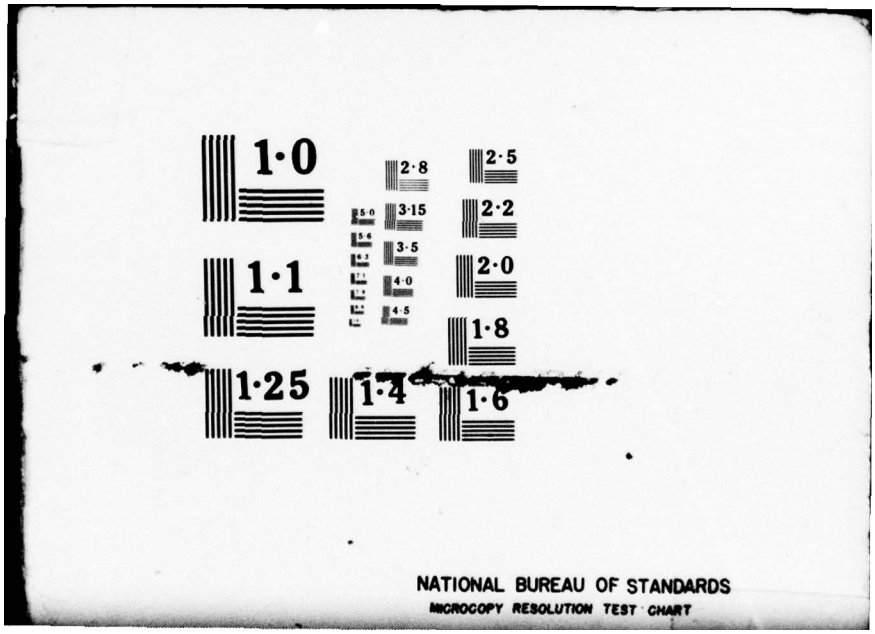
AD-A057 901 NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF F/G 9/2
A USER'S GUIDE FOR THE RITA PRODUCTION RULE SYSTEM.(U)
JUN 78 T E WARREN

UNCLASSIFIED NL

1 OF 2
ADA
057901


1·0
1·1
1·25
2·8
3·15
3·5
4·0
4·5
1·4
2·5
2·2
2·0
1·8
1·6
NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART

(2) LEVEL II

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A USER'S GUIDE FOR THE
RITA PRODUCTION RULE SYSTEM

by

Thomas Early Warren

June 1978

Thesis Advisor; G. K. Poock

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER
2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)

A USER'S GUIDE FOR THE RITA PRODUCTION RULE SYSTEM.

5. TYPE OF REPORT & PERIOD COVERED

Master's Thesis, (June 1978)

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)

Thomas Early Warren

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION NAME AND ADDRESS

Naval Postgraduate School
Monterey, California 93940

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS

Naval Postgraduate School
Monterey, California 93940

12. REPORT DATE

June 1978

13. NUMBER OF PAGES

142

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)

Naval Postgraduate School
Monterey, California 93940

139 p.

15. SECURITY CLASS. (of this report)

Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Tutorial
RITA
Intelligent Terminal
Production rules

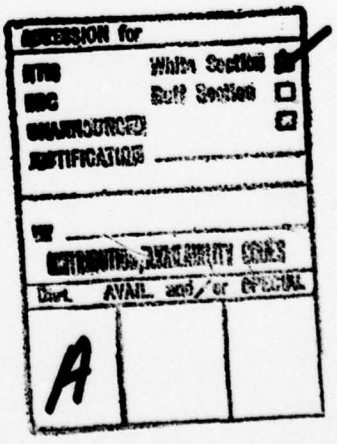
20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Interactive computer systems have traditionally communicated with the sophisticated computer user through a precise computer language. That language involved numerous control structures. Hardware costs and technology trends have excited a new breed of computer user. They, however, have lacked the sophistication and technical background to use most of the currently designed computer systems. In an → next page

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

cont. attempt to meet the needs of the new breed of computer-naive user, The Rand Corporation has developed the Rule-directed Interactive Transaction Agent (RITA) system. RITA has used production systems and an English-like grammar to simplify program control structures and to communicate with the computer-naive user in a language of familiarity. A tutorial document has been developed to support the computer-naive user in using the RITA system.



Approved for public release; distribution unlimited

A USER'S GUIDE FOR THE RITA PRODUCTION RULE SYSTEM

by

Thomas Early Warren
Lieutenant, United States Navy
B.S., United States Naval Academy, 1973

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1978

Author Thomas E. Warren

Approved by: Gary Poock
Thesis Advisor

Second Reader

Chairman, Department of Computer Science

Dean of Information and Policy Sciences

ABSTRACT

Interactive computer systems have traditionally communicated with the sophisticated computer user through a precise computer language. That language involved numerous control structures. Hardware costs and technology trends have excited a new breed of computer user. They, however, have lacked the sophistication and technical background to use most of the currently designed computer systems. In an attempt to meet the needs of the new breed of computer-naive user, The Rand Corporation has developed the Rule-directed Interactive Transaction Agent (RITA) system. RITA has used production systems and an English-like grammar to simplify program control structures and to communicate with the computer-naive user in a language of familiarity. A tutorial document has been developed to support the computer-naive user in using the RITA system.

TABLE OF CONTENTS

ACKNOWLEDGEMENT

Professor Gary K. Poock and Lieutenant Colonel R. Jay Roland, USAF, have reviewed all of the material contained in this report. Their suggestions, comments, and support have been extremely valuable and greatly appreciated.

Much of the credit must go to my wife. Her support and patience have contributed significantly to the completion of the thesis.

I. INTRODUCTION

As the population of computer users has continued to rise and the costs of computer hardware have declined, developers of computer hardware and software have prepared to meet the challenge of a new breed of computer user. That user was well versed in the responsibilities of his job and needed a computer to accomplish his tasks more efficiently and economically. However, the new breed of computer user was not a "computer sophisticate." He viewed the computer as a tool to accomplish a job. The user was neither a technical person nor an abstract thinker. His language was English, not Fortran or Cobol.

To reach the new breed of computer user, the language of interaction with the computer must have been easy to use and very English-like. The language must have contained very few, if any, fancy control structures. It must have allowed the user to write computer programs in the same natural way as he has thought about a problem.

The Rand Corporation, in a research grant sponsored by the Defense Advanced Research Projects Agency (DARPA), has developed a computer language with an operating environment in which programs can be written to perform a variety of tasks. The language and its operating environment have been called RITA (Rule-directed Interactive Transaction Agent.)

RITA has met the needs of the new breed of computer user by having an English-like grammar and only one primary

control structure, production rules.

The purpose of this research was to describe a particular production rule system (RITA) and to develop a tutorial document to introduce the computer-naive user to that system. The "computer-naive" user, as discussed in this report, has little or no computer experience, has a limited technical background, and must interact with a computer.

II. BACKGROUND

The Rule-directed Interactive Transaction Agent (RITA), a set of computer programs written in the "C" programming language, was developed on a PDP-11/45 minicomputer. The programs were developed to perform a variety of user tasks, both under direct user control or semiautonomously over extended periods of time. Among these tasks were (1) handling and modifying data on local storage files, (2) communicating interactively with external information systems over telephone lines or the ARPANET, (3) providing local instructional information and error-checking of input data, and (4) heuristic modeling of a limited set of relationships [Anderson and Gillogly, 1976].

For a system to meet the needs of the computer-naive user, several criteria were necessary. The criteria for the system were (1) be capable of explaining its behavior upon request, (2) be capable of having its behavior modified by the user, (3) be controlled by a set of heuristics stated as rules, rather than as formal algorithms, and (4) retain a memory of tasks assigned, progress, schedules, and deadlines. A fifth requirement of the system, although only implied, was that the RITA system must communicate with the user in a natural language, such as English [Anderson and Gillogly, 1976].

To meet the designers' requirements, the RITA system used production systems. Production systems are sets of

predicate - action rules (If - Then rules) operating on a database under the direction of a rule interpreter or monitor.

III. DESIGN

A. PRODUCTION SYSTEMS

A production system is a collection of rules of the form conditions --> actions [Newell and Simon, 1972], where the conditions are statements about the contents of a data base and the actions are procedures which may have modified the contents of that data base. The production system checks the data base to determine if the conditions of the production rule are true. When the conditions of the production rule are true, all the actions associated with the true rule are performed. The production rule system continues to check the conditions of the production rules and performs the actions of the true rules until the conditions for all the production rules are false or a special halting action is executed [Waterman, 1976].

The execution of the production rules was based upon the contents of the data base. These production rules caused the sequence of program execution to depend completely upon the contents of the data base. Typical computer programs executed sequentially or have had explicit knowledge about where code will next be executed in the program [Waterman, 1976].

Production systems, because of the method of program execution, have provided a simple and uniform way of handling control flow and data management in programs which exhibit intelligent behavior. They have been particularly

useful for developing programs which can learn from experience, i.e., which can demonstrate adaptive behavior [Waterman, 1976].

Production systems have fallen into two general classes. The first class was the conventional condition-driven one in which the conditions of a production rule were compared to the contents of the data base. Based upon the determination that all the conditions of the production rule were true, the appropriate actions were performed. Several systems have used this scheme [Newell and Simon, 1972; Newell, 1973; Waterman, 1975].

The second class of production systems interacted with the data base through the actions of a production rule [Shortliffe and others, 1975; Davis, 1976]. The action-driven systems have production rules analogous to logical implication statements, i.e., A & B --> C, means that if A and B are true then C is true. The action-driven system would try to show that C was true by looking for C in the data base. Items were only in the data base if they were true. If that failed, then the action-driven system would attempt to show that A and B were true. If A and B were true, then C was true and would be put into the data base [Waterman, 1976].

The conditon-driven production system and the action-driven production system were incorporated into the RITA design and implementation. The condition-driven production system provided the flexibility necessary for a system

requiring a dynamic data base. The action-driven production system provided the ability to answer user questions through deductive inference.

Examples of both classes of production systems are shown below. In these examples, data base elements are letters and are considered true if they are in the data base. The first example is that of a condition-driven production system.

DATA BASE: B

RULES: 1. A --> C

2. B --> A

The conditions of Rule 1 were tested. The element A was not in the data base, so Rule 1 was false. The conditions of Rule 2 were tested. Since the element B was in the data base, Rule 2 was true. The actions of Rule 2 were performed. Element A became a member of the data base. The procedure began again and tested Rule 1. The element A was now in the data base. Rule 1 was true and the actions were performed. The element C was put into the data base. For purposes of the example, assume Rule 1 had a special halting action. The condition-driven system stopped testing rules as a result of the special halting action in Rule 1. The data base has been modified by the testing and execution of the two rules in the example. The data base now contains three elements: A, B, and C.

The second example [Waterman, 1976] is that of an

action-driven production system. Again, the data base elements in this example are letters and considered true if they are in the data base.

DATA BASE: B

RULES: 1. A --> C

2. B --> A

3. C --> D

In this example, the goal was to show that D was true. The system first checked to see if the data base contained the element D. The data base did not contain the element D. The action-driven system tried to deduce D by using the rules that had D on the right-hand side. Rule 3 had D on the right-hand side. The system checked Rule 3 to see if the element C was a part of the data base. It was not there. The system then checked the right-hand side of each rule to see if the element C was there. If the system could have found a rule that made element C true, then Rule 3 could have been executed to make element D true. The system checked Rule 1 because element C was on the right-hand side. Rule 1 was not true. Element A was not part of the data base. The system then checked the rules that had element A on the right-hand side. Rule 2 was tested. It was true because element B was a part of the data base. Rule 2 was executed. Element A became a part of the data base. Rule 1 was executed. Element C became part of the data base. Rule 3 was then executed. Element D became a part of the data

base. The goal has been satisfied. Element D is now true.

The action-driven system solved the problem by back-tracking to find a rule that was useful in causing the requested goal to be satisfied. The action-driven system, as implemented in the RITA system, would have prompted the user if the back-tracking process was not successful in solving the problem. The user could have then provided the necessary information to assist the action-driven system in satisfying the goal.

B. ENGLISH-LIKE LANGUAGE

For the computer user to successfully interact with a computer, the computer must be told what to do and the computer must understand the instructions [Weizenbaum, 1976]. The needs of the computer-naive user and the computer have been in conflict. The computer-naive user has a need to communicate with a computer, but has not understood the conventional computer languages. The computer-naive user has been best able to interact with the computer through a language of familiarity, the English language. But the English language has been less precise than most computer languages. The computer is unlikely to prepare a proper program unless its comprehension of the problem is entirely correct [Weizenbaum, 1976].

One of the most appealing aspects of the RITA system has been the near-English command language. The designers of the RITA system chose an English-like language for its broad appeal to the new user population.

C. INTELLIGENT TERMINAL

The RITA language has been designed for broad-based user appeal. The language specified has been English-like and the basic control structures, production rules, have the qualities of simplicity and elegance. However, the computer-naive user has neither the expertise nor the resources needed to maintain RITA as currently implemented on the PDP-11/45 minicomputer.

The designers of the RITA system have felt that the acceptable alternative to the current hardware requirement was to develop an intelligent terminal [Anderson and Gillogly, 1976]. Hardware and software technology advances have indicated the feasibility of such a terminal. The intelligent terminal would provide a powerful tool at relatively low cost.

The RITA system has been designed as a set of programs expected to reside in an intelligent terminal. The following observations concerning man/machine interaction and its supporting technologies have formed the basis for the research by The Rand Corporation into the area of intelligent terminals [Anderson and Gillogly, 1976]:

1. The rapid decline in the cost of computer hardware and data communications has made the purchase of interactive computer-based information systems cost effective for a broad category of users. However, the user will need assistance in tailoring the system to his specific needs and especially in freeing him from routine interaction and

protocols not directly related to the performance of his job.

2. Projected computer hardware cost trends and advances in microprocessor technology make it extremely likely that interactive computer terminals can be produced prior to 1983 containing processing power equivalent to a present-day minicomputer.

3. The intelligent terminal could provide local information storage and handling capabilities. This service, provided locally, could yield high speed responses, lower cost, increased reliability, and improved security. The immediacy of response (e.g., to simple input error conditions) is the primary reason for advocating local processing.

