Explanation Mechanisms for Knowledge-Based Models in the RAND Strategy Assessment System

Paul K. Davis

August 1988

Prepared for
The Director of Net Assessment,
Office of the Secretary of Defense

A research publication from
The RAND Strategy Assessment Center

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED
This Note is one of a number of papers being written for users of the RAND Strategy Assessment System, Version 3.0 (RSAS 3.0). Its style is informal and its approach a mix of specific examples ("learn by doing") and theory. Its purpose is to help users cross certain thresholds. For users going beyond standard demonstration runs the system can be intimidating until these thresholds are passed.

This work was conducted by the RAND Strategy Assessment Center and sponsored by the Director of Net Assessment in the Office of the Secretary of Defense, under the auspices of RAND's National Defense Research Institute, a Federally Funded Research and Development Center sponsored by the Office of the Secretary of Defense. Comments on the paper would be appreciated, because updates will probably be issued as more experience is gained with the system and with the particular problems of early users. Comments should be addressed to Dr. Paul K. Davis, who directs the RAND Strategy Assessment Center at The RAND Corporation in Santa Monica, CA (Telephone: (213) 451-6912; electronic mail: pdavis%rondo@rand-unix.ARPA).
- v -

SUMMARY

This Note is a users' manual for understanding the behavior of knowledge-based decision models written in the RAND-ABEL® programming language and incorporated in the RAND Strategy Assessment System (RSAS). It assumes the user has already familiarized himself with the models in question, so that the issue is understanding specific behaviors in specific RSAS runs, rather than understanding the basic assumptions and approach represented in the models as a whole. As a practical matter, then, the manual is concerned with teaching the user how to use the RSAS' explanation mechanisms to answer questions such as "Why did the Red Agent decide to adopt a particular strategy at this point in the exercise?"

Even with this narrow definition of "understanding," there are many different levels of explanation available, and which is appropriate depends on what the user already knows and accepts as valid. The principal issues are: (1) What variables determine the behavior in question (to some level of depth, since lower-level variables determine the values of higher-level variables); (2) what meaning the variables have; (3) what values the key variables are permitted (ranges) and what those values mean; (4) what values the variables had when the behavior occurred; (5) the origin of those values (e.g., inputs versus rules); (6) the underlying rules and algorithms; and (7) the rationale behind those rules and algorithms.

As shown in Fig. S.1, there are a number of on-line ways to address these issues when using the RSAS and its RAND-ABEL environment on Sun workstations. The numbers in the figure indicate the typical order in which the player of an RSAS war game might go about seeking explanations. The sources of information and explanation are:

* A variable-resolution explanation log, which can also be tailored by subject.
Sources of Information and Explanation

<table>
<thead>
<tr>
<th>Information Needed</th>
<th>CR-Tool &amp; Data Logs</th>
<th>Data Dictionary</th>
<th>Data Editor</th>
<th>Source Code (rules)</th>
<th>In-code Documentation</th>
<th>General Model Doc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Variables</td>
<td>1</td>
<td></td>
<td></td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>responsible</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Meanings</td>
<td>1</td>
<td></td>
<td></td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3. Values permitted (ranges)</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. Values that determined decision</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. Origin of values</td>
<td>1</td>
<td></td>
<td></td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>6. Underlying rules and algorithms</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>7. Rationale for models or values</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Fig. S.1--Explanation Mechanisms for Players or Analysts Observing Behavior
(numbers indicate typical order in which sources are used)

- A Cross-Referencing Tool linked to RAND-ABEL's Data Dictionary; it provides authoritative information on variable ranges and displays Data Dictionary entries, which include as comments the "definitions" of the variables in question.

- A Data Editor for viewing and changing the values of RAND-ABEL's global variables.

- RAND-ABEL source code, the principal substantive elements of which are readable by people with subject-area knowledge even if they have only modest knowledge of programming. Much of the RSAS knowledge base is expressed in decision tables or order tables that are part of the language itself, and that can be changed readily to accommodate different decision variables or different judgments. RAND-ABEL is also interpretive, permitting such changes to be made during a simulation without recompiling the entire source code.
- vii -

- In-code documentation and on-line overview documentation of the models and individual rules (admittedly incomplete). Also, on-line HELP files for the various software tools.

- Not shown in Fig. S.1, a Sourcecode-Menu Tool providing access to the relevant parts of the model and a Find-Item Tool that searches through source code for variables, functions, and other items.

Given the information-gathering power of the programming environment, the limiting factor in gaining explanations is often the user's intuition about where to look for the evaluation of variables determining the values of higher-level variables in the RSAS’s hierarchical models. Success depends on a top-down understanding of the models (since the RSAS knowledge-based programs are structured to match the conceptual models) and knowing a relatively few tricks allowing him, for example, to recognize which model variables depend on data from other models and where to find the information in those other models or in the interface between the C-language and RAND-ABEL-language portions of the RSAS. Without such tricks, which are summarized tersely in Sec. V, the user may find the models user-hostile; with them, he can develop explanations to a remarkably deep level before requiring programmer assistance. It is not unusual for an analytically oriented user to track an "explanation" through five to ten levels--ending, perhaps, with a statement explaining that the critical factor was a threshold assumption used in interpreting data from the combat-simulation models. Such cause tracing would be impossible for someone other than a professional programmer (and difficult even for such a programmer) in a system designed with a more conventional programming language and environment.

Providing such explanation mechanisms is essential not only for serving users in war games and simulations, but also for serving the analysts and modelers attempting to improve the system by expanding its knowledge base or, importantly, to use the system for policy studies. Experience with the RAND-ABEL environment on Sun workstations has been extremely positive, although successes continue to generate new and more
ambitious informational demands that would have been regarded as preposterous in the not very distant past. It seems likely that major environmental improvements will continue for some years. The most important of these will probably be the development of a better interface between the RAND-ABEL and C programs within the RSAS.

Finally, a word of warning: Because the Note includes many details and works through "worst-case" challenges, the user who fails to make it through the entire Note may have the impression that learning the requisite skills is more difficult than is the case. As shown in Sec. V, there are systematic summary procedures that work well and efficiently once the user has been through the thickets once. The author therefore recommends that users plan on an uninterrupted work session of two or three hours for going through the Note from start to finish.
ACKNOWLEDGMENTS

The author appreciates the review comments of Herbert Shukiar and David Shlapak and the assistance of Melinda Baccus.
CONTENTS

PREFACE ......................................................... iii
SUMMARY ...................................................... v
ACKNOWLEDGMENTS .......................................... ix
FIGURES ......................................................... xiii

Section

I. INTRODUCTION ............................................. 1
   Objectives ................................................. 1
   Prerequisites for Primer ................................. 1
   A Guide to the Remainder of the Note .................. 2

II. ON UNDERSTANDING MODELS AND GENERATING EXPLANATIONS OF MODELS BEHAVIOR ........................................... 3

III. ILLUSTRATIVE EFFORTS TO UNDERSTAND RSAS RAND-ABEL MODELS ................................................................. 6
   Relatively Painless Explanation: Using the Logs ...... 6
   Understanding the Meaning of Key Variables .......... 8
   Finding the Current Values of Key Variables .......... 13
   The Possible Origins of Variable Values ............... 19

IV. MORE EXAMPLES: UNDERSTANDING THE UNDERLYING RULES .............. 23
   Prefatory Comments About Philosophy .................. 23
   Illustrative Operations in Understanding Decision Rules .................................................. 24
   Comments on Variable Tracing in Other RAND-ABEL Models ............................................. 42

