-membership index.

MG: groups' specifications for a grouped-probability scheme.

MR: a rule matrix sorted according to the rule-firing frequency.

NC: the conflict set.

NCSRT: the sorted conflict set.

NF: the fact matrix.

NR: the rule matrix.

One-Dimensional Arrays.

FIRE: rules-firing frequency.

GRP: the order of the last rule in each group of rules.

LEX: the lexical numbers for the rules in the conflict set.

LVL: the number of rules in each level.

LVE: the order of the first rule in each level.

LVS: the starting location, dedicated to each level, in a random-assertions vector.

REC: a list of facts in ascending order according to their recency.

S1: the LHS of a rule.

S2: fact-assertion frequency for the LHS of a rule.

SFIRE: rules-firing frequency sorted in descending order.

TRU: the fact-assertion frequency.

Variables.

ARF: the average number of the rules fired in an experiment.

ATC: the average number of the match-tests in an experiment.

IASS: the number of true assertions for all levels.

IPOS: a flag which is set when the LHS of a rule has at least one positive fact.

IREC: a counter contains the location of the most-recent fact in the REC array.

IX: the number of assertions selected from a group.

IY: the starting location in a random-assertion vector for a group of assertions.

JRU: the order of the rule selected from the preceding level to build the LHS of the current rule.

KI: the most-recent fact.

KRU: a flag which is set when the length of the LHS of the current rule is greater than the RHS of the rule selected from the preceding level.

LA: the last action in the current level.

LB: the last action in the preceding level.

LEND: the location of the last condition in the LHS of the current rule.

LG: the number of assertion groups.

LRF: the number of rules fired at least once in an experiment.

LRU: the number of conditions in the LHS taken from the RHS of

a rule in the preceding level.

LSTRT: the starting location in the LHS of a rule for conditions taken from initial assertions.

MAXL: upper bound for the LHS.

MAXR: upper bound for the RHS.

MINL: lower bound for the LHS.

MINR: lower bound for the RHS.

MLV: the order of the first rule in the current level.

MP: the number of probability branches.

MRU: a randomly-selected assertion vector to build the rest of the LHS of a rule.

NA: a counter contains the last fact added to the RBS.

NA1: the number of assertions for the first level.

NA2: the number of assertions for the other levels.

NASS: the number of assertions that can be used to build the LHS of the current rule.

NC: the last action that can be repeated.

NG: the number of groups.

NPT: a pointer to the most-recent fact.

NRF: the number of the rules fired in a run.

NRS: the order of the last rule in a group of rules.

NRTF: the ID-number of the rule selected to fire.

NTA1: the number of true assertions for the first level.

NTA2: the number of true assertions for the other levels.

NFAC: the number of facts.

NFIRE: location of the rule status flag.

NLVL: the number of levels.

NRULE: the number of rules.

NRUN: the number of runs.

PD: deletion probability.

PN: negation probability.

PR: repetition probability.