IV. RITA IMPLEMENTATION

A. SIMPLICITY OF PROGRAM WRITING

The RITA system has been designed as a set of programs that will eventually reside in an intelligent terminal and be used by computer-naive persons. The ability for the group of users to write, read, and understand RITA programs were key goals of the implementation.

The production systems provided simple program control. More importantly, the user had to learn only one control structure to write RITA programs. Rules of the form - If a set of conditions are true, Then perform a set of actions - comprised the entire RITA program. The following example demonstrates the character of the production system as implemented in RITA:

```
[DATA BASE]

OBJECT ship<1>:
    name is "JFK",
    type is "Carrier";

OBJECT ship<2>:
    name is "Spruance",
    type is "Destroyer";

[RULE SET]
```

```
RULE one:
    IF:  there is a ship whose name is "JFK"
         and whose port is not known
    THEN:  deduce the port of the ship;

GOAL one:
    IF:  there is a ship whose name is "JFK"
    THEN:  set the port of the ship to "Norfolk"
           and display object ship<1>;
```

Information in the data base has been stored in objects. The objects have attributes and values. Object ship<1> has two attributes, name and type. The value "JFK" has been associated with the attribute name and the value "Carrier" has been associated with the attribute type. Object ship<2> has two attributes with associated values.

Rule one has two conditions: Is there a ship whose name is "JFK"? and Is there a ship whose port is not known? Not known indicates the element is not in the data base. Rule one has one action: Deduce the port of the ship. Deduce has been implemented as a RITA command to start the action-driven production system. Goal one has one condition: Is there a ship whose name is "JFK"? Goal one has two actions: Set the port of the ship to "Norfolk" and display object ship<1>. Set caused the information in the data base to be modified and display caused information to be printed at the user's terminal.

When this program was executed, rule one was true. The

action part of rule one caused the deduction of the port of the ship. The action-driven production system looked at the right-hand side of all the goals to determine the appropriate goal to check. Goal one was tested and found to be true. The port of the ship was set to "Norfolk." The deduction stopped because the question "What is the port of the ship?" has been answered. The display command caused the object ship<1> to be printed at the terminal in the following manner:

```
OBJECT ship<1>
    name is "JFK",
    type is "Carrier",
    port is "Norfolk";
```

The structure of the RITA syntax has been designed to be straight-forward. The English-like grammar assisted the user in writing programs in much the same manner as he thought about the problem.

The RITA system was designed to prompt the user when a rule, goal or object was added. The immediacy of response reinforced the user's thinking about the proper method for writing RITA programs. The RITA system also prompted the user when a syntax error was made. The location of the mistake was identified so that the error could be corrected.

B. INCREMENTAL KNOWLEDGE

Few people know the fundamental concepts, relations, and operations that characterize the use of a computer.

Consequently, teaching must begin at a more basic level [Grignetti and others, 1974]. When a program grows in power by an evolution of partially understood patches and fixes, the programmer begins to lose track of internal details, loses his ability to predict what will happen, begins to hope instead of know, and watches the results as though the program were an individual whose range of behavior were unknown [Weizenbaum, 1976].

The computer-naive user should not be expected to make or understand patches to large programs. He has been better able to add or delete a production rule that performed a particular task. Given the ability to monitor the progress of each production rule, the user has been able to determine what his program was doing at all times.

The production systems as implemented in *RITA* have provided an interesting form of program organization. Production rules have tended to represent independent components of behavior. The creation and addition of new production rules can be incremental, a feature which has facilitated modeling learning processes [Waterman 1970, 1975].

The computer-naive user has been able to take advantage of the incremental feature of the RITA system and write computer programs that perform a variety of tasks by adding a single production rule at a time. The user's learning has increased with the addition of each production rule. However, adding the fifth rule has been just as easy as

adding the first rule.

C. EXPLANATION SUBSYSTEMS

One of the designers' goals was to provide a system that was able to explain its behavior upon request. Two explanation subsystems were implemented that complement the two classes of production systems.

The first explanation subsystem kept a running account of all major activities during the RITA session. The file of events kept by the RITA session was valuable in determining the behavior of user programs. The intent of this subsystem was to provide information for post program execution analysis. The user was able to re-create the sequence of events for portions or all of the RITA session. Any difficulties encountered during the RITA session could be duplicated and analyzed.

The information was available to the user by issuing the RITA command "what." Also a range of events could be specified by using optional parameters with the "what" command.

The second explanation subsystem implemented in the RITA system served the user during program execution. The explanation subsystem was associated with the action-driven production system. This subsystem provided information to the user during the deduction of RITA goals. The user, by typing "why", caused the RITA system to display the requested subgoal that the action-driven production system sought to deduce. The user had a means of determining the

chaining process used by the RITA system to satisfy the user's request for the deduction of a RITA goal.

The implementation of both explantion subsystems was simplified because of the choice to use production systems. Their facility of allowing incremental information quite naturally provided for discrete activities to be recorded easily.

V. TUTORIAL

A. OBJECTIVES

The RITA system was designed and implemented as a human-oriented interactive computer system. Two support documents for the RITA system have been produced [Anderson and Gillogly, 1976; Anderson and others, 1977]. Neither document was written to explain the use of RITA to a computer-naive person. It was the opinion of this author that a carefully constructed document was necessary to introduce the computer-naive person to the RITA system. As a result of this thinking, the major thrust of the research was to deliver a useable tutorial explaining the RITA system.

The objectives of the tutorial follow:

1. Introduce the computer-naive user to the features of the RITA system as simply as possible.

2. Write the tutorial assuming the user has no programming experience.

3. Demonstrate the major features of the RITA system by simple examples.

4. Require computer interaction by the user during the tutorial.

5. Identify plainly the sections to be typed by the user.

6. Introduce RITA topics in order of simplicity and importance.

The first objective was the result of the need for a user document at a basic level. The user, prior to the development of the accompanying tutorial, was forced to piece together information from a reference manual in order to use the RITA system. The reference manual [Anderson and others, 1977] was written for a "sophisticated user."

The tutorial was written for a computer-naive user, therefore, computer terms were explained in layman's words. The user was not presumed to understand the concepts of program execution or data base management.

The use of simple examples to illustrate the nature of RITA commands assured the user that the RITA commands worked. Without examples, the user was not sure how the RITA commands functioned.

The requirement for the user to actively participate in the learning process by interacting with the computer provided two benefits. The user was forced to get involved with the computer, maintaining a certain attention level, and the typing of RITA commands reinforced the user's understanding of RITA.

The portions of the tutorial to be typed at the computer terminal by the user were separated from the other text. Explicit instructions to the user were capitalized.

In meeting the last objective, that of introducing RITA topics in order of simplicity and importance, the user-oriented tutorial provided the medium for a well-structured presentation of information.

The tutorial was not designed to encompass each feature of the RITA system nor was it designed to replace any previous user documents. The tutorial was designed to present material to the computer-naive user in a simple and useful manner.

B. PRESENTATION OF INFORMATION

The information in the tutorial was presented to the user in order of simplicity and importance. In each section the material to be illustrated was described and definitions of technical words were provided. Each specific RITA command was demonstrated by a simple example. The user was asked to type in the example and observe the computer's response. The example was then thoroughly explained and the response of the computer was described when necessary.

The examples were intentionally kept simple and brief to reduce the amount of new information presented to the user. Each example was constructed to work correctly. The user was not forced to encounter examples that failed to work properly. The tutorial assumed adaptive learning by the user and, therefore, only correct examples were presented. Presentation and discussion of ill-behaved examples were not included in the tutorial. Convenient stopping points were clearly indicated in the tutorial, relieving the user of the responsibility for finishing the tutorial in one session.

C. VERIFICATION

The examples presented in the tutorial for the user to

type were tested on the PDP-11/45 computer at The Rand Corporation. The information was accurate at the writing of this report. The RITA system has gone through several changes and the version of RITA used for the research was dated November, 1977.

Analysis of user feedback was minimal. The tutorial was used by three individuals. Only one was correctly termed "computer-naive." The other two users of the tutorial had a high degree of computer expertise. In the absence of sufficient user feedback, this author was unable to anticipate or meet the variety of user needs. Clearly, the tutorial has not been examined by a sufficient number of users to determine its effectiveness in introducing "computer-naive" users to the RITA system.

VI. APPLICATIONS

The RITA system has been designed to be widely applicable in two general areas. The first area was as a front-end to a remote computing system [Anderson and others, 1977]. Specifically, RITA programs would handle communication protocols and information transfer for the user. The second area was as a support tool for administrative and command functions [Anderson and others, 1977].

Several possible applications for the RITA system follow:

(1) There has existed a need in the U.S. Navy for automatic message routing between communication centers. The RITA system, implemented in an intelligent terminal, would be useful as a tool at each Naval communication center. RITA programs could be written to handle message sorting and transfer. The communication personnel at the communication center would be responsible for typing the messages into the RITA system. RITA programs could be written to determine addresses, open communication channels, send messages, and maintain status of messages sent and received. RITA programs could be written to determine alternate routes or alternate frequencies for message transfer.

(2) The commander at sea has lacked sufficient information about the status of the ships in his command.

RITA programs could be written to provide status information for the commander. The individual ships would update information in their data base. The individual ships could then execute RITA programs to send the information to the at-sea commander or wait for the commander to query the individual ships' data bases. RITA programs would handle all the information transfer including communication protocol.

(3) Ships at sea have depended on the timely communication of information to function effectively. RITA programs could be written to operate as a communication management system. The data base of the RITA system on each ship could contain all radio frequencies for a given geographical area of the world. The user could ask the RITA system to deduce the proper frequency for communication reception and transmission based on the ship's geographical location. RITA programs could be written to establish communication based on the deduction by the RITA system of the proper frequency. At another level, there could be a set of RITA programs that maintained the status of communication centers to communicate at certain frequencies. The interaction between the RITA system onboard and the RITA system at a communication center could provide the ability for the RITA system onboard to automatically deduce a secondary frequency if necessary.

(4) Numerous command functions onboard ship have needed computer support. The NAVFORSTAT information and battle

efficiency exercise scores could be maintained and reported by the RITA system. The English-like grammar of the RITA system has been designed to permit easy updating of information in the data base by computer-naive persons. RITA programs could be written to send the required information to the appropriate administrative commander whenever critical data base information had changed.

VII. CONCLUSIONS

The goals of the research were to describe the nature of a specific production system (RITA) and to develop a tutorial document aimed at explaining the RITA system to a computer-naive user. The goals of the research have been met. However, an evaluation of the effectiveness of the tutorial as a teaching tool was not possible from the limited sampling of users.

The designers of the RITA system have provided a human-oriented interactive computer system by using production systems for program control and data base management and an English-like language for human interaction. The facility for creating and adding production rules has provided a simple method for incrementally adding user tasks. The explanation subsystems in RITA have allowed the user the ability to ask the RITA system why particular actions were performed.

The features of the RITA system that have permitted the user to communicate with user files and other computer systems have been extremely valuable. The power of the RITA system has been its nature of intelligent behavior. The ability of the RITA system to perform various tasks based on the contents of a dynamic data base has been the key to its broad appeal.

As a future research effort, the automation of the tutorial might prove worthwhile. The program could be

written using a "lower fork" to execute RITA while the executive level handled the tutorial. This effort may be possible without substantial revision to the contents of the tutorial.

APPENDIX A - RITA SYSTEM : A TUTORIAL INTRODUCTION

TABLE OF CONTENTS

ACKNOWLEDGEMENT

Dr. Gary Martins of The Rand Corporation and Mr. Curt Blais of the Naval Ocean Sea Systems Command (NOSC) have reviewed the rough draft of the tutorial. Their suggestions and comments have been greatly appreciated.

Lieutenant Fred Adams, USCG, has reviewed and used the rough drafts of the tutorial. His suggestions and comments have been extremely valuable.

I. PURPOSE OF THE RITA TUTORIAL

This tutorial will attempt to act as your guide in learning and using the RITA system. The tutorial assumes no prior knowledge of computer programming techniques and therefore has been written with clearness and simplicity as key goals. Three documents will be referenced during the discussion of how to use the RITA system. The references are RITA Reference Manual, UNIX for Beginners, and A Tutorial Introduction to the UNIX Text Editor. The references will complement this tutorial. None of the references will be essential to using this tutorial. However, Appendix A of the RITA Reference Manual should be useful to show the structure of the RITA language. The tutorial is designed to meet the needs of a user who must be able to understand, use, and modify simple RITA programs.

Instructions to the user are in capital letters and are separated from other text throughout the tutorial.

This tutorial does not necessarily reflect the opinions or policies of The Rand Corporation or any agencies of the Department of Defense. The opinions in this tutorial are those of the author alone.

II. EXPLANATION OF THE RITA SYSTEM

RITA is an acronym for Rule-directed Interactive Transaction Agent. The system was developed by The Rand Corporation to attempt to meet several philosophical as well as practical goals. The RITA system was designed as a stand-alone computing resource for local text manipulation, as a limited heuristic modeling tool, and as a front end to remote computing systems and networks. It should be useful for maintenance scheduling and control, command and control systems, intelligence collection and dissemination, and remote accessing of large data bases.

The RITA system is a set of programs written in the "C" programming language that can run under the UNIX operating system on minicomputers such as the PDP 11/45 and PDP 11/70. RITA makes available a language for writing rules and an environment in which those rules are interpreted. The user can write a set of IF-THEN rules which can operate on local information or on information received over communication links to external computer systems.

III. MOTIVATION FOR USING THE RITA SYSTEM

RITA consists of programs that are very English-like and, therefore, are easy to read and write. The language allows the individual to write programs in the manner in which he thinks about a problem. The language's English-like nature assists the programmer in writing programs that are easy for a colleague to read and understand. The debugging features of the RITA system provide a comfortable environment in which the user can locate and correct errors in his program. The explanation subsystems in the RITA system can explain to the user why a particular action was taken. The RITA system has been designed to be used by a computer-naive user and, as such, it has the capability of explaining its actions upon request.

Sections IV through VII of the tutorial show specific procedures to follow to connect a terminal to the computer at The Rand Corporation. If you are accessing RITA at another host computer the sections mentioned do not apply.