V. SOME SUMMARY PROCEDURES AND SUGGESTIONS ........................................... 43

REFERENCES TO THE RSAS AND ITS TOOLS ........................................... 47
   Unpublished and Forthcoming Documents ................. 47
   Published Documents ...................................... 48
**FIGURES**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>S.1</td>
<td>Explanation Mechanisms for Players or Analysts Observing Behavior</td>
<td>vi</td>
</tr>
<tr>
<td>1</td>
<td>Different Aspects of &quot;Understanding&quot; Model Behavior</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>An Illustrative Section of a Blue NCL Model’s Explanation Log for the Assessment of Presumed-opponent</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>Selecting a Submodel from a Walking Menu to RAND-ABEL Source Code</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>Rule-Set for Adjusting Opponent Model</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>Finding the Exact Variable Name</td>
<td>11</td>
</tr>
<tr>
<td>6</td>
<td>Using Cross-Referencing Tool to View Enumeration Values</td>
<td>12</td>
</tr>
<tr>
<td>7</td>
<td>Using Cross-Referencing Tool to View Declarations</td>
<td>12</td>
</tr>
<tr>
<td>8</td>
<td>Finding a Global Variable in a Tableau Definitions File</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>Use of the Data Editor to Display Variable Values</td>
<td>16</td>
</tr>
<tr>
<td>10</td>
<td>Terminology in Modeling and Programming</td>
<td>18</td>
</tr>
<tr>
<td>11</td>
<td>Establishing the Time-Zero RAND-ABEL World State (WSDS-A)</td>
<td>21</td>
</tr>
<tr>
<td>12</td>
<td>Illustrative Top-of-File Comments in Source Code</td>
<td>25</td>
</tr>
<tr>
<td>13</td>
<td>Illustrative Top-of-File Comments (Continued)</td>
<td>26</td>
</tr>
<tr>
<td>14</td>
<td>Illustrative Decision Table in Strategic Warning Model</td>
<td>27</td>
</tr>
<tr>
<td>15</td>
<td>Using the Walking Menu to Select Military-actions.A</td>
<td>30</td>
</tr>
<tr>
<td>16</td>
<td>Using the Find-Item Tool to Search for IRBM</td>
<td>31</td>
</tr>
<tr>
<td>17</td>
<td>Using the Walking Menu to Select Interface,’to-Force-C/query.A</td>
<td>32</td>
</tr>
<tr>
<td>18</td>
<td>The Test-Mobilization Function</td>
<td>37</td>
</tr>
<tr>
<td>19</td>
<td>Finding What Force Display Generates the Data</td>
<td>40</td>
</tr>
<tr>
<td>20</td>
<td>The Force Display &quot;Region&quot;</td>
<td>40</td>
</tr>
<tr>
<td>21</td>
<td>HELP for the Force Display &quot;Region&quot;</td>
<td>41</td>
</tr>
<tr>
<td>22</td>
<td>A Generalized Search Procedure for Finding Antecedent Variables</td>
<td>44</td>
</tr>
<tr>
<td>23</td>
<td>Sources of Information and Explanation</td>
<td>45</td>
</tr>
</tbody>
</table>
I. INTRODUCTION

OBJECTIVES

This Note is a primer on how to understand the behavior of knowledge-based models written in the RAND-ABEL programming language as part of the RAND Strategy Assessment System (RSAS), with which the reader is assumed to be somewhat acquainted. Although most of the examples are drawn from the RSAS' National Command Level (NCL) models, much of the discussion is general, because the techniques used to find out why a RAND-ABEL model reached a particular decision are largely independent of model and depend instead on the integrated environment of computer tools used to build all RSAS models written in RAND-ABEL.¹

PREREQUISITES FOR PRIMER

To achieve reasonable brevity, the Note assumes some reader familiarity with the RSAS, with the models he is trying to understand, with the basic issues of computer programming (even though he need not be himself a programmer), and with RAND-ABEL. As a test to see whether other background material should first be studied before going through this primer, the reader should be reasonably familiar with:

- **RSAS architecture**: e.g., the names and functions of the various agents; the hierarchical structuring of most agents; and the World Situation Data Set (WSDS) (both WSDS-C and WSDS-A, representing the parts maintained in the C and RAND-ABEL sides of the system).

- **RSAS tools**: the RAND Editor ("e"), the Sourcecode-Menu-Tool providing walking menus to RAND-ABEL rules, the Data Editor, the Cross-Referencing Tool, the Interpreter, and the Find-Item Tool. Also, simple UNIX commands such as "cd" and instructions such as "bring up a shell window."

- **RAND-ABEL**: e.g., the Data Dictionary, data types, decision tables, global vs. local variables, and functions.

¹The RAND-ABEL environment for Sun computer workstations has been designed and developed under the leadership of H. Edward Hall.
*The Specific Model:* e.g., the basic concepts and architecture of the NCL, Green, or military-command-level models (often referred to as analytic war plans).

A reader needing more background should refer to the materials cited in the bibliography, as well as to the RSAS's on-line documentation, which will be more effective if perused initially in hard-copy to provide a top-down sense for the models or tools being used. Most importantly, users should go through the at-work-station exercises of the Beginner's Guide (Gillogly, forthcoming).

A GUIDE TO THE REMAINDER OF THE NOTE

Assuming appropriate background, then, we are ready to proceed with this Note's discussion of how to "understand" RAND-ABEL RSAS models. Section II discusses what it means to "understand" such models and to generate explanations from the RSAS. In Secs. III and IV we shall walk through a series of examples of how one actually proceeds at the workstation when seeking to understand model behavior. The examples include some difficult and near-worst cases rather than being restricted to the easy cases one stresses in demonstrations. Most explanation efforts, indeed, are simple. Section V then pulls together some of the lessons learned, generalizes, and provides some systematic procedures. As he gains expertise with the RSAS, a user will learn to use a variety of shortcuts making use of specialized knowledge (e.g., will "know" where to look for information), but the systematic procedures will continue to be useful as a fallback and quick-reference guide.
II. ON UNDERSTANDING MODELS AND GENERATING EXPLANATIONS
OF MODEL BEHAVIOR

Suppose that you have run an RSAS model written in RAND-ABEL and
are now trying to find out why it reached a particular conclusion or
took some particular action. You might ask questions such as:

- Why did France (represented in Green Agent) decide to use
  nuclear weapons?
- Why did Ivan0 (a particular Red NCL model) conclude that there
  was strategic warning in Europe of Blue's imminent use of
  tactical nuclear weapons?
- Why did the Warsaw Pact commander for the attack on NATO's
  Center Region conduct nuclear strikes on France (as reflected
  by orders issued from a particular analytic war plan, HCFW1.A)?

It is important to recognize here that it is not clear what "Why?"
means. You might want to know what variables the conclusions depend on
(i.e., what were the issues?). Or, you might already have a sense for
the variables, in which case you might want to know the values of one or
two of the key variables—perhaps you agree there is a threshold effect,
but you don't know whether the variables exceeded the threshold. Or,
you might want to know the underlying rules1 that determined those
values. If the RSAS agents represent "knowledge-based models," what
knowledge do they possess and is it valid, or at least reasonable?
Going even further, you may want to understand better the rationale for
the underlying rules. This last item is particularly difficult, because
rationale is often implicit rather than explicit—i.e., it represents
"deep knowledge" reflected by, rather than revealed in, the rules.

1In this Note the term "rules" covers all of the relationships
contained by the RAND-ABEL models. Many of them are qualitative
relationships such as "If a certain situation occurs, then the
appropriate action is..." Others involve what would elsewhere be
described as algorithms: some rules are simple equations, while others
are essentially bookkeeping procedures such as "If <event> happens,
increase the number of strategic indicators by 1."
It is also unclear how deeply you want to go into explanations of why something happened. You will find that each explanation involves new variables, which you may then want to have explained as well—resulting in an iterative process that can take you far into the details. Sometimes this is called "getting into the weeds" (i.e., losing the larger perspective the RSAS attempts to preserve), but it is often unavoidable because lower-level details matter greatly and are not something to be taken for granted. Suppose you are told that France takes a certain action, in part because she sees a "Mortal threat." You may be satisfied. Or, you may want to know why France thinks the threat is mortal, whether that assessment is sound logically, and whether it is based on valid information rather than misperceptions. At a more elementary level, you may want to know what France (or the rule-writer attempting to describe plausible French behavior) meant by a "mortal threat." The rules might say when the threat is judged mortal, but to know whether they are sound you need to have a sense for what "mortal" is supposed to mean.

Understanding, then, is a complex subject—as is providing explanations (the flip side of the coin). Figure 1 summarizes the principal issues we shall discuss in this Note. Each successive issue reflects the desire to dig one step more deeply into the reasons for the model's behavior and the original rule-writer's rationale.

2The interested reader may wish to pursue the issue of model explanations by reviewing some of the relevant literature from research in artificial intelligence and the subfield of expert systems. See, for example, Waterman, Paul, Florman, and Kipps (1986) for a discussion of explanation experiments with the ROSE® language which, unlike RAND-ABEL, includes an inference engine that has the side benefit of making cause-tracing queries relatively straightforward. See also Hayes-Roth, Waterman, and Lenat (1983), which is virtually a textbook on expert systems. Paul (1987) is a recent dissertation dealing with user-tailored explanation facilities.
o Which variables determine the behavior (to some level of depth in a hierarchy of variables)?

o What do the variables mean?

o What values could the variables have (i.e., what are their ranges)?

o What were the then-current values of the key variables when the behavior in question occurred?

o What was the origin of those values? That is, where were they set (inputs, rules, algorithms)?

o What were the underlying rules (including algorithms) used to evaluate the variables?

o What was the rationale for those rules?

Fig. 1--Different Aspects of "Understanding" Model Behavior
III. ILLUSTRATIVE EFFORTS TO UNDERSTAND RSAS RAND-ABEL MODELS

This section and the next proceed as though we were at a workstation tracking down answers to questions such as those raised in the preceding section. It describes by example the types of operations we might go through. Later, in Sec. IV, we will consider more systematic procedures, but the purpose here is to build a base of experience.