TC: a counter contains the number of match-tests.

```
C---------------- CONDITIONAL-SCHEME PROGRAM -----------------
C
C      INPUT FILES
C      -----------
C      CONDITIONAL-SCHEME DATA FILE
C
C      OUTPUT FILES
C      ------------
C      RANDOM-ASSERTIONS SET
C
C      BOUNDARIES
C      ----------
C      10 LEVELS
C      50 BRANCHED ASSERTIONS
C      5 PROBABILITY BRANCHES
C      1000 RUNS
C
C      PURPOSE
C      -------
C      GENERATE A RANDOM-ASSERTIONS SET FROM A CONDITIONAL SCHEME
C
C---------------------------------------------------------------
C
      INTEGER ASR(0:50,5),ASRT(1000,50)
      REAL APP(0:50,5)
      OPEN (UNIT=9,FILE='PROB.DAT',STATUS='OLD')
      OPEN (UNIT=10,FILE='ASRT.DAT',STATUS='NEW',RECL=132)
C-
C-ENTER INITIAL DATA
C
      PRINT*,'# OF LEVELS'
      READ*,NLVL
      PRINT*,'# OF PROBABILITY BRANCHES'
      READ*,MP
      PRINT*,'# OF RUNS'
      READ*,NRUN
      PRINT*,'# OF BRANCHED ASSERTIONS FOR LEVEL-1'
      READ*,MASS
      PRINT*,'# OF LEVEL-1 TRUE ASSERTIONS '
      READ*,NTA1
      IASS=NTA1
C
      PRINT*,'RANDOM NUMBER STREAM (1--3)'
      READ*,NRS
      IF (NRS.EQ.1)II=11234521
      IF (NRS.EQ.2)II=428956419
      IF (NRS.EQ.3)II=200496737
C
C-     ESTABLISH THE ASSERTIONS AND THE PROBABILTY MATRICES
C-
      DO 10 I=0,MASS
         READ(9,*)(ASR(I,J),J=1,MP)
         READ(9,*)(APP(I,J),J=1,MP)
10    CONTINUEC
      DO 11 KK=1,NRUN
```

```
          I=0
C
C-BUILD A RANDOM-ASSERTION VECTOR
C
          DO 22 K=1,IASS
             XX=RAN(II)
             DO 33 J=1,MP
               IF (XX .LT. APP(I,J)) GOTO 44
33           CONTINUE
44           ASRT(KK,K)=ASR(I,J)
             I=ASR(I,J)
22        CONTINUE
11     CONTINUE
C
C-WRITE THE RANDOM-ASSERTIONS SET
C
       WRITE(10,*)NRUN,IASS
       DO 55 I=1,NRUN
          WRITE(10,*)(ASRT(I,J),J=1,IASS)
55     CONTINUE
C
       STOP
       END
C
```

```
C----------------- INDEPENDENT-SCHEME PROGRAM ----------------
C
C     INPUT FILES
C     -----------
C     INDEPENDENT-SCHEME DATA FILE
C
C     OUTPUT FILES
C     ------------
C     RANDOM-ASSERTIONS SET
C
C     BOUNDARIES
C     ----------
C     10 LEVELS
C     50 BRANCHED ASSERTIONS
C     5 PROBABILITY BRANCHES
C     1000 RUNS
C
C     PURPOSE
C     -------
C     GENERATE A RANDOM-ASSERTIONS SET FROM AN INDEPENDENT-SCHEME
C
C-------------------------------------------------------------
C
      INTEGER ASR(50,5),ASRT(1000,50)
      REAL APP(50,5)
      OPEN (UNIT=9,FILE='PROBI.DAT',STATUS='OLD')
      OPEN (UNIT=10,FILE='ASRT.DAT',STATUS='NEW',RECL=132)
C
C-ENTER INITIAL DATA
C
      PRINT*,'# OF LEVELS'
      READ*,NLVL
      PRINT*,'# OF PROBABILITY BRANCHES'
      READ*,MP
      PRINT*,'# OF RUNS'
      READ*,NRUN
      PRINT*,'# OF LEVEL-1 TRUE ASSERTIONS'
      READ*,NTA1
      PRINT*,'# OF TRUE ASSERTIONS FOR THE OTHER LEVELS'
      READ*,NTA2
      IASS=NTA1+(NLVL-1)*NTA2
C
      PRINT*,'RANDOM NUMBER STREAM (1--3)'
      READ*,NRS
      IF (NRS.EQ.1)II=11234521
      IF (NRS.EQ.2)II=428956419
      IF (NRS.EQ.3)II=200496737
C
C-ESTABLISH THE ASSERTION AND PROBABILITY MATRICES
C
      DO 10 I=1,IASS
         READ(9,*)(ASR(I,J),J=1,MP)
         READ(9,*)(APP(I,J),J=1,MP)
10    CONTINUEC
      DO 11 KK=1,NRUN
```

```
C
C-BUILD A RANDOM-ASSERTION VECTOR
C
            DO 22 I=1,IASS
               XX=RAN(II)
               DO 33 J=1,MP
                  IF (XX .LT. APP(I,J)) GOTO 44
33             CONTINUE
44             ASRT(KK,I)=ASR(I,J)
22           CONTINUE
11      CONTINUE
C
C-WRITE THE RANDOM-ASSERTIONS SET
C
        WRITE(10,*)NRUN,IASS
        DO 55 I=1,NRUN
           WRITE(10,*)(ASRT(I,J),J=1,IASS)
55      CONTINUE
C
        STOP
        END
C
```

```
C------------------- GROUPED-SCHEME PROGRAM -------------------
C
C     INPUT FILES
C     -----------
C     GROUPED-SCHEME DATA FILE
C
C     OUTPUT FILES
C     ------------
C     RANDOM-ASSERTIONS SET
C
C     BOUNDARIES
C     ----------
C     10 LEVELS
C     10 GROUPS
C     50 GROUPED ASSERTIONS
C     1000 RUNS
C
C     PURPOSE
C     -------
C     GENERATE A RANDOM-ASSERTIONS SET FROM A GROUPED-SCHEME
C
C----------------------------------------------------------------
C
      INTEGER MG(10,3),ASRT(1000,50)
      OPEN (UNIT=9,FILE='PROBG.DAT',STATUS='OLD')
      OPEN (UNIT=10,FILE='ASRT.DAT',STATUS='NEW',RECL=132)
C
C-ENTER INITIAL DATA
C
      PRINT*,'# OF LEVELS'
      READ*,NLVL
      PRINT*,'# OF ASSERTIONS GROUPS'
      READ*,LG
      PRINT*,'# OF RUNS'
      READ*,NRUN
      PRINT*,'# OF LEVEL-1 TRUE ASSERTIONS'
      READ*,NTA1
      PRINT*,'# OF TRUE ASSERTIONS FOR THE OTHER LEVELS'
      READ*,NTA2
      IASS=NTA1+(NLVL-1)*NTA2
C
      PRINT*,'RANDOM NUMBER STREAM (1--3)'
      READ*,NRS
      IF (NRS.EQ.1)II=11234521
      IF (NRS.EQ.2)II=428956419
      IF (NRS.EQ.3)II=200496737
C
C-READ THE GROUPED ASSERTIONS MATRIX
C
      DO 5 I=1,LG
         READ(9,*)(MG(I,J),J=1,3)
5     CONTINUE
C
      DO 11 KK=1,NRUN
         IX=0
```

```
         IY=0
         DO 22 K=1,LG
C
C-DETERMINE GROUP BOUNDARIES IN THE RANDOM-ASSERTION VECTOR
C
           IY=IY+IX
           IX=MG(K,3)
           ASRT(KK,IY+1)=MG(K,1)+INT(RAN(II)*(MG(K,2)-MG(K,1)+1))

           DO 33 I=IY+2,IY+IX
C
C-BUILD THE PART OF THE ASSERTION VECTOR TAKEN FROM A GROUP
C
34               NN=MG(K,1)+INT(RAN(II)*(MG(K,2)-MG(K,1)+1))
                 IW=IW+1
                 DO 44 L=1,I-1
                    IF (NN.EQ.ASRT(KK,L) .AND. IW.LE.10) GOTO 34
44               CONTINUE
                 IW=0
                 ASRT(KK,I)=NN
33            CONTINUE
22         CONTINUE
11      CONTINUE
C
C-WRITE THE RANDOM-ASSERTIONS SET
C
      WRITE(10,*)NRUN,IASS
      DO 55 I=1,NRUN
         WRITE(10,*)(ASRT(I,J),J=1,IASS)
55    CONTINUE
C
      STOP
      END
C
```

```
C---------------------- THE RULE GENERATOR --------------------
C
C      INPUT FILES
C      -----------
C      RBS SPECIFICATIONS
C      RANDOM-ASSERTIONS SET
C
C      OUTPUT FILES
C      ------------
C      THE GENERATED RBS
C
C      BOUNDARIES
C      ----------
C      1000 RULE, 4000 FACT, 8 CONDITIONS+ACTIONS
C      10 LEVELS
C      200 RANDOM ASSERTIONS
C
C      PURPOSE
C      -------
C      GENERATING RULE-BASED SYSTEMS
C
C
--------------------------------------------------------------
C
      INTEGER NR(1000,12),ASRT(200,50),LVL(0:10),LVS(10),LVE(0:10)
      OPEN (UNIT=10,FILE='RGEN.DAT',STATUS='NEW',RECL=132)
      OPEN (UNIT=9,FILE='RSPC.DAT',STATUS='OLD')
      OPEN (UNIT=8,FILE='ASRT.DAT',STATUS='OLD',RECL=132)
C
C-RANDOM STREAM SEED
C
      II=11234521
C
C-READ RBS SPECIFICATIONS
C
      READ(9,*)NRULE,NA1,NTA1,NA2,NTA2
      READ(9,*)NLVL
      LVL(0)=0
      LVE(0)=0
      DO 105 I=1,NLVL
         READ(9,*),LVL(I)
         LVE(I)=LVL(I)+LVE(I-1)
105   CONTINUE
      READ(9,*)MINL,MAXL,MINR,MAXR,PN,PR,PD
C
C-THE BOUNDARIES OF LEVELS IN A RANDOM-ASSERTION VECTOR
C
      LVS(1)=1
      LVS(2)=NTA1+1
      DO 115 I=3,NLVL
         LVS(I)=LVS(I-1)+NTA2
115   CONTINUE
C

C-INITIALIZE THE RULE MATRIX
```

```
C
      NFIRE=4+MAXL+MAXR
      NFAC=0
      IL=MAXL-MINL+1
      IR=MAXR-MINR+1
      DO 10 I=1,NRULE
         NR(I,1)=I
         NR(I,2)=MINL+INT(RAN(II)*IL)
         NR(I,3)=MINR+INT(RAN(II)*IR)
         NR(I,NFIRE)=0
10    CONTINUE
C
C-STORE THE RANDOM-ASSERTION SET IN A MATRIX
C
      READ(8,*)NRUN,IASS
      DO 120 I=1,NRUN
         READ(8,*)(ASRT(I,J),J=1,IASS)
120   CONTINUE
      WRITE(10,*)NRULE,MAXL,MAXR,NFIRE
C
C-INITIALIZE THE BOUNDARIES OF THE FIRST LEVEL
C
      LA=0
      NA=NA1+(NLVL-1)*NA2
      MLV=1
C
C- BUILD THE RBS
C- *************
C
      DO 99 K=1,NLVL
C
C-INITIALIZE THE PARAMETERS OF THE LEVEL
C
      LB=NA
      IF (K.EQ.1)THEN
         NASS=NTA1
      ELSE
         NASS=NTA2
      ENDIF
C
C-    RULES OF A LEVEL
C-    *****************
C
      DO 88 I=MLV,LVE(K)
C-          THE LEFT-HAND SIDE
C-          ******************
C
C-FIRST LEVEL PARAMETERS
C
      IF (K .EQ. 1) THEN
         LRU=0
         LSTRT=4
         GOTO 12
      ENDIFC
C-CONDITIONS FROM THE PRECEDING LEVEL
```

```
C
895    LPOS=0
       JRU=INT(RAN(II)*LVL(K-1))+LVE(K-2)+1
c   the selected rule should has at least one positive fact
c
        DO 896 J=4+MAXL,3+MAXL+NR(JRU,3)
           IF (NR(JRU,J).GT.0)LPOS=1
896     CONTINUE
       IF (LPOS .EQ. 0)GOTO 895
       IF (NR(JRU,3) .GE. NR(I,2)) THEN
          LRU=NR(I,2)
          KRU=0
       ELSE
          LRU=NR(JRU,3)
          KRU=1
       ENDIF
       IF (NR(JRU,3).GT.NR(I,2)) GOTO 15
C
C-LHS >= RHS
C   copy the RHS
c
       DO 11 J=1,LRU
          NR(I,J+3)=NR(JRU,MAXL+3+J)
11     CONTINUE
       GOTO 14
C
C-LHS < RHS
C
15     NN=RAN(II)*NR(JRU,3)+1
       IW=IW+1
       MM=NR(JRU,MAXL+3+NN)
c   the first selected fact should be positive
c
       IF (MM.LT.O .AND. IW.LE.50) GOTO 15
       NR(I,4)=MM
       IW=0
       DO 9 J=5,LRU+3
16     NN=RAN(II)*NR(JRU,3)+1
       NR(I,J)=NR(JRU,MAXL+3+NN)
       IW=IW+1
       DO 8 IC=4,J-1
       IF (NR(I,J).EQ.NR(I,IC).AND.IW.LE.10)GOTO 16
8      CONTINUE
       IW=0
9      CONTINUE
C
C-LHS COMPLETED
C
14     IF (KRU .EQ. 0) GOTO 62
C


C-CONDITIONS FROM INITIAL ASSERTIONS
C
       LSTRT=4+LRU
```

```
12     NR(I,LSTRT)=1+INT(RAN(II)*NASS)
c    first fact can be negated except for the first-level rules
c
       IF (RAN(II).LT.PN .AND. K.GT.1) NR(I,LSTRT)=-1*NR(I,LSTRT)
       LEND=NR(I,2)+3
       IF (LEND .EQ. LSTRT) GOTO 139
       DO 77 J=LSTRT+1,LEND
60        NN=1+INT(RAN(II)*NASS)
          IW=IW+1
c     no repeated facts in the same rule
c
          DO 66 KK=LSTRT,J-1
             IF (NN.EQ.ABS(NR(I,KK)) .AND. IW.LE.10) GOTO 60
66        CONTINUE
          IW=0
          NR(I,J)=NN
          XX=RAN(II)
          IF (XX.LT.PN ) NR(I,J)=-1*NN
77     CONTINUE
C
C-MAPPING FROM ASSERTIONS
C
139    MRU=INT(RAN(II)*NRUN)+1
       DO 138 J=LSTRT,LEND
          NN=ABS(NR(I,J))
          NR(I,J)=(NR(I,J)/NN)*(ASRT(MRU,NN+LVS(K)-1))
138    CONTINUE
C
C            THE RIGHT-HAND SIDE
C            *******************
C
62     NC=NA
       DO 55 JJ=MAXL+4,MAXL+3+NR(I,3)
          XX=RAN(II)
C
C-REPETITION
C      generate few rules without repetition
c
          IF (I .GT.MLV+5-MAXR  .AND. XX .LT. PR) GOTO 91
          NA=NA+1
          NR(I,JJ)=NA
          GOTO 55
c      repeat only from the preceding rules in the level
c
91        NR(I,JJ)=(1+LB+INT(RAN(II)*(NC-LB)))
C
C- DELETION
C
          IF (RAN(II).LT.PD)NR(I,JJ)=-1*NR(I,JJ)
          IW=IW+1
c      no repeated facts in the same rule
c          DO 87 IL=MAXL+4,JJ-1
            IF (NA-LB .EQ. 0 )GOTO 55
            IF(ABS(NR(I,JJ)).EQ.ABS(NR(I,IL)).AND.IW.LE.10)GOTO 91
87          CONTINUE
```

```
            IW=0
55     CONTINUE
C
C-A RULE COMPLETED
C
       WRITE(10,*)(NR(I,IX),IX=1,NFIRE)
88     CONTINUE
C
C- A LEVEL COMPLETED
C
       MLV=1+LVE(K)
       LA=LB
99     CONTINUE
C
C-RBS COMPLETED
C
       NFAC=NA
       WRITE(10,*)NFAC
       IF(NFAC.GT.4000)PRINT*,'WARNING: FACTS EXCEEDED BOUNDARY'
       STOP
       END
C
```