IV. CONNECTING THE TERMINAL TO A COMPUTER BY TELEPHONE

This tutorial assumes that the user of the RITA system does not have the system available locally and must connect his terminal to a remote computer system which has the RITA system available. The connection is made using the telephone, via the ARPANET, to link the user's terminal and the computer system. The example that follows is for a user with a Computer Devices Miniterm terminal. The RITA system discussed is available at The Rand Corporation. Modification may be necessary for accessing the RITA system by another terminal. The RITA system was designed to be accessed by a user with a modified Ann Arbor terminal. The Ann Arbor terminal was not available to this writer so the tutorial is aimed at a user having a crt terminal or hardcopy terminal without any special keys. Instructions to the user are in capital letters throughout the tutorial.

PLUG IN THE POWER CORD OF THE TERMINAL TO A 110 VOLT OUTLET.

SET THE FOLLOWING POSITIONS ON THE TERMINAL.

Power : On
Speed : 30
Mode : Full (Duplex)
Mode : Std (Terminal mode)
Parity : On (Error reset)
Parity : Even

DIAL THE NUMBER FOR THE TIP (TERMINAL INTERFACE MESSAGE PROCESSOR).

LISTEN FOR A HIGH-PITCHED SOUND.

PLACE THE TELEPHONE RECEIVER IN THE BACK OF THE TERMINAL.

(A busy signal or ringing for more than six times is an unsuccessful attempt at making a connection. Hang up and try again.)

TYPE,

e <cr>

The <cr> is a symbol for the carriage-return key on the terminal.

At the time of this writing the author only had access to RITA on the Rand system. If you have access to RITA on another system type the address of that host computer.

TYPE,

@o 199 <cr>

The "@" is the symbol to tell the ARPANET that a command follows. The "o" is the command to open a connection to a computer. The "199" is the address of the computer at Rand Corporation. The <cr> is the symbol for the carriage-return key on the terminal.

The response of the computer is "Trying..." followed by "Open" on the next line. The following example shows the

commands the user should type to access the computer at Rand Corporation:

DO NOT TYPE AGAIN,

e <cr>

WAIT FOR THE TERMINAL TO RESPOND WITH THE NAME OF THE TIP.

DO NOT TYPE AGAIN,

@o 199 <cr>

WAIT FOR THE WORD "Trying..." TO BE PRINTED AT THE TERMINAL.

WAIT FOR THE WORD "Open" TO BE PRINTED AT THE TERMINAL.

V. CORRECTING TYPOGRAPHICAL ERRORS

After opening the connection to the PDP-11 computer at Rand Corporation the command to erase a mistake is the number-sign (#). To erase more than one character type the number-sign (#) the same number of times as the number of characters you wish to erase. After typing the number-sign (#) to erase an error, continue typing as if no mistake had been made.

VI. LOG IN PROCEDURES FOR THE PDP-11

WAIT FOR THE COMPUTER TO ASK FOR THE USER NAME.

TYPE ON THE SAME LINE,

user-name <cr>

WAIT FOR THE COMPUTER TO ASK FOR THE PASSWORD.

TYPE ON THE SAME LINE,

password <cr>

The user-name and the password are unique for each user. (You will need these before you can proceed.) The computer response upon successful login is information about the computer system. The computer responds with the UNIX prompt-sign (%) to indicate it's ready to accept UNIX commands. (If the computer continues to ask for the user-name and password, type the information in until you are logged into the computer. The user-name and password must be typed in exactly right for the computer to recognize you.)

Let's retrace our path. The terminal was connected to the TIP with a telephone call. The TIP (Terminal Interface Message Processor) is a processor that connects a terminal to a computer. The TIP allows us to open a connection from our terminal to one of the computers on the ARPANET. We chose to open a connection to the PDP-11 computer at The

Rand Corporation where the RITA system resides. At Rand Corporation the set of programs that manages the PDP-11 computer is the UNIX operating system. An operating system is a collection of routines for supervising the sequencing of programs by a computer.

VII. USING THE UNIX OPERATING SYSTEM

The prompt-sign (%) for the UNIX operating system lets the user know that he is in the UNIX system and can issue UNIX commands.

TYPE,

```
% date <cr>
```

The UNIX command "date" is one of the commands available to you as a user of the UNIX operating system. For explanations of other commands available on the UNIX system use the reference UNIX for Beginners.

VIII. ACCESSING THE RITA SYSTEM

The RITA system is available under the UNIX system at Rand Corporation. To use the RITA system do the following:

TYPE,

```
% rita <cr>
```

The command "rita" puts the set of programs known as "rita" into the user's space so that he can write programs and issue commands which will be interpreted by the RITA system. The RITA system consists of three programs which are the UFE (User Front End), PARSER, and MON (Monitor). The dates indicate the version of each program that is running. The RITA prompt-sign is the asterisk (*). We are in the RITA system and can use all the capabilities of the RITA system.

IX. USING THE RITA SYSTEM

The RITA system interprets each command as you finish typing it. The semi-colon (;) is the terminator for each RITA command. It must be typed after each command. The RITA system will not respond until the semi-colon (;) followed by the carriage-return <cr> has been typed. When the RITA command has been typed correctly and the RITA system has performed the command, it will respond with the RITA prompt-sign, an asterisk (*). If the RITA system does not understand the command, it will respond with the improper command you typed and the words "syntax error". (A common error is to misspell a RITA command.) To get the RITA prompt-sign (*) after typing in an improper command, type the semi-colon (;) followed by the carriage-return <cr>. Remember, to correct a typographical error, type a number-sign (#). Occasionally, unexpected terminal behavior is encountered.

Problem : Program is looping and will not stop.
Solution : Type the DEL key several times.

Problem : Command being entered is incorrect.
Solution : Type the ctrl-esc key followed by a carriage-return <cr>.

Problem : You are unable to get a RITA prompt-sign (*).
Solution : Type the ctrl-esc key followed by a

carriage-return <cr>.

The typing of the ctrl-esc and the DEL keys several times is often necessary. If the procedures above do not solve the problem then press the @ key followed by the c key. The @c command followed by the carriage-return <cr> closes the connection to the computer on the ARPANET. Then you can open the connection to the computer again by following the procedures in the section of the tutorial titled CONNECTING THE TERMINAL TO A COMPUTER BY TELEPHONE. The first command would be to open a connection by typing @o 199 for the computer at Rand Corporation. If that fails, hang up the phone.

X. ENTERING A RITA OBJECT

Units of information in your RITA program are stored as objects. Each object has a name and an arbitrary number of attributes. The object-name must be an unquoted string of characters not starting with a digit and not containing blanks or any of the following: [] () <> {} & ? : , ; ' ". Each attribute must be an unquoted string of characters with the same restrictions as the object-name. Each attribute has a value. A value may be a quoted string of characters, a RITA number, an ordered list of values, or unknown. A quoted string of characters is an arbitrary arrangement of symbols, letters, and numbers enclosed in quotation marks. An unquoted string is not enclosed in quotation marks. The quoted string may contain blanks as well as any other characters on the terminal keyboard. The discussion of a value as a list of values will be postponed until later in the tutorial. The following examples will show some of the character strings allowed for object-names, attributes and values.

OBJECT-NAME	ATTRIBUTE	VALUE
ship	speed	20
plane1	payload	"Two bombs"
destroyer-name	naval-hero	"John Paul Jones"
meal.time	dinner*menu	NOT KNOWN

The following are not allowed:

OBJECT-NAME	ATTRIBUTE	VALUE
30ships	spruance class	fast&capable
"plane"	speec?	excess of 600 knots
captain's name	3rd	[Jones]

The object-names, attributes and values in the example above were unsatisfactory for the following reasons. An object-name cannot begin with a digit (3), contain a special character (") or contain a space. An attribute cannot contain a space, contain a special character (?) or begin with a digit (3). A value cannot be an unquoted string of characters. Let's enter an object, give the object an attribute, and assign a value to the attribute. Objects can be entered in two ways.

```
TYPE,
 * object ship   <cr>
 name is "John F. Kennedy";      <cr>
```

The object-name is "ship". The attribute of the object is "name". The value of the attribute is "John F. Kennedy". The semi-colon (;) terminates the RITA command and the RITA system can begin to process that command. The second method of entering objects is to use the create command.

```
TYPE,
 * create a plane whose name is "F-14 Tomcat";   <cr>
```

The object-name is "plane". The attribute is "name". The value is "F-14 Tomcat". A semi-colon (;) terminates the RITA command. This form provides for improved readability because the articles A, AN, and THE can be inserted anywhere without affecting the RITA command "create". Let's look at the two objects that have been entered in the RITA system. There are three methods for printing objects.

```
TYPE,
 * display object ship;  <cr>

WAIT FOR PRINTING AT THE TERMINAL TO STOP.

TYPE,
 * display object plane;  <cr>
```

"Display" is the RITA command to print information. The word "object" tells the RITA system that we want to look at an object. "Ship" is the object-name in the first RITA command and "plane" is the object-name in the second RITA command. The semi-colon (;) terminates input of the RITA command.

```
TYPE,
 * display object ship and display object plane;   <cr>
```

Rita commands can be combined by an "and". They may extend for more than one line. The semi-colon (;) lets the RITA system know when it should perform the RITA commands.

```
TYPE,
```

```
* display all objects;     <cr>
```

"All" is a word in the RITA system that is used to indicate that the command applies to every object.

XI. ENTERING A RITA RULE

Most RITA programs are comprised of objects and rule-sets. A rule set is a collection of IF-THEN statements (rules). Each rule has a name, a set of premises, and a set of actions. The IF part of a rule consists of a set of premises. All the premises must be true for the rule to be true. The THEN part of a rule consists of a set of actions. The actions are performed sequentially when the IF part is true. The name of a rule can be an unquoted string of characters not starting with a digit and not containing blanks or any of the following: [](){}<>&?:,;'". The name of a rule can also be a quoted string of characters. There are two methods for entering rules.

TYPE,

```
 * rule carrier   <cr>
  if the name of the ship is "John F. Kennedy"    <cr>
  then set the captain of the ship to "Aviator";    <cr>

 * rule fighter   <cr>
  if the name of the plane is "F-14 Tomcat"    <cr>
  then set the pilot of the plane to "Ace"    <cr>
  and RETURN SUCCESS;    <cr>
```

The first RITA rule has the name "carrier". "If" is a reserved word to indicate the beginning of the IF part of the rule. A reserved word is a word used by the computer

system that has a special meaning and must be used by the user in the same context. "Name" is the attribute of object "ship". The value of the attribute is "John F. Kennedy". "Then" is the reserved word to indicate the beginning of the THEN part of a rule. "Set" is an action word that adds or modifies information about objects. "Captain" is an attribute of object "ship". The value of the attribute is "Aviator". The semi-colon (;) terminates the RITA command. Rule "fighter" is in the same format as rule "carrier". The words "RETURN SUCCESS" in rule fighter have special meaning to the RITA system. The reason for including them will be explained later in the tutorial. There is no restriction on the number of premises or actions in a rule. Each rule should be lengthy enough to perform a function. However, shorter rules are usually less complicated and easier to read. The second method of entering rules is to create an immediate rule.

DO NOT TYPE,

```
 * if there is a ship whose name is "John F. Kennedy"   <cr>
 then set the captain of the ship to "Aviator";         <cr>
```

The immediate rule cannot be saved in an external file for a later RITA session. The immediate rule tests the premises only once and executes the actions if all the premises are true. The immediate rule should be used when the user desires to test the rule only once and does not wish to save the rule for later use. The first method of

entering rules is preferred under most circumstances. Let's print the rules in our program. (If the RITA system at any time stops printing at the terminal and responds with the word "pausing", press the carriage-return <cr> key to cause the printing to continue. The command "pausing" should be expected occasionally.)

```
TYPE,
 * display all rules;      <cr>

TYPE,
 * display rule carrier and display rule fighter;     <cr>
```

The display command is used to print rules just as it was used to print objects. Let's print all the objects and all the rules.

```
TYPE,
 * display all objects and display all rules;      <cr>
```

Objects ship and plane and rules carrier and fighter comprise our entire program.

XII. DISPLAYING RITA OBJECTS AND RITA RULES TO EXTERNAL FILES

To save our program, we put it into an external file. A file is a place to store a collection of information where the order is maintained. External is a term for something outside the RITA system. The external file is outside the RITA system and is a file in the UNIX system. In the examples that follow, replace the author's initials "tew" with your own.

TYPE,

```
* display all objects to file navy-blue.tew;     <cr>
* display all rules to file navy-blue.tew;       <cr>
```

The first Rita command puts all the objects into a file which has a filename of "navy-blue.tew". The second RITA command puts all the rules into the same file which has a filename of "navy-blue.tew". The UNIX file "navy-blue.tew" contains object ship, object plane, rule carrier and rule fighter. The information is in file "navy-blue.tew" in the order that we entered the objects and entered the rules. Notice that the filename "navy-blue.tew" contains no blank spaces and is one continuous name. File "navy-blue.tew" is a permanent record of our program. Leaving the RITA session or breaking the telephone connection will not destroy our file "navy-blue.tew". Our program exists in the RITA system as well. The display command just made a copy of the objects and rules in the external file "navy-blue.tew". No

information was altered in our program in the RITA system.

XIII. CREATING A UNIX SHELL PROGRAM

A "shell" is a UNIX program that reads and interprets UNIX commands. The reason for creating a shell in UNIX is to allow the use of facilities outside the RITA system. We can create a shell, temporarily leave the RITA system, perform tasks in the UNIX system, then return to the RITA system.

TYPE,

```
* shell;      <cr>
```

The RITA command "shell" causes a shell program in the UNIX system to be created. We temporarily leave the RITA system and can issue UNIX commands. The UNIX prompt-sign (%) indicates that we are in the UNIX system.

XIV. VIEWING AN EXTERNAL FILE

Our external file "navy-blue.tew" is available in the UNIX system. Occasionally, we desire to make changes to the file or to print the information in the file. The reference A Tutorial Introduction to the UNIX Text Editor discusses in length how to make changes to UNIX files. (Note: UNIX commands do not end with the semi-colon (;). RITA commands do end with the semi-colon (;).)

TYPE,

```
% ls      <cr>
```

The UNIX command "ls" is the command to list the names of all our files. The unix command "ls" is useful when we have several files and can't remember the exact spelling of the file we wish to review. Other UNIX commands are discussed in the reference UNIX for Beginners.