RELATIVELY PAINLESS EXPLANATION: USING THE LOGS

Basic Features of RAND-ABEL Logs in the RSAS

Most of the time you will find yourself understanding behavior of RAND-ABEL models merely on the basis of reading the agent logs. You may be doing this as an RSAS simulation is proceeding, or subsequently. The biggest problem with logs is usually the quantity of information, but if you have a sense for the underlying model from its documentation, you will then be able to skim the logs quickly to find the information most relevant to your questions. Further, because the logs are generated with a postprocessing tool from a more general file, you can tailor your logs to provide greater or lesser explanation, and to focus on particular subjects of interest (this may require tagging those subjects in the RAND-ABEL source code before running the RSAS).\footnote{The RSAS logging mechanism was designed and implemented by Robert Weissler, drawing on early suggestions by Steven C. Bankes. For on-line documentation select HELP from the menu under Logging on the main background menu. Use the appropriate Data Editor tableau (currently, "Set Controls for Logging Output" in the tableau set run-rsas.t) to determine the information logged as the game proceeds. (Note: the RSAS default is to record only decisions, so you \textit{must} specify more extensive logging if you are seeking explanations). Use the Logging Tool to specify what part of the log you wish to see--either as the game proceeds or afterward.} Figure 2 shows an extract from a Blue national command level decision in a particular RSAS simulation.\footnote{It would be relatively straightforward to dress up these explanations to include flowery and superficially impressive language reminiscent of explanations given verbally by one human to another. We prefer the clarity of the terse no-nonsense approach.} The key independent variables of the decision are...
apparent as the "givens." In the example, Blue is changing his perception of his opponent--i.e., choosing a different Blue's Red. Since this is part of Blue's log, the "givens" are Blue's perceptions of reality and current psychological attributes (e.g., Blue's prevailing mind-set).

**Caveats About Relying on Logs**

Valuable as they are, logs are not always adequate or authoritative. The original modeler (or programmer) used judgments in deciding what should be logged, and you may want other information. Also, it is not uncommon for information to be unintendedly not logged. This is a form of "bug," but one that is often not detected until someone is paying close attention to the particular issues. It is also sometimes misleading to read the logs without seeing the underlying rules, because words don't always mean what they seem to mean. As a practical matter, you may want to review and modify the "log" statements.

---

**Adjustment of Assumptions and Behavior**

<table>
<thead>
<tr>
<th>Given:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Global situation:</td>
<td>Eur-gen-tac-nuc</td>
</tr>
<tr>
<td>Warning level:</td>
<td>IC-strat</td>
</tr>
<tr>
<td>Credibility of warning:</td>
<td>High</td>
</tr>
<tr>
<td>Prior opponent assumption:</td>
<td>Red3</td>
</tr>
<tr>
<td>Prevailing mind-set:</td>
<td>Mixed-controlled</td>
</tr>
<tr>
<td>Earlier mind-set:</td>
<td>Mixed-controlled</td>
</tr>
<tr>
<td>First CV attacker:</td>
<td>Unspecified</td>
</tr>
<tr>
<td>Red nuclear use:</td>
<td>None</td>
</tr>
<tr>
<td>Red demo nuclear use:</td>
<td>No</td>
</tr>
<tr>
<td>Red nuclear use avoidance:</td>
<td>High</td>
</tr>
</tbody>
</table>

So:

| Presumed opponent:              | Red4           |

---

Fig. 2--An Illustrative Section of a Blue NCL Model's Explanation Log for the Assessment of Presumed-opponent (CV means "countervalue")
in the relevant models before running the RSAS so that you will get the type of log needed for your purposes. Doing so does not require any programming expertise (you don't even have to recompile the program, since RAND-ABEL models can be changed "interpretively" as discussed in Gillogly, forthcoming). 3

UNDERSTANDING THE MEANING OF KEY VARIABLES
Definitions of RAND-ABEL Variables

After reading an explanation such as that in Fig. 2 it would be natural to seek more information on the meaning of some key variables (e.g., the variable "mind-set"). The best way to find meanings is usually as follows:

- Use the walking menu to the RAND-ABEL rules and select the file of rules that generated the explanation (Figs. 3 and 4). 4

- Once in the right file of rules, determine the exact spelling of the variable. For the example, the log says "mind-set," but the computer knows the variable as "Mind-set" as we can determine by using the editor "e" to search in the rules for the string of words used in the log, "Prevailing mind-set," and then looking to see what variable is used in the log instruction (see Fig. 5).

3To change RAND-ABEL rules interpretively, one copies the function containing the rules of interest into a file within the INT directory and given the suffix ".A". One then modifies the rules there. The next time the RSAS runs it will use the modified rules. Because interpreted code runs 50-100 times more slowly than compiled code, you should avoid interpreting large functions, especially functions with imbedded loops (e.g., loops over all countries worldwide). Under most circumstances, however, the interpreted material will be such a small fraction of the total RAND-ABEL code that you will notice little if any performance degradation.

4Figure 3 shows this for the example of Fig. 2, which determined Presumed-opponent. In using the walking menu the author first chose RAND-ABEL rules from the background menu, then moved rightward, choosing NCL and Blue for obvious reasons, choosing SamN because in the particular RSAS exercise SamN was the Blue NCL model being used, the "Adjust" rules, which relate to Blue's adjusting assumptions about the world, and finally, "Opponent-model.A" because it sounds like it deals with setting "Presumed-opponent." The Sourcecode-Menu Tool automatically opens up a window and displays the file of rules selected (Fig. 4).
Fig. 3--Selecting a Submodel from a Walking Menu to RAND-ABEL Source Code (choices from each menu are highlighted in "reverse video")

Fig. 4--Rule-Set for Adjusting Opponent Model
• Use the Cross-Referencing Tool (CR-Tool) to view the "enumeration values" (i.e., the range of values permitted for the variable). The range shown (Fig. 6) is guaranteed accurate and may be enough to give you a good sense for the variable's meaning. Mind-set, for example, has the values Fatalistic-reflexive, Fatalistic-controlled, Mixed-reflexive, Mixed-controlled, Optimistic-reflexive, and Optimistic-controlled. Do you really need much more "explanation" of what the rule-writer had in mind? [If so, that is unfortunate, because the rule-writer in this case considered the meanings sufficiently self-evident that he provided no further details anywhere. Were a study being conducted that focused on psychological factors in decisionmaking, the rules and the definition of variables would have to be enhanced.]

• If you seek more discussion, look in the Data Dictionary. Each variable's meaning (definition) is supposed to be explained in comments where the variable is declared to be an integer, enumerated variable, or whatever. First, you can try choosing "View Declaration" in the CR-Tool, which will bring up the relevant portion of Data Dictionary text (Fig. 7).  

• If there are no commented definitions (there were none for Mind-set when this Note was first written), look next in overall model documentation, where key variables such as inputs and outputs should be explained (although internal variables used by the model to organize its logic may not be, especially if their meaning was considered self evident).

• If all else fails, infer meanings from the rules themselves.

Caveats About Definitions

Again, a caveat. The prose descriptions of what a variable means are only comments. It is possible that the rules or algorithms used to evaluate a variable will be inconsistent with the description—perhaps

5The CR-Tool was implemented by Sharyn Blixt following a design by H. Edward Hall. In a future RSAS with fewer memory limitations the CR-Tool will have even greater functionality—e.g., for bringing up definitions of functions without a "search."

6Alternatively, you could select the "Where Declared" option in the CR-Tool, which will produce a statement such as "Declaration for Mind-set on line 814 of file ../../Src/NCL/Blue/Dictionary/SamD." Then open a shell tool, pull up that file using the editor "e," and go to line 814.
Fig. 5--Finding the Exact Variable Name
because of initial sloppiness by the modeler or programmer, or perhaps because of some undocumented later improvement. Also, it is often difficult to express in prose the full dimensions of what a variable means. In some cases there is really no better way to define the variable than to show the rules or algorithms evaluating it; in other cases, the explanations help a great deal. You should also remember to be open-minded, because words mean only what we say they mean and often have different meanings in different fields of study. RAND is continually trying to choose names for variables that will convey intuitively the intended meaning to most users, but it is an uphill battle that can never be completely won.

---

7To better appreciate the difficulties, you may wish to look at the definitions of Type-global-situation as described in the file Src/NCL/Blue/Dictionary/Sam.D (selectable from the walking menu for RAND-ABEL rules).
FINDING THE CURRENT VALUES OF KEY VARIABLES

Sometimes, to understand a model's decisions or actions you need the values of known key variables. For example, level of conflict or the presence of credible tactical warning can dominate decisions of the NCL model. If the decision was made earlier in the RSAS simulation, the only way to find out what the values were is to look at the logs or rerun the simulation. However, if the decision has just been made, then there are several ways to find their current values. These involve the Data Editor (for global RAND-ABEL variables), the logs, or the CMENT interface to the Force-C models. We shall discuss each in turn, and mention also certain "variables" for which you cannot readily find current values.