Sample 1: Fan-Out Random-Assertions Set

3	6	11
1	7	9
3	7	9
4	7	11
4	8	11
3	5	10
1	8	10
3	6	12
4	8	10
3	5	9
1	7	10
4	7	10
2	6	12
4	8	11
2	7	10
2	6	9
1	7	10
2	7	11
4	7	11
4	8	11
2	8	12
2	5	12
3	5	12
3	8	10
4	6	11
3	5	9
1	6	12
2	8	11
4	5	11
4	6	9

NUMBAR OF RUNS = 30

NUMBER OF TRUE ASSERTIONS FOR THE FIRST LEVEL = 3

Sample 2: Fan-Out Monotonic RBS

RULE NUMBER	MAX LHS	MAX RHS	LHS		RHS				RULE STATUS
1	2	3	9	6	13	14	15	0	0
2	2	3	3	5	16	17	18	0	0
3	2	3	5	12	18	19	13	0	0
4	2	4	6	11	17	14	19	20	0
5	2	4	12	1	14	19	21	22	0
6	2	4	10	7	16	23	21	24	0
7	2	4	12	3	25	26	27	22	0
8	2	4	8	4	23	27	28	29	0
9	2	3	13	15	30	31	32	0	0
10	2	4	22	21	33	34	35	36	0
11	2	4	29	28	36	37	31	38	0
12	2	4	13	18	39	31	40	41	0
13	2	4	20	17	42	32	43	38	0
14	2	4	29	28	38	44	45	46	0
15	2	3	32	38	47	48	49	0	0
16	2	3	40	39	50	51	52	0	0
17	2	4	31	39	53	49	48	51	0
18	2	3	46	45	52	54	55	0	0
19	2	3	33	35	50	56	57	0	0
20	2	4	45	38	57	50	58	59	0

SHAPE: FAN-OUT
MONOTONICITY: MONOTONIC
NUMBER OF RULES = 20
NUMBER OF LEVELS = 3

Sample 3: Fan-Out Non-Monotonic RBS

RULE NUMBER	MAX LHS	MAX RHS	LHS		RHS				RULE STATUS
1	2	3	9	6	13	14	15	0	0
2	2	3	3	5	16	17	18	0	0
3	2	3	5	12	18	19	13	0	0
4	2	4	6	11	17	-14	-19	20	0
5	2	4	12	1	-14	19	21	22	0
6	2	4	10	7	16	23	21	24	0
7	2	4	12	3	25	26	27	-22	0
8	2	4	8	4	-23	27	28	29	0
9	2	3	13	15	30	31	32	0	0
10	2	4	22	21	33	34	35	36	0
11	2	4	29	28	-36	37	-31	38	0
12	2	4	13	18	39	31	40	41	0
13	2	4	20	17	42	-32	43	38	0
14	2	4	29	28	-38	44	45	46	0
15	2	3	38	-32	47	48	49	0	0
16	2	3	39	40	50	51	52	0	0
17	2	4	31	30	-51	-47	53	52	0
18	2	3	31	41	54	55	56	0	0
19	2	3	33	35	-51	57	58	0	0
20	2	4	45	-38	58	51	59	60	0

SHAPE: FAN-OUT
MONOTONICITY: NON-MONOTONIC
NUMBER OF RULES = 20
NUMBER OF LEVELS = 3

Sample 4: No-Fan Random-Assertions Set

3	6	11	13	19	21
3	7	9	16	19	23
4	8	11	15	18	22
1	8	10	14	18	24
4	8	10	14	18	21
1	7	10	16	20	22
2	6	12	15	20	22
2	7	10	13	18	21
1	7	10	13	19	23
4	7	11	16	20	23
2	8	12	14	17	23
3	5	12	14	20	22
4	6	11	15	17	21
1	6	12	13	20	23
4	5	11	16	18	21
1	5	12	14	19	22
4	7	10	14	18	22
4	8	9	16	20	23
3	5	11	13	19	22
3	5	11	13	18	22
2	6	11	14	18	22
2	6	12	14	20	23
1	5	12	14	18	22
3	8	11	16	19	22
4	8	12	14	18	24
1	8	9	13	17	23
2	8	9	14	18	21
4	8	11	13	20	21
4	6	9	13	20	21
4	7	12	16	20	22