XV. EDITING A UNIX FILE

After typing the UNIX command "ls", the UNIX system responds with the name of our only file, "navy-blue.tew". Let's make several changes to our file. Replace the initials "tew" with your own.

TYPE,

```
% ed navy-blue.tew    <cr>
```

The UNIX command "ed" makes the standard UNIX text editor available to the user. An editor is a program that can make additions, deletions, and changes to a UNIX file and print portions of that UNIX file at the terminal. Actually, our UNIX file "navy-blue.tew" has been put into the working space of the UNIX editor. The UNIX system responds with the number of characters in our file (344). Let's print all the lines in our file.

TYPE,

```
1,$p    <cr>
```

The "1" tells the editor to start at line one (1). The "$" tells the editor to stop at the last line in the file. The comma (,) separates the first line from the last line. The "p" is an editor command to print. The "1,$p" command causes the editor to print lines one (1) through the end of our file at the terminal.

WAIT FOR PRINTING AT THE TERMINAL TO STOP.

TYPE,

4p <cr>

The editor command "4p" causes line four (4) of our file to be printed at the terminal.

TYPE,

4s/ship/boat/p <cr>

The editor command "s" causes the substitution for the old characters "ship" by the new characters "boat". The slant lines (/) are separators. The "p" is an editor command to print the line at the terminal. The characters "ship" have been changed to "boat".

TYPE,

.w <cr>

The editor command "w" stands for write and is the command to make the change permanently to our file. Until the "w" (write) command is used, the changes are only temporary and will be lost if the user leaves the editor.

TYPE,

4p <cr>

TYPE,

4s/boat/ship/p <cr>

The editor command "4p" prints line four (4) of our

file. The "s" (substitute) command causes the old characters "boat" to be replaced by the new characters "ship". The slant lines (/) are separators. The "p" is an editor command to print the line at our terminal. Let's make the change permanent to our file.

TYPE,

w <cr>

Our file "navy-blue.tew" has been modified by changing the characters "ship" in line four (4) to "boat". The file has been returned to its original condition by changing the characters "boat" back to "ship". Leaving the RITA system by creating a UNIX shell program, then calling and using the UNIX text editor consumes a great amount of time. Changes that are not extensive can best be performed in the RITA system itself. A RITA command "delete" will be introduced that allows objects and rules to be eliminated. The eliminated object or rule could then be typed into the RITA system again with the changes.

TYPE,

q <cr>

The editor command "q" is for quit. The "q" (quit) command causes the user to leave the UNIX editor and return to the UNIX shell program. Notice the UNIX prompt-sign (%).

XVI. RETURNING TO THE RITA SYSTEM

(Remember the ctrl key is one key and not four separate ones.)

TYPE,

```
% ctrl-d
```

(Press the d key while simultaneously holding down the ctrl key.) The UNIX command "ctrl-d" causes us to leave the UNIX shell program and return to the RITA system. Notice the RITA prompt-sign (*).

XVII. DELETING OBJECTS AND RULES IN THE RITA SYSTEM

Until a RITA session is terminated, all the objects and rules entered in the RITA system are still present. RITA objects and RITA rules can be erased from our RITA program by using the RITA command "delete".

```
TYPE,
 * display all objects and display all rules;      <cr>
```

The RITA command "display" causes the objects and rules in our RITA program to be printed at the terminal.

```
TYPE,
 * delete object ship;        <cr>
 * delete rule carrier;       <cr>
```

The RITA command "delete" erases the RITA object or RITA rule specified from our RITA program.

```
TYPE,
 * display all objects and display all rules;      <cr>
```

Object "ship" and rule "carrier" are no longer part of our RITA program.

```
TYPE,
 * delete all objects and delete all rules;      <cr>
```

The RITA command "delete" used with the word "all" permits us to erase all the objects and all the rules in our

RITA program.

TYPE,

```
 * display all objects and display all rules;      <cr>
```

There are no objects or rules that remain as a part of our RITA program. Using the "delete" command is an effective way to eliminate incorrect or unwanted RITA objects or RITA rules. However, the RITA command "delete" should be used cautiously.

IF YOU WISH TO TERMINATE THE RITA SESSION AT THIS TIME TURN TO THE SECTION OF THE TUTORIAL TITLED LEAVING A RITA SESSION.

XVIII. LOADING AN EXTERNAL FILE

All the objects and rules have been deleted from our RITA program. However, the objects and rules are in the UNIX file "navy-blue.tew" and can be put into the RITA system without our retyping them. Supply your initials instead of the author's initials, i.e., tew.

TYPE,

```
* load navy-blue.tew;      <cr>
```

The RITA command "load" causes the information contained in file "navy-blue.tew" to be placed into the RITA system. The file is checked to ensure that the objects and rules are correctly written. The semi-colon (;) terminates the RITA command. The RITA system responds that the objects and rules were correct and have been added to the RITA system.

TYPE,

```
* display all objects and display all rules;      <cr>
```

There is an alternate method for loading an external file. The "fload" command can be used in exactly the same manner as the "load" command. The difference is that the file is not checked for correctness when the "fload" command is used. The safest command to use is the "load" command.

DO NOT TYPE,

```
* fload navy-blue.tew;      <cr>
```

The RITA system responds to the "fload" command exactly as it responds to the "load" command. However, no information about the objects or rules is provided.

XIX. DEBUGGING INFORMATION

Debugging is the process of finding and correcting errors in programs. Several very useful debugging commands are available in RITA to assist the user in finding and correcting program errors. The first debugging command we will use is the "trace" command.

TYPE,

```
* trace all rules;   <cr>
```

The RITA system responds with the RITA prompt-sign (*) to let us know that our command has been understood. Throughout the remainder of the tutorial the system response to a RITA command will usually be the RITA prompt-sign (*). When the system response differs it will be so indicated. We've asked the RITA system to trace all the objects and rules in our program. Let's display them to refresh our memory as to what they were.

TYPE,

```
* display all rules;   <cr>
```

An alternative method of tracing the rules in our RITA program follows:

TYPE,

```
* trace rule carrier and trace rule fighter;     <cr>
```

The RITA command "trace" can also be used to trace a

single object, attribute or rule. Thus, the RITA command "trace" is a command that allows the user to observe the operation of his RITA program.

TYPE,

```
 * set trace 4;      <cr>
 * run;              <cr>
```

More detail about the RITA program is available by using the "set trace" command. The RITA command "set trace" allows the user to obtain more debugging information if desired. The levels of information available range from a low of zero (0) to a high of four (4). The normal mode is level zero (0). The RITA command "run" causes the RITA system to test each rule and to perform the actions for each rule found to be true.

TYPE,

```
 * set trace 0;      <cr>
 * trace all rules;  <cr>
```

WAIT FOR PRINTING AT THE TERMINAL TO STOP.

TYPE,

```
 * run;              <cr>
```

The RITA command "set trace 0" asks the RITA system to provide the least debugging information available. The RITA command "trace" asks the RITA system to print information about the rules when their actions are performed. The RITA

command "run" is a command to start looking at the If part of each rule and determine if the premises are true. The "run" command also causes the Then part of each rule with a set of true premises to be performed. The rules are executed sequentially until all rules are determined to be false or the RITA action word "return" is encountered as part of the Then part of a true rule. (Note: The RITA system will continue to cycle through the rules forever if the action word "return" is not encountered in a true rule. The word "RETURN" was included in rule fighter to prevent the system from cycling through the rules indefinitely. The word "SUCCESS" was used to have something printed at the terminal. If all the rules are false on any one pass the system will also stop.)

The debugging information available about our RITA program is quite substantial. The RITA command "set trace 4" is more useful than the RITA command "set trace 0" for the user when he is trying to locate and correct program errors. The RITA command "set trace" determines the level of debugging information provided by the RITA system. The RITA command "trace" is the command that causes the RITA system to print the debugging information. Both RITA commands "set trace" and "trace" are used prior to issuing the RITA command "run". The RITA command "untrace" causes the RITA system to provide no announcement when the RITA rules are performed. The RITA command "untrace" has the opposite funcion of the RITA command "trace".

```
TYPE,
 * untrace all rules;      <cr>

WAIT FOR PRINTING AT THE TERMINAL TO STOP.

TYPE,
 * run;                   <cr>
```

The debugging facilities in the RITA system allow the user to observe the operation of his RITA program by using the RITA commands "trace" and "untrace". The debugging facilities in the RITA system allow the user to also intervene in the operation of his RITA program by using the RITA commands "stop" and "unstop".

```
TYPE,
 * set trace 4;           <cr>
 * trace all rules;       <cr>

WAIT FOR PRINTING AT THE TERMINAL TO STOP.

TYPE,
 * stop at all rules;     <cr>

WAIT FOR PRINTING AT THE TERMINAL TO STOP.

TYPE,
 * run;                   <cr>
```

The RITA command "set trace 4" sets the level of debugging information high. The RITA command "trace all rules" causes the RITA system to announce each rule that is

found true when the actions are performed. The RITA command "stop at all rules" causes the RITA system to suspend execution of the "run" command before the actions of each true rule are performed. The RITA command "run" causes each rule to be tested and the actions of the rules found to be true are performed.

```
TYPE,
 * continue;     <cr>
 * run;          <cr>
```

The RITA command "continue" is used with the RITA command "stop" to cause the RITA system to resume performing the actions of the current rule. In place of issuing the "continue" command the user could type any other RITA command. The RITA command "unstop" reverses the process that the RITA command "stop" caused.

```
TYPE,
 * unstop all rules;     <cr>
 * run;                  <cr>
```

The RITA command "unstop" causes the RITA system to allow no intervention in the operation of the user's RITA program.

```
TYPE,
 * set trace 0;          <cr>
 * untrace all rules;    <cr>
```

The RITA system debugging commands have been reset to their normal modes. The following are examples of other uses of the debugging commands:

DO NOT TYPE,

```
 * trace rule carrier;                          <cr>
 * trace object ship;                           <cr>
 * trace goal victory;                          <cr>
 * trace all rules;                             <cr>
 * trace all objects;                           <cr>
 * trace all goals;                             <cr>
 * trace all rules that set name of ship;       <cr>
 * trace all rules that test name of plane;     <cr>
 * trace all goals that set name of ship;       <cr>
 * trace all goals that test name of plane;     <cr>
 * trace name of plane;                         <cr>
```

The RITA command "trace rule carrier" is used when a single rule with a specific rule-name (i.e. carrier) is to be traced.

"Trace object ship" is used to trace a single object with a specific object-name (i.e. ship).

"Trace goal victory" is used to trace a single goal with a specific goal-name (i.e. victory). Goals will be discussed later in the tutorial.

"Trace all rules" is used to trace each rule in the RITA program. "Trace all objects" and "trace all goals" are RITA

commands used to trace each object and trace each goal respectively.

"Trace all rules that set name of ship" is used to trace all rules that cause the attribute (i.e. name) of the object (i.e. ship) to be entered or changed in our RITA program. "Trace all rules that test name of plane" is used to trace all rules that check the attribute (i.e. name) of the object (i.e. plane) in our RITA program.

"Trace all goals that set name of ship" is used to trace all goals that cause the attribute (i.e. name) of the object (i.e. ship) to be entered or changed in our RITA program. "Trace all goals that test name of plane" is used to trace all goals that check the attribute (i.e. name) of the object (i.e. plane) in our RITA program.

"Trace name of plane" is used to announce the specified object (i.e. plane) when its attribute (i.e. name) is changed.

All the debugging commands are used with the "run" command. They provide facilities for observing and intervening in the operation of a RITA program. Therefore, the RITA commands "trace," "untrace," "stop at," "unstop" and "set trace" are valuable tools for locating and correcting program errors.

TYPE,

* delete all objects and delete all rules; <cr>

IF YOU WISH TO TERMINATE THE RITA SESSION AT THIS TIME TURN TO THE SECTION OF THE TUTORIAL TITLED LEAVING A RITA SESSION.

XX. EXPLANATION OF THE LHS (LEFT HAND SIDE) MONITORS

The RITA system consists of a set of three programs. One of the three programs is the monitor. A monitor is a computer program that evaluates a set of rules by testing the premises (If part) and performing the corresponding actions (Then part) for all rules that are true. There are two left-hand-side monitors, the ordered and the cyclic, in the RITA system. When either the ordered or the cyclic monitor is used, it will test rule premises and perform the actions of all true rules.

The cyclic, or unordered, monitor begins testing rules at the top of the rule-set list. When one rule is found to be true, the actions of that rule are performed. The cyclic monitor tests the next rule in the list. After checking the last rule in the rule-set list, the cyclic monitor begins again at the top of the rule-set. The process continues until all rules are found to be false in one pass or the RITA word "return" is encountered in the action part of a true rule.

The ordered monitor behaves exactly the same as the cyclic monitor except the next rule tested after finding a true rule is the first one in the rule-set list. Ordered in this sense means that when one rule is found to be true, the monitor should start again at the beginning of the rule-set.

Let's build two small programs to illustrate the differences between the cyclic and ordered monitors. (If

you type in a rule or object incorrectly and the system responds "syntax error", type the semi-colon (;) and type a carriage-return <cr>. The RITA prompt-sign (*) should appear. Now type in the RITA command correctly.)

```
TYPE,
 * object ship                    <cr>
   name is "JFK";                  <cr>
 * object plane                   <cr>
   name is "A-7";                  <cr>
 * object squadron                <cr>
   name is "fighter";              <cr>
 * rule cyclic1                   <cr>
  If the name of the ship is "JFK"      <cr>
  Then set the name of the ship to "JFK";      <cr>
 * rule cyclic2                   <cr>
  If the name of the plane is "F-14"           <cr>
  Then RETURN SUCCESS;                           <cr>
 * rule cyclic3                 <cr>
  If the name of the squadron is "fighter"     <cr>
  Then set the name of the plane to "F-14";    <cr>

TYPE,
 * display all objects and display all rules;   <cr>

TYPE,
 * display all objects to file cyclic.tew;          <cr>
 * display all rules to file cyclic.tew;            <cr>
```

All the objects and rules in our RITA program have been put into an external UNIX file named "cyclic.tew". The cyclic or unordered monitor is called by using the RITA commands "set unordered" and "run".

```
TYPE,
 * set trace 4;        <cr>
 * trace all rules;    <cr>

WAIT FOR PRINTING AT THE TERMINAL TO STOP.

TYPE,
 * set unordered;      <cr>
 * run;                <cr>
```

The RITA command "set trace 4" sets the level of debugging information to be provided to the user. "Trace all rules" causes the RITA system to announce each rule as it's tested. "Set unordered" causes the cyclic monitor to be used. "Run" causes the RITA system to test RITA rules and to execute the actions of the rules found to be true.