Using the Data Editor for Global Variables

If the variable of interest is a "global," the chances are that its value can be found in one or more Data Editor tableaus. Ordinarily, you will work routinely with only one or two tableau sets, in which case it should be fairly clear which tableaus to look in. If not, and assuming you know the precise spelling of the variable, you can search for it within the tableau-definitions file for the tableau set you would expect to find it in. To illustrate, let us look for the variable PACT-mobilization, known to be of interest to the NCL, by: (1) bringing up a shell tool window in the Run directory, (2) moving to the directory T (for "Tableaus") immediately under Run, and then to NCL under that, (3) editing the file "examine-blue-assessment.T," which defines the nature of the tableau set by the same name in the Data Editor, and (4) searching within that file for "PACT-mobilization."

---

8Global variables are variables declared in the Data Dictionary. They can be referenced (e.g., used in an expression such as "If variable-name is variable-value Then Let some-other-variable be Good.") anywhere in a given agent's program. Local variables are declared and used only within a particular function. They are not part of the simulation's "state" and have no existence after the function has been evaluated. They use only local memory in the computer.

9This description assumes familiarity with simple UNIX commands and workstation windows. See the Beginner's manual (Gillogly, forthcoming).
Doing all this, we would find the variable in the text defining a tableau called Blue NCL Theater Strategic Warning (Fig. 8).  

The variable appears in the item "Field a: PACT-mobilization." Looking upward in the file, we see the symbol "~a" in a table showing mobilization levels by theater. The table does not contain the label "PACT mobilization" per se. Looking upward somewhat farther in the file, we see the name of the particular tableau is "Blue NCL Theater Strategic Warning." We can now use the Data Editor to view that tableau, along with the current value of the variable in question (see Fig. 9, with the current value of PACT-mobilization circled).

It is possible you will want to see the value of a global variable that does not happen to exist in any Data Editor tableau. If so, you can add the variable to any tableau that has room for it. For example, if a tableau had five variables in it, you could add an item "Field 6: <Arbitrary-variable>" at the bottom of the "field" listings, and then find a spot in the display to insert "Arbitrary-variable ~6." After exiting the tableau definitions file, exiting the Data Editor and then recalling it, you would see the variable displayed.

Using the Log

We discussed the log earlier, but it is important to mention it again here as a convenient way to display values of key variables. The log mechanism is also the only way to record the values of local variables. If the variable of interest is not currently recorded, you can add an appropriate log statement in the middle of a simulation by using the RAND-ABEL interpreter--i.e., you needn't stop and recompile the RSAS, or begin the simulation from the beginning.

---

16All examples are valid as of January 5, 1988. Tableaus and tableau-definition files change, but the same principles will continue to apply.
Fig. 8 -- Finding a Global Variable in a Tableau Definitions File
Functions and Pointers, Which Have No Values

Some entities of an RAND-ABEL program are neither global nor local variables, notably functions and pointers. While a modeler may have a view such as that in the top of Fig. 10, one in which the inputs to a submodel are "variables," "parameters," and "data" (from a data base), and in which he thinks of all of the inputs having clear-cut values, to the computer scientist things are more complex. Some of what he calls "data" is generated only when needed and has no existence otherwise.\textsuperscript{11} That is, some of his data come from calls to functions or from the

\textsuperscript{11}To illustrate why not everything of interest is turned into variables, consider the square roots of arbitrary numbers. It would clearly be impossible to store the value of the square root of N for
operation of pointers. Since these are not state variables, there is no way to see their values in either the Data Editor or log statements. Nor can we specify the value of those variables (in the modeler's sense of the word) as input--interactively or otherwise.\textsuperscript{12} As a side note, it is regrettable that workers on the modeling and computer-science sides of the problem use the same names for different entities, because this in itself is a frequent source of problems.

How do we recognize such entities when looking at RSAS rules? First, all functions that return a value (what might be called subroutines in FORTRAN) are referenced with specific syntaxes such as:

If \texttt{report from} Ask-force country-status using... is \texttt{<value>}
Then Let ...

In this case, Ask-force country-status is a function returning a value. It is not a state variable and does not therefore "have" a value. However, this rule could be rewritten in terms of a new variable that would be evaluated by: "Let new-variable be the report from Ask-force country-status..." Thus, if it is important to see what values are being seen by particular rules, then some modest programming will make that possible.

RAND-ABEL pointers can be recognized by keywords "function," "address of," "using the function," and "occupant of," as in the statement:

Perform Sleep-to-next-move using the function
Test-nuclear-weapons-use as planned-wakeup, and \texttt{((Today + 1) \times 24)} as time-limit.

\textsuperscript{12}This is closely related to the issue of modularity in software. If all data flowing to a program subcomponent can be inserted as input (as through a Data Editor or from a data file), then that program subcomponent can be individually designed and tested on a standalone basis. If the subcomponent cannot "run" without being linked up to other subcomponents, then it cannot be so tested. It is then not a true \textit{module} in the more careful usage of the term. This is equivalent to the statement that a true module must have an "internal representation" of all its incoming data.
This statement tells a particular military command (analytic war plan) to "go to sleep" until called or until certain events take place such as nuclear use. The syntax "using the function Test-nuclear-weapons-use" implies use of a pointer to the function Test-nuclear-weapons-use. As the result of this statement, later in the simulation when wakeup tests are being conducted, the particular function Test-nuclear-weapons-use will be invoked. If the test is positive, the National Command Level model will be awakened.

In practice, users will seldom have to worry about pointers, because they appear primarily in control-related statements such as the above one, rather than in the types of rules they will be most interested in.
The Special Case of Data from Force Models Written in C

Some of the rules in NCL models and other RAND-ABEL models have inputs from Force-C models that have not been translated into RAND-ABEL variables (e.g., the "Ask-force...") input mentioned above. How can we see the values of such inputs?

It so happens that the way RAND-ABEL models receive data from the Force-C models is by "reading" the same displays that humans can see in the Force window. That is, when they ask for data from the Force-C model, the model generates the same information it would if a human were asking for a display containing that data, but then sends the whole package of information as a stream of information bits for a special program to parse for the data of interest. This is inefficient computationally, but it has simplified developmental integration by assuring that all information important enough to be put into a human display will be available through a common mechanism to the decision models.

This is relevant to the current discussion because it implies that any Force data used by the RAND-ABEL models can be viewed in some Force display. Thus, you can track down the current value of an important Force datum determining a decision model's output. We will show how to find and read the relevant Force displays later in the Note, but will not do so here because the intended focus is the RAND-ABEL models themselves.

THE POSSIBLE ORIGINS OF VARIABLE VALUES

Suppose that we know the key variables, and even what their current values are. Often, this will not be enough to satisfy us. Our next question is how they came to have those values—i.e., where were their values "set." There are several places where variables may have been given their values:

• Assignment statements within the RAND-ABEL source code (for Red, Blue, Green, and Control Agents, and for System Monitor)\textsuperscript{13}

\textsuperscript{13}Standard assignment statements have the syntax: If <condition>
- 20 -

- From the Force-C program (called CAMPER sometimes)

- Input data to the WSDS-A (provided initially or during a simulation) (i.e., the variable may really be what a modeler would call a parameter or fixed data).\(^\text{14}\)

We will discuss the first two of these in Sec. IV. Let us discuss input data in more detail here. First, there is the possibility that a human specified some of the variable values interactively during the simulation by using the Data Editor. There is a log kept of such operations, and you will ordinarily know when they occurred. More troublesome, however, is the possibility that the variable was specified in "initial" input data, and you may not recognize that fact. You could waste considerable time searching for assignment statements in the RAND-ABEL source code when this happens.

In trying to track down where input data were provided to the WSDS-A it is important to recognize that the "initial state" is an ambiguous concept in the RSAS. Figure 11 illustrates this by showing how the "final initial state" (the Time 0++ state) is generated as the product of several types of operation. The possibilities for the origin of our variable values are:

- Initial input data contained in the first WSDS read in by the RSAS as they "come up" (refer to this as occurring at Time 0). If the RSAS was generated using the default WSDS, then the origin of the variable settings can be found in the Initialization files.\(^\text{15}\)

Then Let <variable name> be <value>. Table statements are also important, however, and these are more complex. Basically, the name of the variable of interest appears to the right of the "/" in the table heading, and the name may be spread over several lines to shorten horizontal space. This complicates search.

\(^{14}\)"WSDS" stands for World Situation Data Set, and the WSDS-A is that part of the RSAS WSDS that maintains values of RAND-ABEL variables (as distinct from variables in the "C" programming language used by many of the combat models).

\(^{15}\)The Initialization directory is at the same level as the Run directory, so you get to it in a shellwindow in the Run directory by typing "cd ..../Init". Since the directory is filled with files that would be very confusing to most users, it has been made possible to get at the most important of these files by looking instead under the relevant source directories. One pathname, for example, would be
Fig. 11--Establishing the Time-Zero RAND-ABEL World State (WSDS-A)

- Close-to-initial input data contained in WSDDS substituted for the original WSDDs after bringing up the RSAS, or by Delta WSDDS applied at that time, or both (refer to this as occurring at Time 0+).