NUMBER OF RUNS = 30

NUMBER OF TRUE ASSERTIONS FOR THE FIRST LEVEL = 4

Sample 5: No-Fan Monotonic RBS

RULE NUMBER	MAX LHS	MAX RHS	LHS			RHS			RULE STATUS
1	3	2	14	12	5	25	26	0	0
2	3	2	11	5	16	27	28	0	0
3	3	2	14	9	-8	29	30	0	0
4	3	3	13	10	7	31	32	26	0
5	2	3	6	13	0	33	34	30	0
6	3	3	5	-1	14	35	36	37	0
7	3	3	8	9	-14	38	39	40	0
8	3	3	3	11	5	37	34	35	0
9	2	2	26	31	0	41	42	0	0
10	2	3	34	30	0	43	44	45	0
11	2	3	32	26	0	46	47	48	0
12	2	3	29	30	0	49	50	51	0
13	3	3	33	34	30	51	52	42	0
14	2	3	36	37	0	45	52	53	0
15	2	2	48	46	0	54	55	0	0
16	2	2	43	45	0	56	57	0	0
17	2	3	48	47	0	58	59	60	0
18	3	2	49	50	51	61	62	0	0
19	2	2	51	42	0	63	64	0	0
20	3	3	51	52	42	65	66	55	0

SHAPE: NO-FAN
MONOTONICITY: MONOTONIC
NUMBER OF RULES = 20
NUMBER OF LEVELS = 3

Sample 6: No-Fan Non-Monotonic RBS

RULE NUMBER	MAX LHS	MAX RHS	LHS			RHS			RULE STATUS
1	3	2	14	12	5	25	26	0	0
2	3	2	11	5	16	27	28	0	0
3	3	2	14	9	-8	29	30	0	0
4	3	3	13	10	7	31	32	-26	0
5	2	3	6	13	0	33	34	30	0
6	3	3	5	-1	14	35	36	37	0
7	3	3	8	9	-14	38	39	40	0
8	3	3	3	11	5	-37	-34	35	0
9	2	2	31	-26	0	41	42	0	0
10	2	3	38	40	0	43	44	45	0
11	2	3	30	34	0	46	47	48	0
12	2	3	35	-34	0	-48	49	-42	0
13	3	3	35	36	37	-47	50	42	0
14	2	3	35	-37	0	51	-43	52	0
15	2	2	45	44	0	53	54	0	0
16	2	2	51	-43	0	55	56	0	0
17	2	3	49	-42	0	57	58	59	0
18	3	2	-47	50	42	60	61	0	0
19	2	2	49	-42	0	62	54	0	0
20	3	3	41	42	22	-53	63	61	0

SHAPE: NO-FAN
MONOTONICITY: NON-MONOTONIC
NUMBER OF RULES = 20
NUMBER OF LEVELS = 3

Sample 7: Fan-In Random Assertions Set

3	6	11	13	19	21	27	31	33
4	7	11	15	20	23	28	29	34
1	8	10	14	18	24	28	32	34
3	5	9	13	19	22	28	31	34
2	6	12	15	20	22	27	31	34
2	6	9	13	19	22	26	30	36
4	7	11	16	20	23	26	32	36
2	5	12	14	17	23	27	31	34
4	6	11	15	17	21	26	29	36
2	8	11	15	17	22	28	29	33
1	5	12	14	19	22	28	31	34
3	6	10	15	20	21	28	31	36
3	5	11	13	19	22	28	29	36
1	6	11	13	18	22	27	29	35
2	6	12	14	20	23	25	29	36
2	6	10	14	20	22	28	30	34
4	8	12	14	18	24	26	32	33
1	5	11	13	20	21	27	29	33
4	8	11	13	20	21	28	29	33
1	8	9	15	19	23	28	32	34
1	7	9	14	19	22	28	29	36
2	8	11	14	19	21	25	31	36
3	6	9	15	17	22	28	29	33
1	6	11	13	20	23	27	29	36
3	8	11	13	18	22	25	30	33
4	8	12	15	18	22	28	32	33
2	7	10	16	20	23	25	31	33
1	7	11	13	20	23	25	32	34
2	5	12	14	19	22	27	29	33
2	8	9	16	20	23	25	29	36

NUMBAR OF RUNS = 30
NUMBER OF TRUE ASSERTIONS FOR THE FIRST LEVEL = 5
NUMBER OF TRUE ASSERTIONS FOR EACH OTHER LEVEL= 2

Sample 8: Fan-In Monotonic RBS

RULE NUMBER	MAX LHS	MAX RHS	LHS				RHS		RULE STATUS
1	4	2	14	12	8	-4	37	38	0
2	4	2	12	15	18	8	39	40	0
3	4	2	20	-11	13	8	41	42	0
4	4	2	7	20	16	2	43	44	0
5	3	2	1	20	11	0	45	41	0
6	4	2	6	15	2	20	46	40	0
7	4	2	14	1	7	9	39	47	0
8	4	2	13	-19	3	11	48	49	0
9	3	2	46	40	28	0	50	51	0
10	3	2	39	47	27	0	52	53	0
11	3	2	37	38	-21	0	54	55	0
12	3	2	37	38	22	0	56	57	0
13	4	2	46	40	23	26	56	58	0
14	3	2	43	44	28	0	59	55	0
15	3	2	56	57	33	0	60	61	0
16	3	2	54	55	33	0	62	63	0
17	3	2	50	51	33	0	64	65	0
18	4	2	56	57	30	36	66	67	0
19	3	2	56	58	32	0	68	66	0
20	4	2	56	58	34	29	69	66	0

SHAPE: FAN-IN
MONOTONICITY: MONOTONIC
NUMBER OF RULES = 20
NUMBER OF LEVELS = 3

Sample 9: Fan-In Non-Monotonic RBS

RULE NUMBER	MAX LHS	MAX RHS	LHS				RHS		RULE STATUS
1	4	2	14	12	8	-4	37	38	0
2	4	2	12	15	18	8	39	40	0
3	4	2	20	-11	13	8	41	42	0
4	4	2	7	20	16	2	43	44	0
5	3	2	1	20	11	0	45	-41	0
6	4	2	6	15	2	20	46	40	0
7	4	2	14	1	7	9	39	47	0
8	4	2	13	-19	3	11	48	49	0
9	3	2	46	40	28	0	50	51	0
10	3	2	39	47	27	0	52	53	0
11	3	2	37	38	-21	0	54	55	0
12	3	2	37	38	22	0	56	57	0
13	4	2	46	40	23	26	-56	58	0
14	3	2	43	44	28	0	59	-54	0
15	3	2	56	57	33	0	60	61	0
16	3	2	54	55	33	0	62	63	0
17	3	2	50	51	33	0	64	65	0
18	4	2	56	57	30	36	66	67	0
19	3	2	-56	58	32	0	68	66	0
20	4	2	-55	58	34	29	69	66	0