Let's follow the RITA program as the cyclic monitor tests and executes RITA rules. The If part of rule "cyclic1" is true because when we created the object ship, the value of the attribute (i.e. name) was set to "JFK". Rule "cyclic2" is false. The name of the plane is "A-7", not "F-14". When we created the object plane, we made the attribute name have a value "A-7". Rule "cyclic3" is true so the name of the plane is changed to "F-14".

The cyclic monitor starts again at the top of the rule-set. Rule "cyclic1" is true. Rule "cyclic2" is true because rule "cyclic3" set the name of the plane to "F-14" previously. The cyclic monitor stops because it encounters the RITA word "return" in a true rule "cyclic2".

```
TYPE,
 * set trace 0;    <cr>
 * untrace all rules;    <cr>
 * delete all objects and delete all rules;      <cr>
 * display all objects and display all rules;    <cr>
```

The second program will be used to show how the ordered monitor works. Remember, the ordered monitor tests the rule-set sequentially from the top until the first true rule is found. Its actions are performed and the ordered monitor returns to the top of the rule-set to continue testing. The ordered monitor stops when the RITA word "return" is encountered as an action in a true rule or when all the rules are false on one pass.

```
TYPE,
 * object ship            <cr>
  name is "JFK";          <cr>
 * object plane           <cr>
  name is "A-7";          <cr>
 * object squadron        <cr>
  name is "fighter";      <cr>
 * rule ordered1          <cr>
```

```
  If the name of the ship is "JFK"                <cr>
  Then set the name of the ship to "America"; <cr>
 * rule ordered2          <cr>
  If the name of the plane is "F-14"              <cr>
  Then set the name of the plane to "F-14";  <cr>
 * rule ordered3          <cr>
  If the name of the squadron is "fighter"    <cr>
  Then RETURN SUCCESS;                          <cr>

TYPE,
 * display all objects and display all rules;   <cr>
```

Notice that the program to demonstrate the cyclic monitor is different than the program above. Let's save our program by putting it into an external UNIX file named "ordered.tew".

```
TYPE,
 * display all objects to file ordered.tew;      <cr>
 * display all rules to file ordered.tew;        <cr>

TYPE,
 * set trace 4;        <cr>
 * trace all rules;    <cr>

WAIT FOR PRINTING AT THE TERMINAL TO STOP.

TYPE,
 * set ordered;        <cr>
 * run;                <cr>
```

The ordered monitor is called by using the "set ordered" and "run" commands. The ordered monitor tests rule "ordered1". The rule is true because the name of the ship is "JFK". The action of rule "ordered1" is to set the name of the ship to "America". The ordered monitor has found a true rule, performed its actions and returned to the top of the rule-set. Rule "ordered1" is tested again. This time it is false because the name of the ship is now "America", not "JFK". Rule "ordered2" is tested and is found to be false. The name of the plane is "A-7", not "F-14". Rule "ordered3" is tested and since the name of the squadron is "fighter", the action of the rule "ordered3" is performed. Because the RITA word "return" is part of the actions for rule "ordered3", the ordered monitor stops.

The LHS (Left Hand Scan) monitors, cyclic and ordered, are useful for situations where the information stored in the objects can change. The LHS (Left Hand Side) monitors are also referred to as pattern-matching or rule-directed monitors. The behavior of the monitors depends on matching a pattern (If part) and performing actions based on which rules are found to be true.

TYPE,

```
 * set trace 0;    <cr>
 * untrace all rules;    <cr>
 * set unordered;    <cr>
 * delete all objects and delete all rules;    <cr>
```

IF YOU WISH TO TERMINATE THE RITA SESSION AT THIS TIME TURN TO THE SECTION OF THE TUTORIAL TITLED LEAVING A RITA SESSION.

XXI. RUNNING A RITA PROGRAM

Running, or executing, a program means the process of carrying out the steps specified to produce the required results. We've run several programs already in our discussion of other RITA commands. The RITA command "run" has two forms.

```
TYPE,
 * load navy-blue.tew;        <cr>

WAIT FOR PRINTING AT THE TERMINAL TO STOP.

TYPE,
 * display all objects and display all rules;     <cr>
```

We've loaded our file "navy-blue.tew" in order to have a program to run.

```
TYPE,
 * run rule carrier;        <cr>
```

The first form of the "run" command is "run rule rule-name". The rule we wish to execute is rule "carrier". That one rule has been interpreted and executed. No other rules were affected by the "run rule rule-name" command. The second form of the "run" command causes all rules in the program to be interpreted and executed. If the particular LHS monitor is not indicated, the default monitor is the cyclic, or unordered, monitor.

TYPE,

```
* run;          <cr>
```

All rules in our RITA program have been interpreted and executed. Execution stops when all the rules in the rule-set are found to be false on one pass or the RITA word "return" is encountered in the actions of a true rule.

TYPE,

```
* delete all objects and delete all rules;     <cr>
```

IF YOU WISH TO TERMINATE THE RITA SESSION AT THIS TIME TURN TO THE SECTION OF THE TUTORIAL TITLED LEAVING A RITA SESSION.

XXII. ENTERING AND DISPLAYING A RITA GOAL

RITA programs can contain goals as well as objects and rules. A RITA goal is a RITA rule that is used with the RHS (Right-Hand-Side) monitor only. THe RHS (Right-Hand-Side), or goal-directed, monitor differs from the LHS (Left-Hand-Side) monitors in that the goal-directed monitor checks the action part (Then part) of each RITA goal to determine the goals which will be useful in determining the information desired by the user. When the actions of a goal are determined to be useful, the premises (If part) of the goal are tested to see if all the premises are true. If so, the actions of the true goal are executed. Perhaps an example will show the nature of a RITA goal.

```
TYPE,
 * goal performance                          <cr>
  If the productivity of the ship is "excellent"     <cr>
  Then set the performance of the ship to "outstanding";<cr>
 * goal captain                              <cr>
  If the status of the ship is "underway"            <cr>
  Then set the captain of the ship to "happy";       <cr>
 * goal productivity                         <cr>
  If the morale of the ship is "high"                <cr>
  Then set the productivity of the ship to    <cr>
   "excellent";    <cr>
 * goal morale                               <cr>
```

```
If the captain of the ship is "happy"                    <cr>
Then set the morale of the ship to "high";               <cr>
```

RITA goals are entered into the RITA system in the same form as RITA rules. However, goals differ in that the RHS (Right-Hand-Side) monitor tests RITA goals until it finds the user-requested information, or until it determines that the information requested is not currently available. The RITA system in the latter case prompts the user to supply preliminary information needed for the goal-directed monitor to continue its search for the requested information. The goal-directed monitor is most useful in a static situation, where the information once determined will keep the same values.

```
TYPE,
 * display all goals to file monitor.tew;   <cr>
 * display all goals;                       <cr>
```

The "display all goals to file monitor.tew" command makes a copy of the goals in our RITA program and puts it into an external UNIX file named "monitor.tew". The command "display all goals" causes the RITA system to print the RITA goals at the terminal. The four goals make up our entire Rita program.

XXIII. DEDUCING A RITA GOAL

The goal-oriented monitor is called by using the RITA command "deduce". We must tell the RITA system what we want to be determined.

```
TYPE,
 * create a ship whose status is "underway";      <cr>
 * display object ship;     <cr>

WAIT FOR PRINTING AT THE TERMINAL TO STOP.

TYPE,
 * deduce the performance of the ship;            <cr>

WAIT FOR PRINTING AT THE TERMINAL TO STOP.

TYPE,
 * display object ship;     <cr>
```

The "create" form of entering an object was used to enter an object (i.e. ship) with an attribute (i.e. status) whose value is "underway". Goal "captain", when tested, will be true and the process of finding the performance of the ship can begin.

The RITA command "decuce the performance of the ship" causes the RITA system to start the goal-oriented monitor. The information requested is in the form attribute (i.e. performance) of the object (i.e. ship). The monitor will determine the value of the attribute (i.e. performance) of

the object (i.e. ship). Remember, the goal-oriented monitor does not test the goals sequentially. The action part (Then part) of each goal is checked to see if it applies. If so, the premises of the goal are tested. If the premises are all found to be true, then the actions of that goal are performed.

In our program the goal-oriented monitor checks the action parts of the goals until one applies. Goal "performance", if true, would provide the information requested. (What is the value of the performance of the ship?) However, the goal "performance" is false. The value of the "productivity of the ship" is unknown. The goal-oriented monitor then scans the actions parts of each goal until it can find a value for the "productivity of the ship". Goal "productivity", if true, would provide a value for the "productivity of the ship". Goal "productivity" is false. The value of the "morale of the ship" is unknown. The monitor scans the action parts until it finds a value for the "morale of the ship". Goal "morale", if true, would provide a value for the "morale of the ship". Goal "morale" is false. The value of the "captain of the ship" is unknown. The monitor checks until it finds a value for the "captain of the ship". Goal "captain" is tested and is found to be true. The status of the ship is "underway". Remember, we created an object (i.e. ship) with an attribute (i.e. status) whose value was "underway".

The goal-oriented monitor performs the actions of goal

"captain". The captain of the ship is set to happy. Goal "morale" is now true. The morale of the ship is set to high. Goal "productivity" is now true. The productivity of the ship is set to excellent. Goal "performance" is now true. The performance of the ship is set to outstanding. Now, our original question can be answered. What is the performance of the ship? The value of the performance of the ship is "outstanding".

If the value of the specified attribute of a particular object is already known, the deduction terminates. If no goals help in performing the deduction, the monitor will prompt the user for the correct value. If the value is known, the user can enter it. However, if the user does not know the value requested, he can type a question mark (?) or a carriage return <cr>. The value of the attribute will remain NOT KNOWN.

If the user wants to know why the RITA monitor prompted him, he can use the RITA command "why" followed by the semi-colon (;) and a carriage return <cr>. When a deduction has been completed and an attribute of a particular object has been deduced or found to be NOT KNOWN, any later "deduce" command for the same attribute of the same object will fail to have any effect. The RITA system will not attempt to perform the deduction. If you desire to deduce the same attribute of the same object again, the value of the attribute must be set to NOT KNOWN by the user.

TYPE,

* delete all objects and delete all goals; <cr>

We no longer need the objects and goals in our RITA program so we delete them.

IF YOU WISH TO TERMINATE THE RITA SESSION AT THIS TIME TURN TO THE SECTION OF THE TUTORIAL TITLED LEAVING A RITA SESSION.

XXIV. BUILT-IN FUNCTIONS

A number of built-in functions are available in the RITA system. A function is a RITA command with a special purpose. The purposes are performing arithmetic and string operations on values. The built-in functions give the user a specific piece of information. There are currently seventeen built-in functions in the RITA system. Each function will be discussed and an example of each function will be shown.

```
TYPE,
 * object missiles                <cr>
   quantity is 5;                 <cr>
 * display abs(quantity of missiles);      <cr>
```

An object was entered with an object-name of missiles. The attribute was quantity. The value of the attribute was 5. The RITA functions "abs" returns the absolute value of the value (i.e. 5). The value must be a RITA number. A RITA number is a string consisting of one or more digits, optionally including a decimal point, optionally preceded by a plus or minus sign, and having optional leading and trailing blanks. Its magnitude must be less than 30 digits long. The following are equivalent and proper RITA numbers:

```
5
+5
```

```
5.
+5.
5.0
+5.0
05
+05
05.
+05.
05.0
+05.0
```

If the value of the quantity of missiles had been -5, the "abs" function would have returned a five (5). Remember a RITA number is a string of digits (0-9). The string "five" is not equivalent to the string "5". The string "five" is not a RITA number.

```
TYPE,
 * display clock();  <cr>
```

The RITA function "clock" returns the current date and time as a string of characters. Notice the parentheses in the RITA function "clock". They contain nothing and are typed consecutively-left then right parenthesis-without a space.

```
TYPE,
 * object ship        <cr>
  color is "grey",       <cr>
  hue is "battleship";  <cr>
```

```
* set shade of ship to concat(hue of ship,"-",     <cr>
   color of ship);     <cr>
* display object ship;    <cr>
```

An object was entered named "ship". The attribute (i.e. color) had a value of "grey". The attribute (i.e. hue) had a value of "battleship". The RITA command "set" gave the object (i.e. ship) an attribute (i.e. shade). The value was put together as a string by the RITA function "concat".

Concatenation is a process of putting strings of characters together in a particular order. The RITA function "concat" took the value of the hue of the ship (i.e. battleship), put a dash (-) beside it and put the value of the color of the ship (i.e. grey) alongside those strings to form one string (i.e. battleship-grey). The value of the shade of the ship becomes "battleship-grey". The RITA command "display" demonstrates that the "concat" function was able to build the string "battleship-grey" and the "set" command put that value (i.e. battleship-grey) into the attribute (i.e. shade) of the object (i.e. ship).