- Settings dictated by the first running of Control Agent--before any other agents run (call this Time 0++). Suppose, for example, that the Control Agent included Scenario Generator tableau settings (also called User-generated data) having the effect of setting Red, Blue, and Green variable values at Time 0 (or, more precisely, when Control Agent runs for the first time at what we call here Time 0++). These settings would be part of the WSDD-A at Time 0+, but they would not have "taken effect" yet. Thus, if one brought up the RSAS and looked, at Time 0+, at various tableaus for Red, Blue, and Green, they would not yet show the variable values they would have "zero time later" after Control Agent runs. For this reason, we recommend that users avoid having nontrivial initialization assumptions in the baseline WSDD. Instead, to assure that

(relative to Run): ../Src/NCL/Red/Initialization/Initialization.A. In some cases you can reach the initialization files by using the walking menu (e.g., via RAND-ABEL rules/NCL/Red/Initialization).
such assumptions are highlighted, we recommend they be built into Delta WSDSs or Control Agent plans as discussed elsewhere. It is straightforward to develop reports on the content of the Deltas or Control Agent plans for future reference (see HELP file for the Data Editor).\textsuperscript{16}\

With this background on the full range of ways variables can obtain their values, let us now practice finding our way within the RAND-ABEL source code--i.e., moving around among the rules themselves. We shall include looking into the interface between the C and RAND-ABEL programs.

\textsuperscript{16}Warning: the Delta WSDS mechanism affects only RAND-ABEL state variables, and not Force variables in the C program part of the RSAS (mostly CAMPER). Thus, one cannot turn the clock forward or backward merely by reading in a new delta. One can, however, read in a complete WSDS to do so, which takes longer because it contains so much information.
IV. MORE EXAMPLES: UNDERSTANDING THE UNDERLYING RULES

PREFATORY COMMENTS ABOUT PHILOSOPHY

One of the design ethics behind RSAS architecture and the RAND-ABEL language has been the view that analytically inclined users should be able to review and modify much of the underlying base of rules—particularly rules dealing with concepts of operations, larger strategies, and political-military judgments—but also many of the rules important in the adjudication of combat (see Allen and Wilson, 1987). For this to be possible, it is strongly desirable that users be able to work directly with the source code rather than having to rely upon documentation or the assertions of programmers, who may not understand precisely the questions being asked or the instructions being provided.

Another ethic has been the vision that senior analysts should be able to run the RSAS by themselves (e.g., late at night or over a weekend), making rule changes as needed to test alternative concepts and conducting sensitivity studies with little or no programmer assistance—assuming appropriate preparations.

The above philosophy and goals are fine in principle, but there are definite thresholds for a new user to cross. Until they are crossed, claims about RAND-ABEL's readability and clarity will seem like hype. Let us therefore walk through the kinds of operations you will find necessary to understand RAND-ABEL code. Some of these will be simple, but others are admittedly complex—not so much because of RAND-ABEL limitations, but because the RSAS and its component models are complex. The intention here has been to expose you to the types of problems that have proven most annoying and troublesome to RAND users. Again, then, we proceed by example.
ILLUSTRATIVE OPERATIONS IN UNDERSTANDING DECISION RULES
Finding the Set of Rules Relevant to a Model Decision

Suppose that we are trying to understand (and perhaps improve) the rules for Blue's assessment of strategic warning in the SAMN National Command Level model. Perhaps we had just noted a log entry saying that SAMN had assessed strategic warning in Europe to be "Eur-conv" (i.e., warning of conventional war in Europe), and we wanted to know why, and also whether he "should" be seeing warning of nuclear war. Since this is a rather detailed level of information, the logs are not very informative (although they could readily be expanded and the exercise rerun on any given subject such as this). Thus, we must look at the underlying source code. We use the walking menu to select Assess/Warning/Strategic-warning. A for SAMN and then we search within the file for "EUROPE," whence we see in the window the beginning of the submodule evaluating strategic warning in the European theater.

Gaining a Top-Down View
Having found the rules of interest, we now go back to the beginning of the file to look for overall rationale. In this case, there are indeed comments providing such rationale—as indicated in Figs. 12 and 13. These top-of-file comments describe the contents of the file, the inputs and outputs, values of the key variables, and a description of the modeling approach. They do not try to paraphrase the rules as would be the practice with other programming languages. Here we should look directly at the rules themselves and not rely on such paraphrasing.

Identifying and Understanding Key Variables
The inputs for the entire strategic-warning model were provided in top-of-file comments, but those unique to European warning were not highlighted. Thus, one question is what those inputs are. Log statements can omit (intentionally or unintentionally) some of the variables actually used. The rule in looking for inputs is simple: The inputs (independent variables) are to be found on the left side of assignment statements, either in the form of "If <input> is <value>..."
**Strategic Warning A**

**U.S. Warning: Blue**

[Insert diagrams and tables here]

---

### Strategic Warning

**Purpose:** Establishes strategic warning and the credibility of such warning for major theaters and globally.

Strategic warning affects the psychological mindset of MOC models. It also affects decisions directly in some instances (e.g., launch-under-attack situations).

Both globally and on a theater-by-theater basis:

- **Strategic Warning Variables** have the range:
  - (None, Regional-conv, Regional-nuc, Eur-conv, IC-conv, Eur-nuc, Eur-strat, IC-strat)

We distinguish Eur-nuc from Eur-strat because the delivery means for tactical and operational-level attacks are so different from those on strategic targets. Hence, there may be a basis for a distinction in warning. No such distinction is made in the intercontinental arena.

- **Credibility-of-Warning Variables** have the range:
  - (None, Low, Medium, High)

What follows examines strategic warning theater-by-theater, and then folds the results together for a global assessment.

The "basic" approach for each theater is the same:

- Identify one or more big indicators to be explicit
- List "other" indicators to be added up
- List "other" indicators of nuclear escalation to be added
- Build the table in terms of the big indicator(s), the total "other-nuc-indicators", the total "other-indicators", and the current state.
- To be sure that warning is turned off after the event warned against occurs

There are, however, various special cases.

**IC-situation, SNA-situation, etc.** have the range of **Type-conflict-level**:

- (Peace, Crisis, Nonsuperpower-conflict, Superpower-intervention, Demo-conv, Regional-nuclear, Gen-conv, Demo-tac-nuc, Gen-tac-nuc, Demo-strat-nuc, Gen-strat-nuc)

---

### Inputs:

- Red-SSBN-ASW
- Red-SSBN-ASW-status
- Recent-SSBN-attrition (over last 2 days) numerical
- Setting-up-bastions [flag]
- USSR-SSBN-displacement [flag]
- (Alert rates of Soviet H-bomb and SSM weapons)
- Red-available-warheads in IC
- Critical-intelligence in IC and Europe
- USSR-civil-defense-preps [flag]
- Space-situation (Situation of Space)
- European-situation
- Highest-alert-posture: {Normal, Alert, Superalerted}
- Red-strat-ASAT
- Red-strat-alert
  - Keeps track of past alert postures as part of deescalation test

---

**Fig. 12—Illustrative Top-of-File Comments in Source Code**

(Note: these unclassified rules are simpler than those that might be needed in studies.)
or as column headings on the left side of the "/" symbol in RAND-ABEL tables. We could search for and then itemize all of these in short order, but shall not bother to do so here. Figure 14 shows a decision table within the portion of the code dealing with strategic warning in Europe. We see that two of the independent variables are "other-nuc-indicators" and "PACT-mobilization." Let's now investigate these variables further.
Fig. 14--Illustrative Decision Table in Strategic Warning Model

**Tracing Back the Meaning and Evaluation of Variables: A Simple Case**

The principal challenge in reading and understanding the NCL models as they appear in RAND-ABEL source code is tracking backward to find out how relatively high-level variables are evaluated in terms of lower- and lower-level variables until, finally, you are satisfied that you "understand." Let's start with the variable "other-nuc-indicators," which represents an easy case. It is not capitalized, which by RSAS
convention means that the variable has been declared and evaluated within the same function. In programming terms, it is supposed to be a "local variable," one that would not appear in the Data Dictionary or WSDS. Searching backward for the string "other-nuc" (remembering that searches can miss things if the item is spread over two lines, as in a table heading, so that searching for the shortest unique feature is wise), we find that the variable is declared at the top of the strategic warning model (see last part of Fig. 13). It is then initialized to 0 at the top of each theater's submodel (e.g., at the top of the submodel for European strategic warning), and then evaluated using theater-specific indicators.

Continuing, we then see rules such as

If PACT-nuc-weapon-dispersal is Yes Then Increase other-nuc-indicators by 1.

Looking over these rules, it should be rather apparent—just from the names of variables—what the rules determining the indicators are. In these cases, you might need to go no further to "understand" this part of the model. Sometimes, then, explanation is easy.

More on Tracing Back Variables: A Less Simple Case

Let us next consider one of the other rules within the module for strategic warning in Europe. It states:

If PACT-IRBM-dispersal is High Then Increase other-nuc-indicators by 2.

This is intuitively reasonable, but what is meant by "High"? We need to see the rule evaluating PACT-IRBM-dispersal.