SHAPE: FAN-IN
MONOTONICITY: NON-MONOTONIC
NUMBER OF RULES = 20
NUMBER OF LEVELS = 3

```
C----------------- ENGINE 1 SIMULATOR (VERSION 2) -------------
C
C       ENGINE SPECIFICATIONS
C       ---------------------
C       PRODUCTION-ORDERING (PO1) CONFLICT-RESOLUTION STRATEGY
C       NO FILTERING
C
C       INPUT FILES
C       -----------
C       THE RBS
C       A RANDOM-ASSERTIONS SET
C
C       OUTPUT FILES
C       ------------
C       PERFORMANCE FILE
C       STATISTICS FILE
C
C       BOUNDARIES
C       ----------
C       1000 RANDOM ASSERTIONS
C
C       PURPOSE
C       -------
C       COLLECT STATISTICS ABOUT THE PERFORMANCE OF THE ENGINE
C
C
---------------------------------------------------------------
C
       INTEGER NR(1000,12),NF(4000,2)
       INTEGER FIRE(1000),TRU(4000),ASRT(1000,50)
       OPEN (UNIT=12,FILE='ASRT.DAT',STATUS='OLD')
       OPEN (UNIT=10,FILE='MAD.DAT',STATUS='OLD',RECL=132)
       OPEN (UNIT=11,FILE='PFIRST.DAT',STATUS='NEW',RECL=132)
       OPEN (UNIT=9,FILE='TRU.DAT',STATUS='NEW',RECL=132)
C
C-READ THE RBS
C
       READ(10,*)NRULE,MAXL,MAXR,NFIRE
       DO 1 I=1,NRULE
          READ(10,*) (NR(I,J),J=1,NFIRE)
1      CONTINUE
       READ(10,*)NFAC
       READ(12,*)NRUN,IASS
       GTC=0.0
       GTR=0.0
C
       DO 101 IJK=1,NRUN
C
       IREC=IASS
       NRF=0
C
C-READ AN ASSERTION VECTOR
C
       READ(12,*)(ASRT(IJK,J),J=1,IREC)
       DO 102 I=1,NRULE
```

```
          NR(I,NFIRE)=0
102     CONTINUE
C
C-INITIALIZE THE FACT MATRIX
C
        DO 2 I=1,NFAC
          NF(I,1)=-1
          NF(I,2)=0
2       CONTINUE
c
        DO 3 J=1,IREC
          NN=ASRT(IJK,J)
          NF(NN,1)=1
          NF(NN,2)=J
          TRU(NN)=TRU(NN)+1
3       CONTINUE
C
C-INITIALIZE THE MATCH-TESTS COUNTER
C
        TC=0.0
C
C -RECOGNIZE AND ACT CYCLE
C  ***********************
C       MATCH
C       *****
5        K=0
         DO 11 I=1,NRULE
C
C-CHECK A RULE
C
           IF (NR(I,NFIRE) .NE. 0) GOTO 11
           DO 12 J=4,NR(I,2)+3
              TC=TC+1.0
              NN=NR(I,J)
              MM=ABS(NN)
              IF (NF(MM,1)*NN .LT. 0) GOTO 11
12         CONTINUE
           K=K+1
           GOTO 66
11       CONTINUE
C
C-THE STOPPING CRITERION
C
66       IF (K .EQ. 0) GOTO 16
C
c       FIRE
C       ****
         NRTF=NR(I,1)
         NR(I,NFIRE)=1
         FIRE(NRTF)=FIRE(NRTF)+1
         NRF=NRF+1


         DO 15 J=4+MAXL,3+MAXL+NR(I,3)
            NN=NR(I,J)
```

```
            LL=ABS(NN)
            NF(LL,1)=NN/LL
            IF (NN.GT.0)  TRU(LL)=TRU(LL)+1
15       CONTINUE
C
C-END OF A CYCLE
C
         GOTO 5
C
C-END OF A RUN
C-WRITE THE PERFORMANCE MEASURE OF THE RUN
C
16       WRITE(11,*)TC,NRF
         GTC=GTC+TC
         GTR=GTR+NRF
101      CONTINUE
C
C-END OF ALL RUNS
C
         ATC=GTC/NRUN
         ARF=GTR/NRUN
         PRINT*,'AVERAGE TOTAL TESTS = ',ATC
         PRINT*,'AVERAGE RULE FIRED  = ',ARF
C
         DO 326 I=1,NRULE
           IF (FIRE(I).GT.0) LRF=LRF+1
326      CONTINUE
         PRINT*,'# OF RULES FIRED    = ',LRF
         WRITE(11,*)ATC,LRF,ARF
C
C-WRITE FACT-ASSERTION FREQUENCY
C
         WRITE(9,*)NRULE,NFAC,NRUN
         WRITE(9,*)(TRU(I),I=1,NFAC)
C
C-WRITE RULE-FIRING FREQUENCY
C
         WRITE(9,*)(FIRE(I),I=1,NRULE)
C
         STOP
         END
C
```

```
C---------------- ENGINE 2 SIMULATOR (VERSION 2) --------------
C
C       ENGINE SPECIFICATIONS
C       ---------------------
C       RECENCY (R5) CONFLICT-RESOLUTION STRATEGY
C       NO FILTERING
C
C       INPUT FILES
C       -----------
C       THE RBS
C       A RANDOM-ASSERTIONS SET
C
C       OUTPUT FILES
C       ------------
C       PERFORMANCE FILE
C       STATISTICS FILE
C
C       BOUNDARIES
C       ----------
C       1000 RANDOM ASSERTIONS
C       200 RULES IN THE CONFLICT SET
C
C       PURPOSE
C       -------
C       COLLECT STATISTICS ABOUT THE PERFORMANCE OF THE ENGINE
C
C
--------------------------------------------------------------
C
        INTEGER NR(1000,12),NF(4000,2),NC(200,6),NCSRT(200,6)
        DOUBLE PRECISION LEX(200)
        INTEGER FIRE(1000),TRU(4000),ASRT(1000,50)
        OPEN (UNIT=10,FILE='MAD.DAT',STATUS='OLD',RECL=132)
        OPEN (UNIT=11,FILE='PLEX.DAT',STATUS='NEW',RECL=132)
        OPEN (UNIT=12,FILE='ASRT.DAT',STATUS='OLD',RECL=132)
        OPEN (UNIT=9,FILE='TRU.DAT',STATUS='NEW',RECL=132)
C
C-READ THE RBS
C
        READ(10,*)NRULE,MAXL,MAXR,NFIRE
        DO 1 I=1,NRULE
           READ(10,*) (NR(I,J),J=1,NFIRE)
1       CONTINUE
        READ(10,*)NFAC
C
        READ(12,*)NRUN,IASS
        GTC=0.0
        GTR=0.0
C
        DO 101 IJK=1,NRUN
C
        NRF=0.0
        IREC=IASS

C-READ AN ASSERTION VECTOR
```

```
C
       READ(12,*)(ASRT(IJK,J),J=1,IREC)
       DO 102 I=1,NRULE
          NR(I,NFIRE)=0
102    CONTINUE
C
C-INITIALIZE THE FACT MATRIX
C
       DO 2 I=1,NFAC
          NF(I,1)= -1
          NF(I,2)=0
2      CONTINUE
C
       DO 3 J=1,IREC
          NN=ASRT(IJK,J)
          NF(NN,1)=1
          NF(NN,2)=J
          TRU(NN)=TRU(NN)+1
3      CONTINUE
C
C-INITIALIZE THE MATCH-TESTS COUNTER
C
        TC=0
C
C-RECOGNIZE AND ACT CYCLE
C ***********************
C      MATCH
C      *****
5       K=0
        DO 11 I=1,NRULE
C
C-CHECK A RULE
C
           IF (NR(I,NFIRE) .NE. 0) GOTO 11
           DO 12 J=4,NR(I,2)+3
              TC=TC+1.0
              NN=NR(I,J)
              MM=ABS(NN)
              IF (NF(MM,1)*NN .LT. 0) GOTO 11
12         CONTINUE
           K=K+1
C
C-PUT A RULE IN THE CONFLICT SET
C
           NC(K,1)=I
           NC(K,2)=NR(I,1)
           DO 13 J=3,NR(I,2)+2
              NN=NR(I,J+1)
              NC(K,J)=NF(NN,2)
13         CONTINUE
11      CONTINUE
C


C-THE STOPPING CRITERION
```

```
C
         IF (K .EQ. 0) GOTO 16
C
C-      SELECT
C       ******
         IF (K .GT. 1) THEN
            CALL SORT(NC,NCSRT,K,MAXL,200,6)
            CALL SLCT(NCSRT,K,MAXL,LEX,NRO,200,6)
         ELSE
            NRO=1
         ENDIF
C
C       FIRE
C       ****
        MM=NC(NRO,1)
        NR(MM,NFIRE)=1
        NN=NC(NRO,1)
        FIRE(NN)=FIRE(NN)+1
        NRF=NRF+1
        DO 15 J=4+MAXL,3+MAXL+NR(MM,3)
           NN=NR(MM,J)
           LL=ABS(NN)
           NF(LL,1)=NN/LL
           IF (NN.GT.0) TRU(LL)=TRU(LL)+1
           IREC=IREC+1
           NF(LL,2)=IREC
15      CONTINUE
C
C-NULLIFY THE CONFLICT-SET MATRIX
C
        DO 17 I=1,K
           DO 18 J=1,MAXL+2
              NC(I,J)=0
18         CONTINUE
           LEX(K)=0
17      CONTINUE
C
        DO 19,I=1,K
           DO 20,J=1,MAXL
           NCSRT(I,J)=0
20         CONTINUE
19      CONTINUE
C
C-END OF A CYCLE
C
        GOTO 5
C-END OF A RUN
C-WRITE THE PERFORMANCE MEASURE OF THE RUN
C
16      WRITE(11,*)TC,NRF
        GTC=GTC+TC
        GTR=GTR+NRF
101     CONTINUEC-END OF ALL RUNS
C
        ATC=GTC/NRUN
```

```
      ATR=GTR/NRUN
      PRINT*,'AVERAGE TOTAL TESTS = ',ATC
      PRINT*,'AVERAGE RULE FIRED  = ',ATR
C
      LRF=0
      DO 326 I=1,NRULE
         IF (FIRE(I).GT.0)LRF=LRF+1
326   CONTINUE
      PRINT*,'# OF RULES FIRED = ',LRF
      WRITE(11,*)ATC,LRF,ATR
C
C-WRITE FACT-ASSERTION FREQUENCY
C
      WRITE(9,*)NRULE,NFAC,NRUN
      WRITE(9,*)(TRU(I),I=1,NFAC)
C
C-WRITE RULE-FIRING FREQUENCY
C
      WRITE(9,*)(FIRE(I),I=1,NRULE)
C
      STOP
      END
C-----------------------------------------------------------------
C-SORT THE CONFLICT SET IN DESCENDING ORDER
C
      SUBROUTINE SORT(X,Y,N,M,MAXR,MAXC)
      INTEGER MAXR,MAXC,X(MAXR,MAXC),Y(MAXR,MAXC),N,M
      LOGICAL SORTED
      DO 10 I=1,N
         DO 11 J=3,M+2
            Y(I,J-2)=X(I,J)
11       CONTINUE
10    CONTINUE
C
      DO 21 I=1,N
         SORTED=.FALSE.
15       IF (.NOT. SORTED) THEN
            SORTED=.TRUE.
            DO 20 J=1,M-1
               IF (Y(I,J) .LT. Y(I,J+1)) THEN
                  TEMP=Y(I,J)
                  Y(I,J)=Y(I,J+1)
                  Y(I,J+1)=TEMP
                  SORTED=.FALSE.
               ENDIF
20          CONTINUE
          GOTO 15
          ENDIF
21    CONTINUE
      RETURN
      END

C-SELECT THE RULE HAVING THE MAXIMUM LEXICAL NUMBER
C
      SUBROUTINE SLCT(Y,N,M,LEX,MAX,MAXR,MAXC)
```

```
      INTEGER MAXR,MAXC,Y(MAXR,MAXC),M,N
      REAL L
      DOUBLE PRECISION BEX,LEX(MAXR)
C
C-CALCULATE THE LEXICAL NUMBER
C
      DO 10 I=1,N
         LEX(I)=0.0
         DO 11 J=1,M
            L=-3.0*J
            BEX=Y(I,J)*10**L
            LEX(I)=LEX(I)+BEX
11       CONTINUE
10    CONTINUE
C
C-FIND THE MAXIMUM LEXICAL NUMBER
C
      MAX=1
      DO 20 I=2,N
         IF (LEX(I) .LE. LEX(MAX)) GOTO 20
         MAX=I
20    CONTINUE
      RETURN
      END
C
```

```
C----------------- ENGINE 3 SIMULATOR (VERSION 2) -------------
C
C       ENGINE SPECIFICATIONS
C       ---------------------
C       RECENCY (R1) CONFLICT-RESOLUTION STRATEGY
C       NO FILTERING
C
C       INPUT FILES
C       -----------
C       THE RBS
C       A RANDOM-ASSERTIONS SET
C
C       OUTPUT FILES
C       ------------
C       PERFORMANCE FILE
C       STATISTICS FILE
C
C       BOUNDARIES
C       ----------
C       1000 RANDOM ASSERTIONS
C
C       PURPOSE
C       -------
C       COLLECT STATISTICS ABOUT THE PERFORMANCE OF THE ENGINE
C
C
-----------------------------------------------------------------
C
```