DO NOT TYPE,

```
* display eval(shade, ship);       <cr>
```

The RITA function "eval" evaluates the attribute (i.e. shade) of the object (i.e. ship) and returns the value (i.e. battleship-grey). The first item must be an attribute (i.e. shade) and the second item must be an object (i.e. ship). If the value had not been set, the "eval" function would

have returned "NOT KNOWN". (The built-in function "eval" does not respond as expected at the time of the writing of this tutorial.)

```
TYPE,
 * object sailor                <cr>
   temperature is "98.6";       <cr>
 * display floor(temperature of sailor);   <cr>
```

The object (i.e. sailor) was entered with an attribute (i.e. temperature) with a value (i.e. 98.6). The "floor" function determines the value of the temperature of the sailor to see if the string is a RITA number. The "floor" function then returns the largest integer less than or equal to the RITA number. The value was "98.6". The value is a RITA number. The largest integer less than or equal to the RITA number (i.e. 98.6) is "98". The value "98" is returned by the "floor" function. If the value is not a RITA number, an error message is returned.

```
TYPE,
 * display object ship;        <cr>
```

WAIT FOR PRINTING AT THE TERMINAL TO STOP.

```
TYPE,
 * display index("grey","battleship-grey");     <cr>
```

The function "index" takes the two values and checks to see that they are strings of charcters. Then the "index"

function returns the number of the position of the start of the first string (i.e. grey) in the second string (i.e. battleship-grey). The first string "grey" is found in the second string "battleship-grey". The position in the second string (i.e. battleship-grey) where the first character (i.e. g) of the first string (i.e. grey) appears is position 12. Positions in a string of characters begin with character position 1.

```
TYPE,
 * display index(color of ship,shade of ship);      <cr>
```

The value of the color of ship is "grey". The value of the shade of the ship is "battleship-grey". The "index" function finds the first string (i.e. grey) in position 12 of the second string (i.e. battleship- grey) as before.

```
TYPE,
 * display islist(("A-7","F-4","F-14"));     <cr>
```

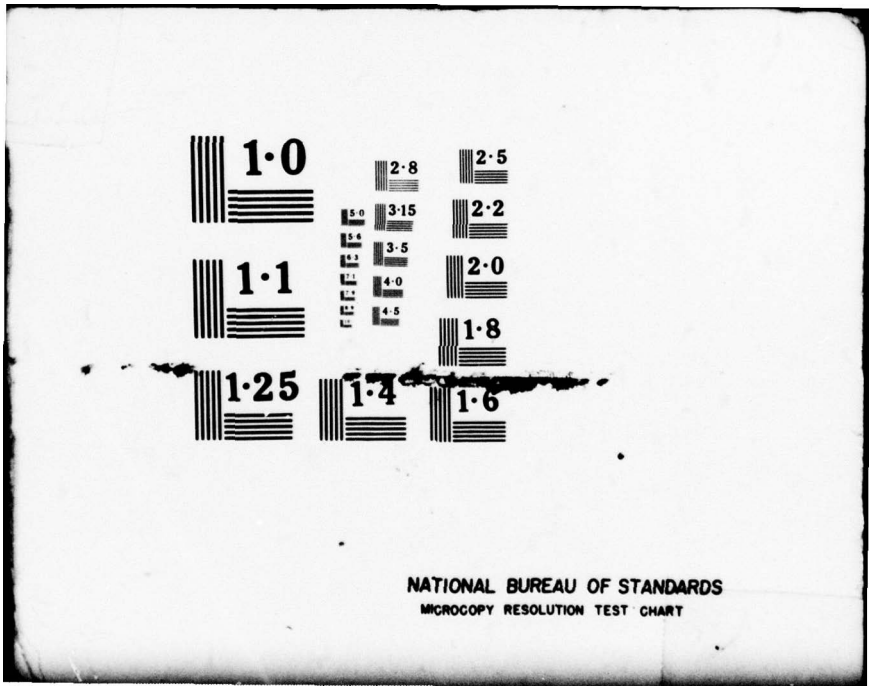
The function "islist" evaluates the items (i.e.("A-7","F-4","F-14")) to determine if the items are a list. A list is a left parenthesis, a number of values (strings or lists) separated by commas, and a right parenthesis. In the example, the first left parenthesis and the last right parenthesis enclose the items. The second left parenthesis and the next to the last right parenthesis are part of the list. The items "A-7", "F-4", and "F-14" are values. The commas separate the values in the list.

AD-A057 901 NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF F/G 9/2
A USER'S GUIDE FOR THE RITA PRODUCTION RULE SYSTEM.(U)
JUN 78 T E WARREN

UNCLASSIFIED NL

2 OF 2
ADA
057901

END
DATE
FILMED
10-78
DDC


1·0
1·1
1·25
1·4
1·6
1·8
2·0
2·2
2·5
2·8
3·15
3·5
4·0
4·5
NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART

The "islist" function returns "TRUE" if the items are a list and "FALSE" otherwise.

```
TYPE,
 * display islist("A-7","F-4","F-14");      <cr>
```

The items are not a list because a set of parentheses is missing. The one set of parentheses encloses the items belonging to the "islist" function. The function "islist" properly returns the error.

```
TYPE,
 * display object missiles;    <cr>
```

WAIT FOR PRINTING AT THE TERMINAL TO STOP.

```
TYPE,
 * display isnum(quantity of missiles);     <cr>
```

The function "isnum" determines if the value (i.e. 5) of the attribute (i.e. quantity) of the object (i.e. missiles) is a RITA number. The function "isnum" returns the word "TRUE" if the value is a RITA number, otherwise, it returns "FALSE".

```
TYPE,
 * object plane          <cr>
   speed is "SUPERSONIC";      <cr>
 * display lc(speed of plane); <cr>
```

We entered an object (i.e. plane) with an attribute

(i.e. speed) with a value (i.e. SUPERSONIC). The function "lc" stands for lower case. It evaluates the value and determines whether it is a string of characters or not. If the value is a string of characters, each capital letter in the string is returned as a small letter.

```
TYPE,
 * display length(speed of plane);    <cr>
```

The function "length" evaluates the value (i.e. SUPERSONIC). If it is a string, the "length" function returns the number of characters in the string (i.e. 10).

```
TYPE,
 * display lindex("F-4",("A-7","F-4","F-14"));  <cr>
```

The function "lindex" takes the first string (i.e. "F-4") and locates the position of that string of characters in the second item which must be a list. The function "lindex" finds that the first string of characters (i.e. "F-4") is the second member of the list. "A-7" is the first member, "F-4" is the second member, and "F-14" is the third member of the list. If the first string of characters does not occur as a member of the list, the function "lindex" returns "FALSE". If the first string of characters does appear as a member of the list, the function "lindex" returns its position in the list.

```
TYPE,
 * display max(quantity of missiles,    <cr>
```

```
    temperature of sailor);     <cr>
```

The "max", maximum, function determines if the values of all the items are RITA numbers. If each value (i.e. 5, 98.6) is a RITA number, the "max" function returns the largest RITA number (i.e. 98.6).

```
TYPE,
 * display min(quantity of missiles,     <cr>
    temperature of sailor);     <cr>
```

The "min", minimum, function determines if all the values of the items are RITA numbers. If each value (i.e. 5, 98.6) is a RITA number, the "min" function returns the smallest RITA number (i.e. 5).

```
TYPE,
 * display mod(10,3);          <cr>
```

The "mod" function evaluates value one (i.e. 10) and value two (i.e. 3) to determine that both are RITA numbers. Value one (i.e. 10) is divided by value two (i.e. 3). The remainder (i.e. 1), is returned by the "mod" function.

```
TYPE,
 * display object missiles and display object ship; <cr>

WAIT FOR PRINTING AT THE TERMINAL TO STOP.

TYPE,
 * display nsubstr(12, 4, "battleship-grey");          <cr>
```

The "nsubstr" function finds a string of characters in a target string. The first item (i.e. 12) should evaluate to a positive integer - whole number larger than zero. The second item (i.e. 4) should evaluate to a non-negative integer - whole number larger than or equal to zero (0). The third item (i.e. battleship-grey) is the target string. The positive integer (i.e. 12) gives the starting position of the string of characters to be located in the target string. The non-negative number (i.e. 4) gives the length of the string of characters desired. Position 12 in the target string is the character "g". The four (4) characters requested are "g r e y". They are positions 12, 13, 14, 15 in the target string (i.e. battleship-grey). If the starting position requested exceeds the length of the target string, the "nsubstr" function returns the empty string ("").

```
TYPE,
 * display sused();          <cr>
```

The "sused" function returns the amount of memory used so far. Memory is any device into which a unit of information can be copied, retained, and retrieved at a later time. The units of memory are words. A word is a set of characters which occupies one storage location. The word on the PDP-11 at Rand Corporation is two characters long. The "sused" function returns the amount of memory (in K words) used so far. K is a term for 1024.

TYPE,

 * display object ship; <cr>

WAIT FOR PRINTING AT THE TERMINAL TO STOP.

TYPE,

 * display uc(shade of ship); <cr>

The "uc" function evaluates the item (i.e. shade of ship) to see if the value (i.e. battleship-grey) is a string of characters. The "uc", upper case, function returns a capital letter for each small letter in the string.

XXV. ARITHMETIC OPERATORS

A RITA arithmetic operator is a basic action to be performed on two RITA numbers. The four arithmetic operators in the RITA system are + - * /, which represent addition, subtraction, multiplication, and division. Multiplication and division take precedence over addition and subtraction, as in normal mathematical usage. To override this precedence, the user may group terms with angle brackets (<>). An arithmetic operator must be preceded and followed by a blank space.

```
TYPE,
 * display 3 + 2 ;          <cr>
```

The "display" command prints the information at the terminal. The arithmetic operator (i.e. +) adds value one (i.e. 3) and value two (i.e. 2). Remember both values must be RITA numbers.

```
TYPE,
 * display object missiles;     <cr>

WAIT FOR PRINTING AT THE TERMINAL TO STOP.

TYPE,
 * display quantity of missiles - 2;      <cr>
```

The arithmetic operator subtraction (-) determines if the value (i.e. 5) of the quantity of missiles and the 2

are RITA numbers. Since both (i.e. 5, 2) are RITA numbers, the subtraction is performed and the RITA number (i.e. 3) is returned.

```
TYPE,
 * set the quantity of missiles to    <cr>
    quantity of missiles * 2;    <cr>
 * display quantity of missiles;        <cr>
```

The "set" command changes the value (i.e. 5) of the quantity of missiles to the value of the quantity of missiles multiplied by 2. Notice the value of the quantity of missiles has been changed to 10. The arithmetic operator multiplication (*) took the first value (i.e. 5) and multiplied it by the second value (i.e. 2).

```
TYPE,
 * display quantity of missiles  / <4 - 2>;        <cr>
```

The pair of angle brackets (<>) causes the arithmetic expression (i.e. 4 - 2) to be evaluated first. The arithmetic operator division (/) determines the value of the quantity of the missiles to be 10 and divides it be the value of 2 (i.e. 4 - 2). The RITA number 5 is returned.

XXVI. MATCHING A PATTERN IN A STRING OF CHARACTERS

The RITA system has the capability to compare strings of characters to determine if one of the strings of characters is identical or contained in another string of characters. For example, the feature is essential for determining if the current value of a specified attribute will make the premise of a rule true. The string of characters checked by the monitor must be exactly the same as the string of characters that comprise the specified value.

Often we wish to have our RITA program check for strings of characters. However, all of the characters are not always known. The RITA system has a feature whereby the user can give it a pattern specification. A pattern specification is a precise arrangement of characters. The RITA system can then, without knowing the exact string of characters, locate an arrangement of characters - a pattern. A pattern specification may be used in the premise (If part) of a rule or goal. The RITA words "contains" or "does not contain" must also be part of the pattern specification.

```
TYPE,
 * delete object plane;     <cr>
 * delete object missiles;     <cr>
 * delete object sailor;     <cr>
 * display object ship;     <cr>

WAIT FOR PRINTING AT THE TERMINAL TO STOP.
```

```
TYPE,
 * rule pattern                                  <cr>
  If the shade of the ship contains             <cr>
  {anything followed by "grey"}                 <cr>
  Then set the service of the ship to "Navy"    <cr>
  and RETURN SUCCESS;                           <cr>
```

The pattern specified is "anything", which means exactly that. The pattern to be searched can begin with any character or characters. The RITA words "followed by" indicates that there is at least one more pattern to be matched. The word "grey" enclosed by quotation marks is an exact string of characters to be matched (i.e. grey). The curly braces enclose the pattern specification. The word "contains" tells the RITA system that the premise should be true if the following pattern is matched. The words "does not contain" would have been used if the desire was to have the premise be true for cases other than when the pattern was matched.

```
TYPE,
 * run rule pattern;      <cr>
```

WAIT FOR PRINTING AT THE TERMINAL TO STOP.

```
TYPE,
 * display object ship;        <cr>
```

The rule "pattern" was executed. The value of the shade of the ship was found to be "battleship-grey". The premise

(If part) of the rule was true. The pattern anything followed by "grey" was matched. The action of the rule was performed (i.e. the value of the service of the ship was set to "Navy".)

```
TYPE,
 * object letter                    <cr>
  zip-code is 93940;                <cr>
 * rule mail                        <cr>
  If the zip-code of the letter     <cr>
  contains {"93940"}     <cr>
  Then set the destination of the letter to     <cr>
    "Monterey"     <cr>
  and RETURN SUCCESS;               <cr>
```

The pattern specification in the example above required the value (i.e. 93940) of the zip-code of the letter to be matched exactly.

```
TYPE,
 * rule route-mail                   <cr>
  If the zip-code of the letter     <cr>
  contains { "93" followed by some          <cr>
  in "56789" followed by end}     <cr>
  Then set the destination of the letter to "Cal"    <cr>
  and RETURN SUCCESS;               <cr>
```

The pattern specification was the two numeric characters "93" followed by any numeric characters in the range 5

through 9. Then the next pattern could be any string of characters. The rule would check to see if the first two characters were "93". Then it would require at least the next character to be a number between 5 and 9. Then the rule would match any characters after it found the last successive character in the range 5 through 9. The word "end" will cause the last characters to be matched. So, the following zip-codes would be matched:

93500

93940

93678

The following zip-codes would not be matched:

93499

93078

93356

The pattern specification in a rule or goal may be very general or quite specific. The important idea in using a pattern specification is the fact that patterns can be matched by knowing only their characteristics. The exact string of characters may not be known. The other ways of building pattern specifications are numerous and will not be discussed in the tutorial.

TYPE,

```
 * delete all objects;        <cr>
 * delete all rules;          <cr>
```

```
* delete all goals;                    <cr>
```

The objects, rules, and goals in our RITA program are no longer needed so we've deleted them.

IF YOU WISH TO TERMINATE THE RITA SESSION AT THIS TIME TURN TO THE SECTION OF THE TUTORIAL TITLED LEAVING A RITA SESSION.