We might try to find the evaluation of "PACT-IRBM-dispersal" within the current file, but it isn't there. So where do we look? The most important advice here is to stop and think—using our substantive knowledge rather than relying entirely on technological devices. After all, the NCL models (and other RAND-ABEL models) are highly structured,
and this structure is based on separating logically distinct concepts. So, where would PACT-IRBM-dispersal be introduced and evaluated?\footnote{To avoid wasting time, we should always consider the possibility that this is a parameter or a data item--i.e., something specified externally and then held constant in the simulation unless there is further intervention. For example, if the variable in question were Cold-war-climate, we might reasonably expect that this would be an exogenous parameter establishing a game context and look at the tableau for NCL parameters (there is indeed such an NCL parameter found in the tableau sets entitled play-NCA.T and play-DEFC.T).} 

Asking the question is tantamount to answering it. If we look at the walking menu again (Fig. 15) and begin moving rightward, we should expect to find it in the Assessment module (as distinct from the modules on adjustments of assumptions, deciding objectives, and so on). Moreover, it is essentially "data" being used for decision, it is military data, and it relates to particular military actions. Hence, we might select the corresponding file from the menu (Assess/Data/Military/Military-actions.A) (see Fig. 15). Indeed, we there find "PACT-IRBM-dispersal" referred to in comments, and then in assignment statements that establish its value. In searching for assignment statements, the rule is simple: a variable is evaluated by statements with the syntax "let <variablename> be <variablevalue>," or in tables containing the variable as a column heading on the right side of the "/" sign.

Fortunately, the "Find-Item" Tool makes these searches easier and more systematic.\footnote{The Find-Item Tool was designed and implemented by Robert Weissler and the author. There is good documentation in a HELP file, and you should read it because there are some subtleties and tricks involved.} Had we been unsure whether PACT-IRBM-dispersal was considered "Data," we could have used the Find-Item Tool from the Assessment menu (Fig. 16). We could have speeded the search by selecting the option to find where a variable is evaluated--thereby eliminating references in comments, for example. RSAS experts may go directly to the right file and search, but most users will be better advised to use the Tool. As shown in Fig. 16, the Tool sometimes "finds" items inappropriately: Upon searching in two of the four files
Fig. 15--Using the Walking Menu to Select Military-actions.

indicated, we would find that "Let PACT-IRBM-dispersal" appears only within declaration statements (as an example in terms of which new variables are being declared).

Pursuing this example, we find that PACT-IRBM-dispersal is evaluated by the following:

If Yes is the report from Ask-force-theater-action-status using

Red as actor,
IRBM-dispersal as action, and
Central-Europe as theater
Then Let PACT-IRBM-dispersal be High.
Else Let PACT-IRBM-dispersal be Normal.
Fig. 16--Using the Find-Item Tool to Search for IRBM
This tells us two important things. First, the statement indicates that the criterion being used is very simple and the variable is either "High" or "Normal." Thus, there is a simple threshold being used. The second thing we learn is that this is the lowest level within the NCL code. To go further—e.g., to find out what the threshold is—we must go to a function that exists in the interface between this model and the Force Agent. Why? Because we see that the evaluation depends on a function prefixed by the special phrase "Ask-Force."

Finding Variables in the Force Interface

Suppose that we really wanted to pursue the evaluation farther, not being satisfied with the explanation because we want to know the value of the threshold: Is dispersal of one IRBM enough to trigger this assessment? Because the information evaluating FACT-IRBM-dispersal comes from a function in the interface, we go back to our walking menu and look for Interface (see second menu in Fig. 15). We see (Fig. 17) that we have choices entitled "To-Green," "To-Force-C," "To-Force-A." It is rather obvious that we want one of the "To-Force" files, but which one? We could use the Find-Item Tool from the top of the Interface menu, but the search would take longer than if we guess right.

---

Fig. 17—Using the Walking Menu to Select Interface/to-Force- C/query.A
There are no hard-and-fast rules on these matters, but some guidelines are as follows. First, there are a number of Force Agent models that are written in RAND-ABEL. These include most notably the combat models for the "secondary theaters" (those other than Central Europe, Korea, and the intercontinental theater). These models are collectively known as Referee, because the rules within them serve to adjudicate the results of combat under opposing strategies. Second, nearly all of the main-theater combat models are written in the C programming language. And, finally, there is a special set of "flag models" that are used to keep track of strategically or politically important symbolic actions that are not otherwise simulated in the RSAS.

To elaborate on this, note that the RSAS does not model bacteriological warfare, but if it happened somewhere in the world, the fact of it happening (if known) might be an important factor in the decisions of RSAS agents or human RSAS players. Thus, we permit agents and players to issue orders for flag events. Those "flag models" are really just a software mechanism for assuring that if Red or Blue takes one of a specified set of "flag actions" (the metaphor is to climbing a hill and posting one's flag as a symbolic indication of conquest), then the fact of that flag action will be in the WSWS and available for the other side to see, after an appropriate delay.

Reviewing, then, the function we are looking for could be in several places associated with the Force Agent's interface with RAND-ABEL models. It would seem a good guess that it would be in the To-Force-C interface, and indeed that is where it "should" be. Within that interface there are a number of separate files with names that are not especially friendly. For variables of interest to the NCL, the usual files of interest are query.A (the NCL is "querying" the Force Agent for information) and the lower-level display.A, which is most useful to programmers rather than analysts. So, we might select and search in query.A. (Fig. 17). In fact, the search fails.\(^{3}\) We might then search with the Find-Item Tool at the top of "To-Force-C," but that search fails also.

\(^{3}\)As of January 5, 1988. By the time you read this, linkups may be
As a practical matter, the usual reasons for a "reasonable" search failing are mistyping or the failure to use a small enough string of letters to avoid the problems the editor has when a name is spread over two lines, as in RAND-ABEL tables. Thus, we might try it again being more careful and perhaps searching for something like IRBM. Nonetheless, the search fails--because in this case, the function is not where it should be ideally.

Functions and Variables Within the Flag Model. Since the search failed, the function in question "must be" on the RAND-ABEL side. It could be in a Force-ABEL (also called ABEL-Force) model simulating events, or it could be in the more symbolic "flag model," which is also in RAND-ABEL. Because guessing wrong again can be frustrating, we might be wise to use Find-Item from the top of the To-Force-A menu. For example, we might type "IREM" and select the option to "find function definition." In fact, this still fails (as of January 6, 1988). The reason for using this example is that it illustrates what may be the "worst case," or close to it, for finding your way around the NCL model and its interfaces.

The reason the search fails is technically obscure and has to do with the fact that the flag model is written in a generic way. The model doesn't know "IREMs" from a hole in the ground. Instead, it is just a software mechanism for manipulating symbols, and the symbols it can accept are listed elsewhere in a Data Dictionary file. We might just "know this," of course, in which case we would be ok. However, it is comforting to note that if we try the Find-Item-Tool search again selecting the option "Find all references to name (including comments)" because we are getting frantic, indeed we find that IRBM is referenced in a particular dictionary file. Upon pulling up that file we find IRBM-dispersal in a list of possible values for "Type-action." This list of values for Type-action is in fact the list of permitted "flag actions" handled by the flag model. Had it occurred to us to check

different and you may indeed find the function where it "should be," but the example is instructive.

"The flag-model mechanism is described in on-line documentation (Documentation/Flag/flag.A from the walking menu). It was developed by
earlier, we could have determined that the variable was a flag by merely entering CR-Tool and asking for the enumerated values of Type-action! That would have saved considerable time, just as it is wise to recognize early that a variable may actually be an exogenous parameter and check input data accordingly.\footnote{One can use the CR-Tool to find the variables that use a given value. Thus, we could type "IRBM-dispersal" as input and select item 5 in the CR-Tool menu, "Enumeration names for this value." We would then see, in this case, that it is a value of Type-action.}

At this point, then, we now know that IRBM dispersal is evaluated by looking at a flag action: It either occurs or it doesn’t. By the way, the flag model also permits players or agents to take flags down (in this case, to return IRBMs to a normal, undispersed, status). We may quarrel with the flag model (preferring that the IRBM dispersal be explicitly simulated, as indeed it will be), but insofar as understanding the NCL model, we have now come to the primitives: We now understand completely what a PACT-IRBM-dispersal of High means and how that contributes to strategic warning in Europe.

Another Tracing Problem: A Rather Complex Case

As our next example of a difficult tracing problem, let us consider the issue of mobilization. Up in the strategic-warning code, one of the prime indicators is PACT-mobilization. If we go through the same procedure ("Where would this be evaluated? Ah--it’s an assessment variable, it’s data, it’s military, so it should be in Military-actions.A"), we readily find that PACT-mobilization is set to Yes if Yes is the report from another "Ask-force" function, this one "Ask-force-theater-action-status."