The code of this engine is similar to

engine 2 except for the selection subroutine.

Only the different parts are listed below.

```
C-      SELECT
C       ******
         IF (K .EQ. 0) GOTO 16
         IF (K .GT. 1) THEN
            CALL SORT(NC,NCSRT,K,MAXL,200,6)
            CALL SLCT(NCSRT,K,MAXL,NRO,200,6)
         ELSE
            NRO=1
         ENDIF
C
C-SELECT THE MOST-RECENT RULE
C
         SUBROUTINE SLCT(Y,N,M,MAX,MAXR,MAXC)
         INTEGER MAXR,MAXC,Y(MAXR,MAXC),M,N
         MAX=1
         DO 20 I=2,N
            IF (Y(I,1) .LE. Y(MAX,1)) GOTO 20
            MAX=I
20       CONTINUE
         RETURN
         END
```

```
C--------------- ENGINE 4 SIMULATOR (VERSION 2) ---------------
C
C       ENGINE SPECIFICATIONS
C       ---------------------
C       RECENCY (R3) CONFLICT-RESOLUTION STRATEGY
C       NO FILTERING
C
C       INPUT FILES
C       -----------
C       THE RBS
C       A RANDOM-ASSERTIONS SET
C
C       OUTPUT FILES
C       ------------
C       PERFORMANCE FILE
C       STATISTICS FILE
C
C       BOUNDARIES
C       ----------
C       1000 RANDOM ASSERTIONS
C
C       PURPOSE
C       -------
C       COLLECT STATISTICS ABOUT THE PERFORMANCE OF THE ENGINE
C
C
----------------------------------------------------------------
```

The code of this engine is similar to engine 3 except for the sorting subroutine, where sorting is applied in ascending order instead of descending order.

```
C--------------- ENGINE 5 SIMULATOR (VERSION 2) ---------------
C
C       ENGINE SPECIFICATIONS
C       ---------------------
C       PRODUCTION-ORDERING (PO1) CONFLICT-RESOLUTION STRATEGY
C       CONTROLLED PRODUCTIONS FILTER
C
C       INPUT FILES
C       -----------
C       THE RBS
C       A RANDOM-ASSERTIONS SET
C
C       OUTPUT FILES
C       ------------
C       PERFORMANCE FILE
C       STATISTICS FILE
C
C       BOUNDARIES
C       ----------
C       1000 RANDOM ASSERTIONS
C
C       PURPOSE
C       -------
C       COLLECT STATISTICS ABOUT THE PERFORMANCE OF THE ENGINE
C
C-----------------------------------------------------------------
C
```

The code of this engine is similar to engine 1 except for adding the code needed to define the groups and to test the rules by groups. Only the extra code is listed below.

```
       INTEGER GRP(10)
C
```

Data entry code

```
       PRINT*,'# OF GROUPS'
       READ*,NG
       DO 303 I=1,NG
          PRINT*,'END RULE NUMBER FOR GROUP ',I
          READ*,NRS
          GRP(I)=NRS
303    CONTINUE
C
```

Initialization code

```
C-RECOGNIZE AND ACT CYCLE
C ***********************
```

```
C       MATCH
C       *****
         DO 16 L=1,NG
5        K=0
         IF (L.EQ.1)NGS=1
         IF (L.GT.1)NGS=GRP(L-1)+1
C
C-CHECK THE RULES IN A GROUP
C
         DO 11 I=NGS,GRP(L)
C
```

Recognize and act cycle code

```
C-END OF A CYCLE
C
         GOTO 5
C
C-END OF A GROUP
C
16       CONTINUE
C
C-END OF A RUN
```