XXVII. ACCESSING OBJECTS

Information about the user's RITA program is kept in the RITA objects. The object-name, attributes, and their values are also kept with the objects. The information about the objects must be entered, changed, or removed occasionally to keep the information current and correct. There are several RITA commands to update object information.

```
TYPE,
 * create a ship whose number is 66;        <cr>
 * display object ship;     <cr>
```

The "create" command enters an object (i.e. ship). The object has an attribute (i.e. number). The attribute has a value (i.e. 66).

```
TYPE,
 * set the number of the ship to 67;        <cr>
 * display object ship;                     <cr>
```

The "set" command allows the user to change the value (i.e. 66) of the number of the ship to 67.

```
TYPE,
 * put "America" into name of ship as first member;      <cr>
 * display object ship;                                  <cr>
```

The "put" command sets the value of the attribute (i.e. name) of the object (i.e. ship) to "America". The attribute

of the object must be a list. The attribute of the object must be a number of values. The phrase "first member" tells the RITA system where the value (i.e. America) belongs in the list.

```
TYPE,
 * put "Carrier" into name of ship as last member;     <cr>
 * display object ship;                               <cr>
```

The "put" command took the value (i.e. Carrier) and placed it as the last member in the list. Object ship has an attribute (i.e. name) which is a list. The list has two members. Member one is value "America" and member two is value "Carrier".

```
TYPE,
 * remove first member from name of ship;     <cr>
 * display object ship;                       <cr>
```

The "remove" command takes the value (i.e. America) of the first member of the list and deletes it from the list. The list (i.e. name of ship) has only one member (i.e. Carrier).

```
TYPE,
 * remove last member from name of ship;          <cr>
 * display object ship;                          <cr>
```

The "remove" command takes the last member (i.e. Carrier) and deletes it as a member of the list (i.e. name

of ship). Other forms of the "remove" command follow:

DO NOT TYPE,

* remove "America" from name of ship; <cr>

* remove member "America" from name of ship; <cr>

* remove every "Carrier" from name of ship; <cr>

* remove first "America" from name of ship; <cr>

TYPE,

* delete all objects and delete all rules; <cr>

IF YOU WISH TO TERMINATE THE RITA SESSION AT THIS TIME TURN TO THE SECTION OF THE TUTORIAL TITLED LEAVING A RITA SESSION.

TYPE,

* load monitor.tew; <cr>

WAIT FOR PRINTING AT THE TERMINAL TO STOP.

TYPE,

* display all objects and display all goals; <cr>

The next command to be discussed is the "deduce" command with a special option. We've loaded our external UNIX file "monitor.tew".

TYPE,

* create a ship whose status is "underway"; <cr>

* display object ship; <cr>

WAIT FOR PRINTING AT THE TERMINAL TO STOP.

TYPE,

```
 * deduce the performance of the ship [quietly]; <cr>
 * display object ship;    <cr>
```

The "create" command enters a value (i.e. underway) in the status of the ship. The "deduce" command calls the goal-directed monitor. The optional word "quietly" enclosed in braces ([]) causes the RITA system to provide no information while performing the deduction.

TYPE,

```
 * delete all objects;    <cr>
 * delete all rules;    <cr>
 * delete all goals;    <cr>
```

IF YOU WISH TO TERMINATE THE RITA SESSION AT THIS TIME TURN TO THE SECTION OF THE TUTORIAL TITLED LEAVING A RITA SESSION.

XXVIII. EXTERNAL UNIX PORTS

One of the objectives of the RITA system is to provide a facility for communicating from one computer to another. The RITA system provides for this capability by opening ports to the UNIX system. A port is a communication path. The port is given a name. Then information in our RITA program can be sent to and received from external computer systems by referring to the port by name. The RITA commands "send" and "receive" let the RITA system know that the information will be transferred via the specified port. The following example will show how a user can communicate information between two computer systems.

TYPE,

```
* object local-system                    <cr>
  status is "ready";                     <cr>

* object external-system;                <cr>

* rule communicate                       <cr>
  If the status of the local-system is "ready"     <cr>
  Then send "telnet isia" to port1                 <cr>
    & receive next (anything followed              <cr>
    by "EXEC" followed by                          <cr>
    anything) from port1 as the                    <cr>
    response of external-system                    <cr>
    & display object external-system;              <cr>
```

```
* rule terminate                    <cr>
  If the response of the external-system is     <cr>
     known     <cr>
  Then send concat ("logout","fm") to port1          <cr>
  & RETURN SUCCESS;                  <cr>

* run;          <cr>
```

An object (i.e. local-system) was entered. Object "local-system" was given an attribute (i.e. status). The attribute "status" was given a value (i.e. ready). A second object (i.e. external-system) was entered. Object "external-system" was given an attribute (i.e. response). The attribute "response" was given a value (i.e. EXEC). Rule "communicate" was entered. The If part of rule "communicate" checked to see if the value of the attribute (i.e. status) of the object (i.e. local-system) was "ready". In this case rule "communicate" was true. The actions of the rule were then performed.

The first action was the command "send". The RITA command "send" causes the information following the command (i.e. telnet isia) to be placed into a communication port (i.e. port1). A port is a path whereby communication can take place with an external computer. The information sent was "telnet isia". That command was interpreted by a UNIX shell program. It caused a connection to be opened to the host computer whose name was "isia".

Again, the "send" command causes information in the RITA

system to be transferred to the UNIX system. The place where the information is put is a port - a communication path. We named the port "port1". The symbol "&" means the same as the word "and". The next action taken was "receive" a pattern. The RITA command "receive" allows the user to transfer information from the UNIX system port. The pattern specified was anything followed by "EXEC" followed by anything. A string of charactrs that contained the four characters E X E C and any others would be matched. "Port1" is the name of the UNIX port where the information from the external computer will be put. The command "as the response of external-system" tells the RITA system to take the pattern that is received from "isia" via port1 and set the value of the attribute (i.e. response) of the object (i.e. external-system) to that string of characters.

Notice the value of the response of the external-system originally contained only four characters. Now after the "display object external-system" command the value of the response of the external-system contains numerous characters, not just four. Rule "terminate" checks to see that the value of the response of the external-system is known. That's an indication that we were able to communicate with another computer. If the value of the response of the external-system had not been known, we would have known that our attempt at communication with the computer at "isia" had been fruitless. The Then part of rule "terminate" sends the characters "logout" and the conversion

symbols for the carriage-return <cr> to port1 to be sent to "isia". The characters "logout" represent the command to terminate the connection to the computer at "isia". Notice the "RETURN SUCCESS" to make sure our program stopped.

TYPE,

```
 * delete all rules;       <cr>
 * delete all objects;     <cr>
```

XXIX. LEAVING A RITA SESSION

There are three ways to leave the RITA system under normal conditions. To save our RITA program prior to exiting, do the following:

TYPE,

```
 * display all objects to file well-done.tew;      <cr>
 * display all rules to file well-done.tew;        <cr>
 * display all goals to file well-done.tew;        <cr>
 * exit;                                          <cr>
```

The "display" commands put all the objects, rules and goals in an external file "well-done.tew". The RITA command "exit" causes the user to leave the RITA system and to enter the UNIX operating system. (The end-of-file terminator, ctrl-d, performs the same function as the RITA command "exit".) The RITA system responds by printing the word "exiting" then a UNIX prompt-sign (%). If we did not want to save our RITA program, the RITA command "exit" would have been used. However, the RITA command "display" would not have been used.

DO NOT TYPE,

```
 * exit;          <cr>
```

The third way of leaving a RITA session, under normal conditions, concerns itself with saving two files that the RITA system keeps. The RITA system keeps a "ufe.output"

file and a "rita.history" file. The "ufe.output" file contains a record of all the output that has been generated by the RITA session. The "rita.history" file contains a record of all the major events that took place during a RITA session. Examples of the major events kept in the "rita.history" file follow:

(1) Objects and rules loaded.

(2) Each time a rule or goal fires.

To save the "ufe.output" and "rita.history" files after a RITA session, the RITA command "exit save" would have been used.

DO NOT TYPE,

```
* exit save;     <cr>
```

The "ufe.output" file generated by the RITA system is useful if the user wants to check for error messages or would like to recall information about the output of RITA. The "rita.history" file generated by the RITA system is useful if the user desires to know about the sequence of major events in his RITA program.

Sections XXX and XXXI pertain to the UNIX operating system at The Rand Corporation. Use the appropriate procedures to log off the host computer that you are using.

XXX. LOGOUT PROCEDURES

After leaving the RITA system, we should log out of the UNIX operating system. There are several UNIX files that we have created while following this tutorial. If you have not completed all the sections of the tutorial, skip to the command "% logout". If you have completed all the sections of the tutorial do the following:

TYPE,

```
% rm navy-blue.tew    <cr>
% rm cyclic.tew    <cr>
% rm ordered.tew    <cr>
% rm monitor.tew    <cr>
% rm well-done.tew    <cr>
```

The "rm" command removes the specified file from our user space. Remember the names of your files end with your initials.

TYPE,

```
% logout     <cr>
```

The UNIX command "logout" lets the UNIX operating system know that we are terminating our use of the UNIX facilities. (Do not be concerned that the UNIX system asks for the user name again.)

XXXI. DISCONNECTING THE TIP (TERMINAL INTERFACE MESSAGE PROCESSOR)

After logging out of the UNIX operating system on the PDP-11 at Rand Corporation, our connection should be closed prior to hanging up the telephone.

TYPE,

@c <cr>

The "@" symbol tells the ARPANET that we wish to type a command. The "c" is the command to close the connection from our terminal to the computer system. The TIP (Terminal Interface Message Processor) is still connected. If we desired to open a connection to a computer on the ARPANET, we could do so. However, to disconnect the TIP do the following:

HANG UP THE TELEPHONE.

The tutorial has attempted to make a RITA newcomer as familiar with the RITA system as possible in a brief time. With the information that you have gained during the past few hours, you are ready to use the RITA Reference Manual. The following may be useful in locating the references referred to in this tutorial.

The RITA Reference Manual is available from The Rand Corporation, Santa Monica, California, 90406. The references UNIX for Beginners and A Tutorial Introduction to the UNIX Text Editor should be available at the location of

the host computer you are accessing to use the RITA system.

XXXII. LIST OF REFERENCES

Kernighan, B. W., A Tutorial Introduction to the UNIX Text Editor, internal memorandum at Bell Laboratories, Murray Hill, New Jersey, undated.

Kernighan, B. W., UNIX for Beginners, paper presented at Bell Laboratories, Murray Hill, New Jersey, 10 March 1976.

Rand Report R-1808-ARPA, RITA Reference Manual, by R. H. Anderson and others, September 1977.

Rand Report R-1809-ARPA, Rand Intelligent Terminal Agent (RITA): Design Philosophy, by R. H. Anderson and J. J. Gillogly, p. 1-2, February 1976.

APPENDIX B - SAMPLE RITA PROGRAMS

The major strengths of the RITA system have been the English-like command language, intelligent behavior, and the facilities for communicating with external computer systems. Three sample programs have been written to demonstrate the usefulness of RITA programs acting semiautonomously to communicate with other computer systems on the ARPANET.

These programs depend on each computer system being operational and capable of communicating with other computer systems when the RITA programs are executed. The sample programs do not include features for handling the cases where the external computer systems are unable to respond. For programs that are critical the user should include rules in his prorams that handle the cases where the external computer systems are unable to respond.

The first RITA program opens a connection from a PDP-11 computer at The Rand Corporation to a computer at the University of Southern California. The RITA program logs the user into the computer at USC and logs him out. The connection to the computer at USC is closed and the user is returned to the RITA system at The Rand Corporation. Once the program is started by the user, the RITA system handles all the communication protocols and requires no human intervention.

The second RITA program opens a connection from a PDP-11 computer at The Rand Corporation to a different computer at

USC. The user starts the RITA program running and the RITA system handles all the details of opening a connection to the computer at USC, logging in, listing the files in a directory, renaming a particular file, logging out, closing the connection to the computer at USC, and returning to the computer at The Rand Corporation.

The third RITA program combines programs one and two and performs the same functions as programs one and two did individually. However, program three communicates with the two computers at USC simultaneously.

The fact that a computer at The Rand Corporation and two computers at the University of Southern California are used is not significant. RITA programs could be written to perfom the same functions with other computers on the ARPANET.

To run the sample RITA programs, type them into the RITA system. Type the RITA commands "set trace 4;" and "run;" to start the programs running. The trace command provides the maximum information to the user during RITA program execution.

The sample RITA programs and their explanations follow:

EXAMPLE 1

```
[OBJECTS:]

OBJECT remote-system:
        login-prompt is "EXEC",
        prompt is "JOB";
```

```
OBJECT demo:
        state is "start";

OBJECT port2:;

[RULES:]

RULE host-c:
    IF:  the state of the demo is "check-telnet"
         & the response of the port2 is known
  THEN:  receive next { the login-prompt of the remote-
system} for
         60 seconds from port2 as the response of the port2
         & set the state of the demo to "check remote-host"
         & display object port2
         & display object demo;

RULE command-c:
    IF:  the state of the demo is "check remote-host"
         & the response of the port2 is known
  THEN:  send concat( "log experimental", "↑[", "ex",  "↑[",
"↑[", "↑m" )
         to port2
         &  receive  next  {  anything  followed  by  "JOB"
followed by
         anything  }  for  15  seconds  from  port2  as  the
response of the
         port2
         & display object port2
```

```
        & delay 3 seconds
        & set the state of the demo to "logged in"
        & display object demo;

RULE telnet-c:
    IF:  the state of the demo is "start"
  THEN:  send "telnet isic" to port2
        &  receive next  (  "Connections  established."
followed by
        anything )  for  15  seconds  from  port2  as  the
response of the
        port2
        & display object port2
        & set the state of the demo to "check-telnet"
        & display object demo;

RULE logout-c:
    IF:  the state of the demo is "logged in"
  THEN:  send concat ("logout", "fm") to port2
        & set the state of the demo to "logged out"
        & display object demo
        & return success;
```

In the example program above, rule "telnet-c" opens a connection from the RITA system at The Rand Corporation to a computer at USC with an ARPANET address of "isic". Rule "host-c" allows the RITA system at The Rand Corporation to insure that a connection has been made to the proper computer. Identifying information from the computer at USC

is received requesting that the user login. Rule "command-c" sends the login information to the computer at USC and receives information to confirm a successful login. Rule "logout-c" logs the user out of the computer at USC, closes the connection to the USC computer on the ARPANET, and returns the user to the RITA system at The Rand Corporation.