Proceeding again to look in the interface, and to guess that the information should be in "To-Force-C," and indeed in "query.A" (or using Find-Item Tool from a higher level), we find the function without trouble. We then search on "Mobilization" (the specific action being tested) and find the rule:

Robert Weissler and Barry Wilson, following suggestions by the author stimulated by his interest in reflecting important events that are difficult to simulate in detail (Davis, 1982).
If action is Mobilization
Then
{
   If Yes is the report from Test-mobilization...
}
Then...Yes.

That is, we find yet another level of searching is necessary. As is almost always prudent, we search backward in the same file (query.A), and in this case we find the function Test-mobilization there. However, upon reading through it, we find that the rules germane to our problem depend on a variable "Nominal-mobilized-EDs in region, actor" (this syntax is complicated because the "variable" is really an element of a matrix—i.e., a two-dimensional array of values). The full function is shown in Fig. 18. Note that at its top there is a comment explaining that "nominal level" of mobilization is either set by the analyst (i.e., as an input parameter), or—as a default—is automatically calculated by the RSAS as the day 0 level. This is an interesting and important comment, because one could construct initiating scenarios in which full mobilization had already occurred at time 0, and the default RSAS rules would then be in error in their evaluation of whether the PACT was mobilized. Ideally, the program would include a reminder to the reader saying "In this run the RSAS is assuming that Time 0 force levels represent nominal levels, so that mobilization is defined as mobilization in excess of those levels" (or a terse equivalent). Few programs anywhere are written with such a degree of hand-holding, in part because managers have not required them, and in part because too many such comments can be distracting.

As our last example of backtracking to "fundamentals," suppose that we don't believe or understand the comment about how nominal mobilization is determined. We conduct yet another search, this time for "Nominal-mobilized-EDs," using Find-Item from the top of the To-Force-C menu (with more knowledge we would "know" to look in display.A, because that is the file from which query.A gets its information most of the time). The result is that we see in the file "display.A" the following:
query.A

[ ******************** Test mobilization ******************** ]
[
(U) This situation characterization function returns Yes
if the side is mobilizing troops in the region. Mobilization
beyond the nominal peacetime level is considered a crisis
action. The nominal level of mobilization is either set
explicitly by the analyst in the Data Editor before the game
begins or is automatically set to the day 0 level.

Usage:

If the report from Test-mobilization using
Blue as actor, and
USSR-W as region is Yes
Then ...
]

Define Test-mobilization:

Declare fud-committed: Let fud-committed be 1.0.
Declare from-reserve: Let from-reserve be Type-overlay.

If actor is White
Then Exit reporting No.

Decision Table

<table>
<thead>
<tr>
<th>actor</th>
<th>region</th>
<th>from-reserve</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue</td>
<td>FRG</td>
<td>CEUR-res</td>
</tr>
<tr>
<td>Blue</td>
<td>South-Korea</td>
<td>FEAST-res</td>
</tr>
<tr>
<td>Red</td>
<td>GDR</td>
<td>WTVD-res</td>
</tr>
<tr>
<td>Red</td>
<td>Czechoslovakia</td>
<td>WTVD-res</td>
</tr>
<tr>
<td>Red</td>
<td>North-Korea</td>
<td>FETVD-res</td>
</tr>
<tr>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

If from-reserve is not Unspecified
Then [ Check forward committed troops moving toward the FLAT ]

{ Let fud-committed be the report from Ask-force-overlay-data using
  FLAT-EDs as data, and
  from-reserve as overlay.
}
Else Let fud-committed be 0.0.

If ((the report from Ask-force-region-data using
  actor as side,
  Mobilized-EDs as data, and
  region as region) > fud-committed)
Then Exit reporting Yes.

Else If Yes is the report from Test-polling using
actor as actor, and region as region
Then Exit reporting Yes.

Exit reporting No.

End. [ Test-mobilization ]
If Monitor's Time <= 0 [i.e., the beginning of the game]
Then
{
If Nominal-mobilized-EDs in each-region, Red <= 0.0
Then Let Nominal-mobilized-EDs in each-region, Red be
  Floatarray3 of index.
}[items deleted]

Clearly, this is real computer code, not the most user friendly of possible RAND-ABEL rules. However, even this is understandable with a bit of specialized knowledge. First, one has to understand about Monitor's Time (but there's a comment to help). Second, one has to understand that the variable we are interested in is listed as one part of an array (hence the references to each-region, and so on—the larger function is conducting a loop operation over all regions). And, last, one has to understand that the very lowest level of the RAND-ABEL code and its interface to the Force Agent's C programs is a complicated utility program that provides information to the RAND-ABEL models using the same array of Force data that humans see in the Force Agent window as displays (e.g., the familiar tabular display showing FLOTs, and force levels in the Center Region). The program might know, for example, that the data needed is the 7th item on the second line or something equivalent. That is the type of thing referred to by the reference to

---

6There is extensive on-line documentation by Robert Weissler regarding the Force/RAND-ABEL interface, which was developed by Robert Weissler and Barry Wilson using a display parsing program by Mark LaCasse. To read this, select Documentation/Force-interface from the main background menu. From a technical and managerial point of view, the interface is interesting because the mechanism guarantees that any information available to a human in a parsable Force display is or can readily be made available to RAND-ABEL agents. This reduced integration problems, but at the expense of slow communications between the C and RAND-ABEL agents—something responsible for a significant portion of total RSAS run time. An improved-performance interface has been added, and will gradually displace the one described here.
Data Editor or a Control Agent program, none of the above statements' IF tests will pass, and the player's inputs will hold. Thus, the comment explaining this was true.

Again, then, we have reached the very limits of the explanation trail—at least with respect to the models written in RAND-ABEL. Ultimately, the value of Nominal-mobilized-EDs for the Central Region theater in Europe depends either on input or on whatever the Force Agent models say are in-place forces as of Time 0.

Tracking Back to the Force Agent Data. It is worth going the extra step at this point to illustrate how to find the value of Force Agent variables used by the RAND-ABEL models. Using the current example, the challenge is to find what the Force Agent is reporting for Nominal-mobilized-EDs. The procedure is as follows:

- Having found the low-level interface code referring to our variable, we now page backward to find what function it is in. We see it is in the function "Display-region" (Fig. 19), which means by RSAS convention that the relevant Force display is the "region display." In later versions on the system one will see "Dump-region" instead, where "dump" indicates use of the newer interface.

- Next, we select the Force Agent from the Control Panel and, upon getting a prompt, type "display region." We then see rolling past us a very long display with only minimal header information telling us what we are seeing. It also rolls by too quickly to read.

- We depress the right mouse button and select "enable page mode," which will slow the display so that we can advance it in stages with the space bar. Try it by typing, again, "display region." Now we can read the display (Fig. 20). Upon thinking about it for a moment, however, you may conclude that you’re not sure what the data mean.

- Type "display help region," which will bring up a HELP file explaining the data shown in the region display (Fig. 21). Upon reading this you will see that deciding precisely what to tabulate under the simple-sounding name "mobilized EDs" is not so easy for modelers and programmers. Hence, the HELP file is important.

Ordinarily, a user of the RSAS will not have to trace data back this far, but it should be comforting to know that you can do it without being a "real programmer."
Fig. 19--Finding What Force Display Generates the Data

Fig. 20--The Force Display "Region"
(data values omitted to avoid classification)
The region display counts GROUND forces only in 4 categories as described below. It gives a response for all RSAS LAND regions (one row each) as in the following (partial) example of the display.

<table>
<thead>
<tr>
<th>Region</th>
<th>Status</th>
<th>Total EDs</th>
<th>Mobilized EDs</th>
<th>Deployed EDs</th>
<th>Poised EDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>US-NC</td>
<td>Peace</td>
<td>8.0</td>
<td>2.5</td>
<td>0.0</td>
<td>0.7</td>
</tr>
<tr>
<td>US-NE</td>
<td>Peace</td>
<td>8.0</td>
<td>3.6</td>
<td>0.0</td>
<td>0.9</td>
</tr>
<tr>
<td>US-WP1</td>
<td>Peace</td>
<td>8.0</td>
<td>0.5</td>
<td>0.0</td>
<td>0.1</td>
</tr>
<tr>
<td>US-EC</td>
<td>Peace</td>
<td>8.0</td>
<td>0.7</td>
<td>0.0</td>
<td>0.9</td>
</tr>
<tr>
<td>US-WP2</td>
<td>Peace</td>
<td>8.0</td>
<td>5.2</td>
<td>0.0</td>
<td>3.2</td>
</tr>
<tr>
<td>etc.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Column (header)**

**Description of Data**

1. **Region**
   - Name of RSAS LAND region (implication being that if a ground force is area when the display is requested, as a marine force aboard amphibious lift or any ground force encounts via strategic sealift, the force is NOT in the display totals. Forces being airlifted count in the land region of origin until the lift is completed.

2. **Status**
   - One of:
     - Peace: No hostilities have ever occurred here
     - Limited: Nothing worse than air attacks has occurred
     - Conventional: Nothing worse than land combat has occurred
     - Chemical: Nothing worse than chemical use has occurred
     - Nuclear: Nuclear weapons have detonated here
     - 'Worse than' means 'anything below in theenumeration'.
     - The promotion down the list is irrevocable in force (i.e. there is no code anywhere which can 'degrade' the status of a land region once it has been promoted).