Output preparation code

```
C---------------- ENGINE 6 SIMULATOR (VERSION 2) ----------------
C
C       ENGINE SPECIFICATIONS
C       ---------------------
C       RECENCY (R1) CONFLICT-RESOLUTION STRATEGY
C       CONTEXT-RESTRICTED FILTERING
C
C       INPUT FILES
C       -----------
C       THE RBS
C       A RANDOM-ASSERTIONS SET
C
C       OUTPUT FILES
C       ------------
C       PERFORMANCE MEASURE
C       STATISTICS
C
C       BOUNDARIES
C       ----------
C       1000 RANDOM ASSERTIONS
C
C       PURPOSE
C       -------
C       COLLECT STATISTICS ABOUT THE PERFORMANCE OF THE ENGINE
C
C-----------------------------------------------------------------
       INTEGER NR(1000,12),NF(4000,2),REC(4000),INDX(4000,0:200)
       INTEGER FIRE(1000),TRU(4000),ASRT(1000,50)
       OPEN (UNIT=12,FILE='ASRT.DAT',STATUS='OLD')
       OPEN (UNIT=10,FILE='MAD.DAT',STATUS='OLD',RECL=132)
       OPEN (UNIT=11,FILE='FOCUS.DAT',STATUS='NEW',RECL=132)
       OPEN (UNIT=9,FILE='TRU.DAT',STATUS='NEW',RECL=132)
C
C-READ THE RBS
C
       READ(10,*)NRULE,MAXL,MAXR,NFIRE
       DO 1 I=1,NRULE
          READ(10,*) (NR(I,J),J=1,NFIRE)
1      CONTINUE
       READ(10,*)NFAC
C
c-BUILD THE INDEX
C
       DO 201 I=1,NRULE
          NGAL=0
          DO 202 J=4,3+NR(I,2)
             NN=NR(I,J)
             NN=ABS(NN)
             INDX(NN,0)=INDX(NN,0)+1
             MM=INDX(NN,0)
             INDX(NN,MM)=NR(I,1)
202       CONTINUE
201    CONTINUE
C       READ(12,*)NRUN,IASS
       GTC=0.0
```

```
      GTR=0.0
C
      DO 101 IJK=1,NRUN
C
      NRF=0
      IREC=IASS
C
C-READ AN ASSERTION VECTOR
C
      READ(12,*)(ASRT(IJK,J),J=1,IREC)
      DO 102 I=1,NRULE
      NR(I,NFIRE)=0
102   CONTINUE
C
C-INITIALIZE THE FACT MATRIX
C
      DO 2 I=1,NFAC
         NF(I,1)=-1
         NF(I,2)=0
         REC(I)=0
2     CONTINUE
c
      DO 3 J=1,IREC
         NN=ASRT(IJK,J)
         NF(NN,1)=1
         NF(NN,2)=J
         TRU(NN)=TRU(NN)+1
         REC(J)=NN
3     CONTINUE
C
C-INITIALIZE THE MATCH-TESTS COUNTER
C
       TC=0.0
C
C-RECOGNIZE AND ACT CYCLE
C ***********************
C     MATCH
C     ****8
5      K=0
C
C-GET THE MOST RECENT FACT
C
       NPT=IREC
210    KI=REC(NPT)
C
       DO 11 KJ=1,INDX(KI,0)
C
C-CHECK A RULE FROM THE INDEX
C
          I=INDX(KI,KJ)
          IF (I.EQ.0)GOTO 213
          IF (NR(I,NFIRE) .NE. 0) GOTO 11
          DO 12 J=4,NR(I,2)+3
             TC=TC+1.0
             NN=NR(I,J)
```

```
                  MM=ABS(NN)
                  IF (NF(MM,1)*NN .LT. 0) GOTO 11
12             CONTINUE
               K=K+1
               GOTO 66
11          CONTINUE
C
C-GET THE NEXT MOST-RECENT FACT
C
213         NPT=NPT-1
            IF (NPT.GT.0)GOTO 210
C
C-THE STOPPING CRITERION
C
66          IF (K .EQ. 0) GOTO 16
C
c          FIRE
C          ****
            NRTF=NR(I,1)
            NR(I,NFIRE)=1
            FIRE(NRTF)=FIRE(NRTF)+1
            NRF=NRF+1
            DO 15 J=4+MAXL,3+MAXL+NR(I,3)
               NN=NR(I,J)
               LL=ABS(NN)
               NF(LL,1)=NN/LL
               IREC=IREC+1
               NF(LL,2)=IREC
               REC(IREC)=NN
               IF (NN.GT.0)  TRU(LL)=TRU(LL)+1
15          CONTINUE
C
C-END OF A CYCLE
C
            GOTO 5
C
C-END OF A RUN
C-WRITE THE PERFORMANCE MEASURE OF THE RUN
C
16             WRITE(11,*)TC,NRF
               GTC=GTC+TC
               GTR=GTR+NRF
101            CONTINUE
C
C-END OF ALL RUNS
C
            LRF=0
            DO 412 I=1,NRULE
               IF (FIRE(I).GT.0)LRF=LRF+1
412         CONTINUE

            PRINT*,'# OF RULE FIRED = ',LRF
            ATC=GTC/NRUN
            ATR=GTR/NRUN
            PRINT*,'AVERAGE TOTAL TESTS = ',ATC
```

```
      PRINT*,'AVERAGE RULE FIRED  = ',ATR
      WRITE(11,*)ATC,LRF,ATR
C
C-WRITE FACT-ASSERTION FREQUENCY
C
      WRITE(9,*)NRULE,NFAC,NRUN
      WRITE(9,*)(TRU(I),I=1,NFAC)
C
C-WRITE RULE-FIRING FREQUENCY
C
      WRITE(9,*)(FIRE(I),I=1,NRULE)
      STOP
      END
C
```

```
C-------------------- FACT-ORDERING PROGRAM -------------------
C
C      INPUT FILES
C      -----------
C      THE ORIGINAL RBS
C      THE STATISTICS FILE
C
C      OUTPUT FILES
C      ------------
C      THE NEW STRUCTURE OF THE RBS
C
C      BOUNDARIES
C      ----------
C      4 CONDITIONS IN THE LHS
C
C      PURPOSE
C      -------
C      REARRANGE THE LHS ACCORDING TO FACT-ASSERTION FREQUENCY
C
C-----------------------------------------------------------------
C
      INTEGER NR(1000,12),TRU(4000),S1(4),S2(4)
      OPEN (UNIT=10,FILE='MAD.DAT',STATUS='OLD',RECL=132)
      OPEN (UNIT=11,FILE='TRU.DAT',STATUS='OLD',RECL=132)
      OPEN (UNIT=12,FILE='ARNG.DAT',STATUS='NEW',RECL=132)
C
C-READ THE ORIGINAL RBS
C
      READ(10,*)NRULE,MAXL,MAXR,NFIRE
      DO 5 I=1,NRULE
         READ(10,*)(NR(I,J),J=1,NFIRE)
5     CONTINUE
C
C-READ FACT-ASSERTION FREQUENCY
C
      READ(11,*)NRULE,NFAC,NRUN
      READ(11,*)(TRU(I),I=1,NFAC)
C
      WRITE(12,*)NRULE,MAXL,MAXR,NFIRE
C
C-FIND THE MAXIMUM FREQUENCY
C
      MAX=TRU(1)
      DO 17 I=2,NFAC
         IF (TRU(I).GT.MAX)MAX=TRU(I)
17    CONTINUE
C
C-REARRANGE
C *********
C
      DO 10 I=1,NRULE
         K=0

C-    REARRANGE THE LHS OF A RULE
C     ***************************
```

```
C
          DO 15 J=4,3+NR(I,2)
C-
C           STORE THE FREQUENCY IN AN ARRAY
C           ******************************
C
             K=K+1
             S1(K)=NR(I,J)
             IF (S1(K).LT.0)THEN
                S2(K)=MAX-TRU(S1(K))
             ELSE
                S2(K)=TRU(S1(K))
             ENDIF
15        CONTINUE
C
C           SORT
C           ****
          CALL FSRT(S1,S2,NR(I,2),MAXL)
          K=0
C
C           WRITE THE NEW LHS
C           *****************
C
          DO 20 J=4,3+NR(I,2)
             K=K+1
             NR(I,J)=S1(K)
20        CONTINUE
          WRITE(12,*)(NR(I,JJ),JJ=1,NFIRE)
10     CONTINUE
C
       WRITE(12,*)NFAC
       STOP
       END
C
C-----------------------------------------------------------------
C
```

```
C-SORT THE LHS IN ASCENDING ORDER
C
```

```
      SUBROUTINE FSRT(S1,S2,L,M)
      INTEGER L,M,S1(M),S2(M)
      LOGICAL SORTED
      SORTED=.FALSE.
10    IF (.NOT. SORTED) THEN
         SORTED=.TRUE.
         DO 20 J=1,L-1
            IF (S2(J).GT.S2(J+1)) THEN
               ITEMP=S2(J)
               S2(J)=S2(J+1)
               S2(J+1)=ITEMP
               JTEMP=S1(J)
               S1(J)=S1(J+1)
               S1(J+1)=JTEMP
               SORTED=.FALSE.
            ENDIF
20       CONTINUE
         GOTO 10
      ENDIF
      RETURN
      END
C
```

```
C-------------------- RULE-ORDERING PROGRAM -------------------
C
C      INPUT FILES
C      -----------
C      THE ORIGINAL RBS
C      THE STATISTICS FILE
C
C      OUTPUT FILES
C      ------------
C      THE ORDERED RBS
C
C      PURPOSE
C      -------
C      REARRANGE THE RBS ACCORDING TO RULE-FIRING FREQUENCY
C
C---------------------------------------------------------------
C
      INTEGER NR(1000,12),TRU(4000),MR(1000,12)
      INTEGE  FIRE(1000),SFIRE(1000)
      OPEN (UNIT=10,FILE='MAD.DAT',STATUS='OLD',RECL=132)
      OPEN (UNIT=11,FILE='TRU.DAT',STATUS='OLD',RECL=132)
      OPEN (UNIT=12,FILE='ARNGR.DAT',STATUS='NEW',RECL=132)
C
C-READ THE ORIGINAL RBS
C
      READ(10,*)NRULE,MAXL,MAXR,NFIRE
      DO 5 I=1,NRULE
         READ(10,*)(NR(I,J),J=1,NFIRE)
5     CONTINUE
C
C-READ THE STATISTICS FILE
C
      READ(11,*)NRULE,NFAC,NRUN
      READ(11,*)(TRU(I),I=1,NFAC)
      READ(11,*)(FIRE(I),I=1,NRULE)
C
C-SAVE THE ORIGINAL ORDER IN AN ARRAY
C
      DO 31 I=1,NRULE
      SFIRE(I)=I
31    CONTINUE
C
      WRITE(12,*)NRULE,MAXL,MAXR,NFIRE
C
C-SORT
C
      CALL RSRT(SFIRE,FIRE,NRULE)




C-
C-SAVE THE NEW STRUCTURE
```

```
C
      DO 32 I=1,NRULE
         K=SFIRE(I)
         DO 33 J=1,NFIRE
            MR(I,J)=NR(K,J)
33       CONTINUE
         WRITE(12,*)(MR(I,J),J=1,NFIRE)
32    CONTINUE
C
      WRITE(12,*)NFAC
      STOP
      END
C
C-----------------------------------------------------------------
C
C-SORT THE RULES IN DESCENDING ORDER
C
      SUBROUTINE RSRT(S1,S2,M)
      INTEGER L,M,S1(M),S2(M)
      LOGICAL SORTED
      SORTED=.FALSE.
10    IF (.NOT. SORTED) THEN
         SORTED=.TRUE.
         DO 20 J=1,M-1
            IF (S2(J).LT.S2(J+1)) THEN
               ITEMP=S2(J)
               S2(J)=S2(J+1)
               S2(J+1)=ITEMP
               JTEMP=S1(J)
               S1(J)=S1(J+1)
               S1(J+1)=JTEMP
               SORTED=.FALSE.
            ENDIF
20       CONTINUE
         GOTO 10
      ENDIF
      RETURN
      END
C
```