EXAMPLE 2

```
[OBJECTS:]

OBJECT remote-system-a:
       login-prompt is "EXEC",
       prompt is "JOB";

OBJECT demo-a:
       state is "start";

OBJECT port1:;

[RULES:]

RULE host:
    IF:  the state of the demo-a is "check-telnet"
         & the response of the port1 is known
  THEN:  receive next { anything followed by "SYSTEM-A"
followed by
         anything } for 15 seconds from port1 as the
response of the
         port1
         & set the state of the demo-a to "check remote-
```

```
host"
        & display object port1
        & display object demo-a;

RULE command:
    IF: the state of the demo-a is "check remote-host"
        & the response of the port1 is known
  THEN: send concat ( "log nps-accat", "↑[", "c77", "↑[",
"↑[", "↑m" )
        to port1
        & receive next { anything followed by "JOB"
followed by
        anything } for 15 seconds from port1 as the
response of the
        port1
        & display object port1
        & delay 3 seconds
        & set the state of the demo-a to "logged in"
        & display object demo-a;

RULE telnet:
    IF: the state of the demo-a is "start"
  THEN: send "telnet isia" to port1
        & receive next { "Connections established."
followed by
        anything } for 15 seconds from port1 as the
response of the
        port1
```

```
        & display object port1
        & set the state of the demo-a to "check-telnet"
        & display object demo-a;

RULE logout:
    IF:  the state of the demo-a is "finished"
  THEN:  send concat ("logout", "↑m") to port1
        & set the state of the demo-a to "logged out"
        & display object demo-a
        & return sucess;

RULE directory:
    IF:  the state of the demo-a is "logged in"
  THEN:  send concat ("dir", "↑[", "↑m") to port1
        &  receive  next  {  anything  followed  by
"queryIII.demo;3"
        followed by anything } for 60 seconds from port1 as
the
        response of port1
        & display object port1
        & set the state of the demo-a to "listing"
        & display object demo-a;

RULE rename:
    IF:  the state of the demo-a is "listing"
        & the response of the port1 is known
  THEN:  send concat ("ren", "↑[", "queryIII.demo;3", "↑[",
        "queryIII.gary;3", "↑[", "↑m") to port1
        & receive next { anything followed by ";3" followed
```

```
by
        anything } for 60 seconds from port1 as the
response of
        port1
        & send concat ("dir", "↑[", "↑m") to port1
        & receive next { anything followed by ";3" followed
by
        anything } for 60 seconds from port1 as the
response of
        port1
        & display object port1
        & send concat ("ren", "↑[", "queryIII.gary;3",
"↑[",
        "queryIII.demo;3", "↑[", "↑m") to port1
        & set the state of the demo-a to "finished"
        & display object demo-a;
```

In the example program above, the rule "telnet" opens a connection from the RITA system at The Rand Corporation to a computer at USC with an ARPANET address of "isia". Rule "host" allows the RITA system at The Rand Corporation to insure that a connection has been made to the proper computer. Identifying information from the computer at USC is received requesting that the user login. Rule "command" sends the proper login commands and receives information from the computer at USC that the login was successful. Rule "directory" causes the names of the user's files at USC to be listed. Rule "rename" chooses a particular file at

USC and changes its name. The names of the user's files are listed again to show that one of the names of the user's files has been changed. Rule "rename" also changes the name of the file back to its original name for convenience. Rule "logout" logs the user out of the computer at USC, closes the connection on the ARPANET, and returns the user to the RITA system at The Rand Corporation.

EXAMPLE3

```
[OBJECTS:]

OBJECT remote-system:
       login-prompt is "EXEC",
       prompt is "JOB";

OBJECT demo:
       state is "start";

OBJECT port2:;

OBJECT remote-system-a:
       login-prompt is "EXEC",
       prompt is "JOB";

OBJECT demo-a:
       state is "start";

OBJECT port1:;

[RULES:]
```

```
RULE host-c:
    IF:  the state of the demo is "check-telnet"
         & the response of the port2 is known
  THEN:  receive next { the login-prompt of the remote-
system} for
         60 seconds from port2 as the response of the port2
         & set the state of the demo to "check remote-host"
         & display object port2
         & display object demo;

RULE command-c:
    IF:  the state of the demo is "check remote-host"
         & the response of the port2 is known
  THEN:  send concat( "log experimental", "↑[", "ex",  "↑[",
"↑[", "↑m" )
         to port2
         &  receive  next  {  anything  followed  by  "JOB"
followed by
         anything  }  for  15  seconds  from  port2  as  the
response of the
         port2
         & display object port2
         & delay 3 seconds
         & set the state of the demo to "logged in"
         & display object demo;

RULE telnet-c:
    IF:  the state of the demo is "start"
```

```
  THEN:  send "telnet isic" to port2
         & receive next { "Connections established."
followed by
         anything } for 15 seconds from port2 as the
response of the
         port2
         & display object port2
         & set the state of the demo to "check-telnet"
         & display object demo;

RULE host:
    IF:  the state of the demo-a is "check-telnet"
         & the response of the port1 is known
  THEN:  receive next { anything followed by "SYSTEM-A"
followed by
         anything } for 15 seconds from port1 as the
response of the
         port1
         & set the state of the demo-a to "check remote-
host"
         & display object port1
         & display object demo-a;

RULE command:
    IF:  the state of the demo-a is "check remote-host"
         & the response of the port1 is known
  THEN:  send concat ( "log nps-accat", "↑[", "c77", "↑[",
"↑[", "↑m" )
```

```
        to port1
        & receive next { anything followed by "JOB"
followed by
        anything } for 15 seconds from port1 as the
response of the
        port1
        & display object port1
        & delay 3 seconds
        & set the state of the demo-a to "logged in"
        & display object demo-a;

RULE telnet:
    IF: the state of the demo-a is "start"
  THEN: send "telnet isia" to port1
        & receive next { "Connections established."
followed by
        anything } for 15 seconds from port1 as the
response of the
        port1
        & display object port1
        & set the state of the demo-a to "check-telnet"
        & display object demo-a;

RULE directory:
    IF: the state of the demo-a is "logged in"
  THEN: send concat ("dir", "↑[", "↑m") to port1
        & receive next { anything followed by
"queryIII.demo;3"
```

```
        followed by anything } for 60 seconds from port1 as
the
        response of port1
        & display object port1
        & set the state of the demo-a to "listing"
        & display object demo-a;

RULE rename:
    IF: the state of the demo-a is "listing"
        & the response of the port1 is known
  THEN: send concat ("ren", "↑[", "queryIII.demo;3", "↑[",
        "queryIII.gary;3", "↑[", "↑m") to port1
        & receive next { anything followed by ";3" followed
by
        anything } for 60 seconds from port1 as the
response of
        port1
        & send concat ("dir", "↑[", "↑m") to port1
        & receive next { anything followed by ";3" followed
by
        anything } for 60 seconds from port1 as the resonse
of
        port1
        & display object port1
        & send concat ("ren", "↑[", "queryIII.gary;3",
"↑[",
        "queryIII.demo;3", "↑[", "↑m") to port1
        & set the state of the demo-a to "finished"
```

```
        & display object demo-a;

RULE logout:
    IF:  the state of the demo-a is "finished"
         & the state of the demo is "logged in"
  THEN:  send concat ("logout", "↑m") to port1
         & send concat ("logout", "↑m") to port2
         & set the state of the demo-a to "logged out"
         & set the state of the demo to "logged out"
         & display object demo-a
         & display object demo
         & return success;
```

The example program above communicates with two computers simultaneously. The rule "logout" was modified to insure that the RITA program terminated only after the user was ready to logout of both external computer systems.

The method by which the RITA programs communicate with other computer systems depends on the user knowing the expected responses from the external computer systems and matching some of the characters in each response. The RITA system uses the "send" and "receive" commands to transfer information in and out of the RITA system.

The user should be familiar enough with the RITA system, after working through Appendix A, to understand most of the specifics contained in the sample RITA programs. However, several items were not discussed in Appendix A. The "↑m" represents the carriage- return and the "↑[" represents the

ctrl-esc. Telnet is an ARPANET communication command to open a connection to another computer.

After running the sample programs and observing their output, the user should be able to understand how the RITA system communicates with other computer systems. The objects are printed at the terminal or crt frequently to reinforce the idea that the RITA system is making intelligent decisions about what should be done next based upon the contents of its dynamic database.

BIBLIOGRAPHY

Bolt, Beranek, and Newman, Inc. Report ESD-TR-75-58, Mixed-Initiative Tutorial System to Aid Users of the On-Line System (NLS), by M. C. Grignetti and others, p. 27, November 1974.

Defense Documentation Center Report ESD-TR-75-56, Evaluation of TICS: A Multics Subsystem for Development and Use of CAI Courseware, by S. Goheen and D. Jordon, April 1975.

Kernighan, B. W., A Tutorial Introduction to the UNIX Text Editor, internal memorandum at Bell Laboratories, Murray Hill, New Jersey, uncated.

Kernighan, B. W., UNIX for Beginners, paper presented at Bell Laboratories, Murray Hill, New Jersey, 10 March 1976.

Newell, A., "Production Systems: Models of Control Structures," Visual Information Processing, p. 463-526, 1973.

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

Rand Report R-1808-ARPA, RITA Reference Manual, by R. H. Anderson and others, September 1977.

Rand Report R-1809-ARPA, Rand Intelligent Terminal Agent (RITA): Design Philosophy, by R. H. Anderson and J. J. Gillogly, p. 1-2, February 1976.

Shortliffe, E. H., and others, "Computer-based Consultations in Clinical Therapeutics: Exploration and Rule Acquisition Capabilities of the MYCIN System," Computers and Biomedical Research, v. 8, p. 303-320, 1975.

Waterman, D. A., "Adaptive Production Systems," 4th IJCAI Conference Proceedings, p. 296-303, September 1975.

Waterman, D. A., An Introduction to Production Systems, paper presented at The Rand Corporation, Santa Monica, California, November 1976.

Waterman, D. A., "Generalization Learning Techniques for Automating the Learning of Heuristics," Artificial Intelligence, v. 1, p. 121-170, 1970.

Weizenbaum, J., Computer Power and Human Reason, p. 183, 235, 243, W. H. Freeman and Company, 1976.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
4.	Department Chairman, Code 55 Department of Operations Research Naval Postgraduate School Monterey, California 93940	1
5.	Professor G. K. Poock, Code 55Pk Department of Operations Research Naval Postgraduate School Monterey, California 93940	25
6.	LTCOL R. J. Roland, USAF, Code 52Rl Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
7.	LT Thomas E. Warren, USN 4274 Lynda Street Jackson, Mississippi 39209	1
8.	CAPT R. Ammann, USN, Code OP942C Department of the Navy Office of the Chief of Naval Operations OP-094 Washington, D. C. 20350	1
9.	Professor D. R. Barr, Code 55Bn Department of Operations Research Naval Postgraduate School Monterey, California 93940	1
10.	Mr. C. Blais Naval Ocean Systems Center Code 8321 San Diego, California 92152	1

11. Mr. R. Brandenburg — 1
Naval Ocean Systems Center
Code 8321
San Diego, California 92152

12. Dr. M. Denicoff, Code 437 — 1
Office of Naval Research
800 N. Quincy Street
Arlington, Virginia 22217

13. LCDR A. J. Dietzler — 1
Advanced Research Projects Agency
Information Processing Techniques Office
1400 Wilson Boulevard
Arlington, Virginia 22209

14. Professor D. P. Gaver, Code 55Gv — 1
Department of Operations Research
Naval Postgraduate School
Monterey, California 93940

15. Dr. N. Greenfeld — 1
Bolt Beranek & Newman, Inc.
50 Moulton Street
Cambridge, Massachusetts 02138

16. CDR F. H. Hollister, USN — 1
Advanced Research Projects Agency
Information Processing Techniques Office
1400 Wilson Boulevard
Arlington, Virginia 22209

17. Dr. G. Martins — 1
The Rand Corporation
1700 Main Street
Santa Monica, California 90406

18. Dr. H. Miller — 1
Naval Ocean Systems Center
Code 8321
San Diego, California 92152

19. Dr. J. R. Miller — 1
Science Applications, Inc.
1911 N. Fort Myer Drive, Suite 1200
Arlington, Virginia 22209

20. LCDR W. Mitchell, USN — 1
Naval Ocean Systems Center
Code 8321
San Diego, California 92152

21. Dr. H. Morgan 1
Wharton School
University of Pennsylvania
Philadelphia, Pennsylvania 19104

22. LTCOL F. R. Murphy, USAF, Code 39 1
Naval Postgraduate School
Monterey, California 93940

23. Mr. D. Norman 1
Manager of Operations
W. R. Church Computer Center
Naval Postgraduate School
Monterey, California 93940

24. Assoc Professor S. H. Parry, Code 55Py 1
Department of Operations Research
Naval Postgraduate School
Monterey, California 93940

25. Professor G. A. Rahe, Code 52Ra 1
Department of Computer Science
Naval Postgraduate School
Monterey, California 93940

26. Assoc Professor F. R. Richards, Code 55Rh 1
Department of Operations Research
Naval Postgraduate School
Monterey, California 93940

27. COL D. Russell, USA 1
Advanced Research Projects Agency
Information Processing Techniques Office
1400 Wilson Boulevard
Arlington, Virginia 22209

28. Dr. E. Sacerdoti 1
Stanford Research Institute
Artificial Intelligence Center
333 Ravenswood Avenue
Menlo Park, California 94025

29. Dr. J. Schill 1
Naval Ocean System Center
Code 8123
San Diego, California 92152

30. Professor N. F. Schneidewind, Code 54Ss 1
Department of Administrative Sciences
Naval Postgraduate School
Monterey, California 93940

31. Mr. C. C. Stout 1
Naval Electronics Systems Command
Code 330
2511 Jefferson Davis Highway
Arlington, Virginia 20360

32. Dr. M. Tolcott, Code 455 1
Office of Naval Research
800 N. Quincy Street
Arlington, Virginia 22217

33. Professor J. M. Wozencraft, Code 74 1
Naval Postgraduate School
Monterey, California 93940

34. CAPT R. D. Yingling, USAF, Code 62Ya 1
Department of Electrical Engineering
Naval Postgraduate School
Monterey, California 93940