3. **Total EDs**
   - Total surviving EDs (not count EDs not in any nation's forces)

4. **Deployed EDs**
   - Total surviving EDs of above forces multiplied by each counted forces CURRENT level of mobilization.

5. **Poised EDs**
   - Total surviving EDs of above forces that are located in regions that are part of Major or Sub Theater Overlays multiplied by each forces current mobilization level. Thus, there can be no 'Deployed EDs' in (say) US-SC since US-SC is NOT in a theater overlay.

6. **Poised EDs**
   - Total surviving EDs that are currently located in an area of a Major or Sub Theater, or operating in the same rear areas of such an axes (Army, Airborne, etc) multiplied by each forces current mobilization level. Thus, there can be no 'Poised EDs' in (say) US-SC since US-SC is NOT in a theater overlay.

**It's now 0818 08T, day 8. Choose an option:**

---

Fig. 21--HELP for the Force Display "Region"
COMMENTS ON VARIABLE TRACING IN OTHER RAND-ABEL MODELS

Our examples have been drawn exclusively from the NCL models, but the same general principles apply to the others. There are a few items worth noting, however, because they can reduce search time somewhat.

- The military command level models (analytic war plans) depend on the to-Force-C interface functions (red-order.A, blue-order.A, and global-order.A) and utility.A in much the same way that the NCL models depend on query.A. They also make use of variables in display.A.


- When searching within analytic war plans, you should be aware that many standard packages of orders are included in the function library.A for the relevant command. Also, you should remember that authorizations must come from a higher level command. Thus, if you see something like "If authorization-in...is Yes...," then to find out what determines whether the authorization is or is not granted you may have to look at each level of the command hierarchy above the current one. Consider the authorization to mobilize in AFCENT. The NCL may set the authorization, but if it doesn't then the JCS model may, and if it doesn't the EUR model may, and if it doesn't, then ordinarily AFCENT will assume authorization is No. The complexity here has nothing to do with the RSAS or RAND-ABEL, but with the complexities of real-world command and control.

- Sometimes, interface variables are not where you might think they would be from their names. A simple trick that can help in this respect is using the CR-Tool to see where they are declared. For example, you might be looking for the function Test-enemy-mobilization, because it is referenced in an analytic war plan. You might expect that it would be in the Force interface, but you would not find it there. You might worry about having made mistakes (mistyping or whatever), but if you used the CR Tool, you would see it is declared in "./..Src/NCL/Red/Dictionary/IVan.D." While you don't want to see its declaration, this tells you that you were looking in the wrong interface. You should have been in "To-NCL." The reason, in this case, is that it is the NCL models that determine the wakeup rules, so it is there that the wakeup test is to be found. Logical—in retrospect.
Ask-force

Go to relevant Interface file (on menus).
Is it S-Land data or flag data?
If yes, go to To-Force-A.
If no or ?, try To-Force-C first.

Check that variable may be a flag in
To-Force-A/dict/flag.A
or CR-Tool (values of Type-action)

Check RAND-ABEL-force models within
To-Force-A

Is it of form Ask-force or Ask-green?

No

Search within current module

Does it seem like an input parameter?

Yes

Look in relevant Data Editor tableaus.
Search in relevant tableau-set T file.
Search in relevant Initialization files.

No

Think. What type of information is involved. Search in relevant submodel.

See if it is an input parameter

Ask-green

Look in relevant file (query.a for information, message.A for message-related tests and bounds)
|--------------------|---------------------|-------------|-------------|--------------|-------------------|------------|

**When Seeking Explanations of RSAS Runs**  
(e.g., as players or analysts)

<table>
<thead>
<tr>
<th></th>
<th>Sources of Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variables responsible</td>
<td>1</td>
</tr>
<tr>
<td>Meanings</td>
<td>1</td>
</tr>
<tr>
<td>Values permitted (ranges)</td>
<td>1</td>
</tr>
<tr>
<td>Values determining decision</td>
<td>1</td>
</tr>
<tr>
<td>Underlying rules and algorithms</td>
<td>1</td>
</tr>
<tr>
<td>Rationale for models or values</td>
<td>1</td>
</tr>
</tbody>
</table>

**When Reviewing or Improving Rules**  
(e.g., as modelers or analysts)

<table>
<thead>
<tr>
<th></th>
<th>Sources of Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variables responsible</td>
<td>2</td>
</tr>
<tr>
<td>Meanings</td>
<td>2</td>
</tr>
<tr>
<td>Values permitted (ranges)</td>
<td>1</td>
</tr>
<tr>
<td>Values determining decision</td>
<td>1</td>
</tr>
<tr>
<td>Underlying rules and algorithms</td>
<td>2</td>
</tr>
<tr>
<td>Rationale for models or values</td>
<td>2</td>
</tr>
</tbody>
</table>

*Actually, analysts and modelers will only seldom use general model documentation, and then for getting the right top-down sense for the model as a whole. Most of the time they will be working directly with the RAND-ABEL source code, especially since documentation is always obsolete.*

Fig. 23--Sources of Information and Explanation
V. SOME SUMMARY PROCEDURES AND SUGGESTIONS

In the preceding section we marched through a number of specific examples that illustrated some easy and some very difficult variable-tracking challenges—the latter of types that have caused considerable distress even for RSAS users within RAND who thought they understood the system (but had no primer such as this to help). At this point we need to shift toward more general guidance. There are no completely general rules to make using the RSAS (even its RAND-ABEL side) easy, and there will definitely be many times when you will need the assistance of professional programmers, but there are some useful guidelines that will help most of the time.

Figure 22 will always work if you use the Find-Item Tool at a high enough level in the directory tree (albeit, at the expense of search time). Doing so may be less frustrating than guessing wrong about which file to look in within a generally correct directory. Often, failure leads to paralysis. The tree supposes that you are trying to track down the evaluation logic for a particular variable. You start at the top of the procedure diagram and continue until you find the assignment statement. Warnings: (1) Be careful of typographical errors (including misuse of capitals); (2) check for such errors before shifting from a logical search area to one that seems illogical; (3) do searches for shortest distinctive fragments of names, or you may miss the items because of name splitting in table headings; and (4) reuse the Tool specifying the most general search (the one looking for the string anywhere, even in comments) before giving up on a highly logical file, because sometimes the Find-Item Tool is fooled.

Finally, Fig. 23 summarizes this discussion by suggesting the order in which you might use the various tools to answer the questions identified in Sec. II. Part 1 of this figure assumes that you are a user more concerned about "explanation" than with thinking about ways to enhance the model; part 2 is more relevant to an analyst or rule-writer.
REFERENCES TO THE RSAS AND ITS TOOLS

The user of the RSAS may wish specifically to look at the following. The forthcoming items will soon be published. The unpublished items can be made available to bona fide RSAS users upon request to the individual authors or to the present author. They should not be requested through RAND's Publications Department.

UNPUBLISHED AND FORTHCOMING DOCUMENTS


LaCasse, Jean, Defining Displays for the RSAS Data Editor, The RAND Corporation (forthcoming).


If you lack background in modern programming (e.g., if you are unfamiliar with "structured programming"), you should consider studying the elementary chapters of a textbook for PASCAL, a good structured-programming language used widely in undergraduate courses and one for
which good textbooks exist. With such background you will find
RAND-ABEL® easy to understand, since the underlying basics and
terminology are similar.

PUBLISHED DOCUMENTS

Allen, Patrick D., and Barry A. Wilson, Secondary Land Theater Model,

Davis, Paul K., Applying Artificial Intelligence Techniques to
Strategic-Level Gaming and Simulation, The RAND Corporation, P-7120,
November 1985. [Published also as chapter in M. S. Elzas, T. I. Oren,
and B. P. Zeigler (eds.), Simulation in the Artificial Intelligence

Davis, Paul K., Concepts for Improving the Military Content of Automated
War Games, The RAND Corporation, P-6830, November 1982.

Davis, Paul K., Steven C. Bankes, and James P. Kahan, A New Methodology
for Modeling National Command Level Decisionmaking in War Games and

Hayes-Roth, Frederick, Donald A. Waterman, and Douglas B. Lenat (eds.),
Reading, Massachusetts, 1983.

Paul, Jody, Self-Revealing Software: A Method for Producing

Shapiro, Norman Z., H. Edward Hall, Robert H. Anderson, and Mark
LaCasse, The RAND-ABEL® Programming Language: History, Rationale, and

Shlapak, David A., William L. Schwabe, Mark A. Lorell, and Yoav
Ben-Horin, The RAND Strategy Assessment System's Green Agent Model of
Third-Country Behavior in Superpower Crisis and Conflict, The RAND

Waterman, Donald A., Jody Paul, Bruce Florman, and James R. Kipps, An
Explanation Facility for the ROSIE® Knowledge Engineering Language,