Vitae

Lieutenant Colonel Nizar M. Mahaba [redacted] [redacted] in 1968 [redacted] attended Cairo University from which he received the degree of Bachelor of Science in Civil Engineering in July, 1973. He joined the Egyptian Army in 1974 in which he served in the Engineering Corps until 1980. He received a Diploma in Computer Science from Cairo University, the Institute of Statistical Studies & Research in September, 1982 and a Diploma in Operations Research from the Military Technical College in December, 1982. He then served in the Information System Department until entering the School of Engineering, the United States Air Force Institute of Technology in June, 1987.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

Field	Entry
1a. REPORT SECURITY CLASSIFICATION	UNCLASSIFIED
1b. RESTRICTIVE MARKINGS	
2a. SECURITY CLASSIFICATION AUTHORITY	
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
3. DISTRIBUTION / AVAILABILITY OF REPORT	Approved for public release; distribution unlimited.
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	AFIT/GOR/ENS/88D-12
5. MONITORING ORGANIZATION REPORT NUMBER(S)	
6a. NAME OF PERFORMING ORGANIZATION	School of Engineering
6b. OFFICE SYMBOL (If applicable)	AFIT/ENG
7a. NAME OF MONITORING ORGANIZATION	
6c. ADDRESS (City, State, and ZIP Code)	Air Force Institute of Technology, Wright Patterson AFB OH 45433-6583
7b. ADDRESS (City, State, and ZIP Code)	
8a. NAME OF FUNDING / SPONSORING ORGANIZATION	
8b. OFFICE SYMBOL (If applicable)	
9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c. ADDRESS (City, State, and ZIP Code)	
10. SOURCE OF FUNDING NUMBERS	PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification)	Simulating Rule-Based Systems
12. PERSONAL AUTHOR(S)	Nizar Mahmoud Mahaba, Ltc, Egyptian Army
13a. TYPE OF REPORT	MS Thesis
13b. TIME COVERED	FROM ______ TO ______
14. DATE OF REPORT (Year, Month, Day)	1988 December
15. PAGE COUNT	159
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP
12	03	
12	09	

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

Artificial Intelligence, Rule-Based Systems, Simulation Statistics and Probability, Mont Carlo Method.

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The purpose of this study is to develop a methodology for evaluating the performance of rule-based systems (RBSs) using a simulation approach. A numerical scheme for knowledge representation; facts are represented by integer numbers and the rules and data memories are represented by matrices. The numeric representaion can be handled by simplified algorithms that simulate the function of different types of inference engines. Six types of forward-chaining inference engines that vary according to the conflict-resolution strategy and the implementation of filters are simulated and compared. The number of match-tests of the rules against the data memory is used as a measure of performance to estimate the relative matching effort for each inference engine. Also, a methodology to reduce the matching effort of a RBS by changing the order of the facts in the left hand side or

Field	Entry
20. DISTRIBUTION / AVAILABILITY OF ABSTRACT	☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS
21. ABSTRACT SECURITY CLASSIFICATION	UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL	Bruce W. Morlan, Major, USAF
22b. TELEPHONE (Include Area Code)	(513) 255-3576
22c. OFFICE SYMBOL	AFIT/ENG

DD Form 1473, JUN 86 Previous editions are obsolete. SECURITY CLASSIFICATION OF THIS PAGE

19. (continue)

changing the order of the rules is described.

To simulate RBSs, it is assumed that probabilistic relations among the assertions to a RBS can be identified and specified by the experts or after running the system for some time. The numeric representation and the probabilistic relations provide the environment needed to build a simulation model. A rule-generator program is developed to randomly generate RBSs with different specifications. RBSs that vary in size (the number of rules), shape (the number of facts in both sides of the rule), and monotonicity (monotonic or non-monotonic) are generated and used in experimentation.

Two types of experiments are performed on the generated RBSs. The first type estimates the reduction ratio in matching effort achieved by rearranging the facts or the rules in a RBS. The second type estimates the reduction ratio in matching effort achieved by implementing two types of filters or changing the conflict-resolution strategy.