GUESS/1: A General Purpose Expert Systems Shell

by

Newton Saiyuen Lee

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science and Applications

APPROVED:

John W. Roach

Osman Balci

John F. Kramer

March, 1985
Blacksburg, Virginia

GUESS/1: A GENERAL PURPOSE EXPERT SYSTEMS SHELL

by

Newton Saiyuen Lee

(ABSTRACT)

Expert systems are very useful and probably the most fruitful products of applied artificial intelligence. Expert systems, however, are very expensive to develop. Powerful construction tools are indispensable to construct, modify and maintain a practical expert system. GUESS/1 is a domain-independent expert systems shell that captures and enhances the strengths of its predecessors while at the same time overcoming their limitations. GUESS/1 gives a strong emphasis on human engineering, language generality, diversity of data representation and control structures, programming and run-time environment, database construction facilities and security, and many other aspects that are related to the ease of development and maintenance of expert systems.

ACKNOWLEDGEMENTS

I thank my principal advisor John Roach with whom I have done my research on artificial intelligence for two years. It has been very challenging and enjoyable.

I thank Osman Balci, , John Kramer and for their influence on the development of GUESS/1.

I thank and for their tremendous help in developing DIES-II and its predecessor.

I thank who read chapter XI and XII of the thesis and provided valuable suggestions. She, and I are the very first users of GUESS/1.

I thank the Association for Computing Machinery, Washington D.C. Chapter, for offering me the 1984 Master's Degree Grant to work on this thesis.

I thank all those people who had encouraged me during my four and a half years of college study. Among them are and

I thank all those people who had supported me financially at one point during my four and a half years of college study. Among them are and my parents.

I thank and for offering me chances to participate in their exciting research projects (Machine Learning and the CAIS). They have been providing me with valuable work experience.

I thank the who have been so nice to me ever since I came to know them:
and and their puppy
They have broadened my vision of life.

I thank my friends whose playfulness has been adding amusement to my daily life and contributing much to my sanity.

Finally, I thank God who teaches me how to live.

This thesis is dedicated to
and
who make me feel at home in America

TABLE OF CONTENTS

ABSTRACT ii

ACKNOWLEDGEMENTS iii

Chapter page

I. INTRODUCTION 1

Expert Systems Architecture 2
Expert Systems Applications 8
Motivation for GUESS/1 9
General Overview of GUESS/1 11
A Brief Reader's Guide 14

II. DESIGN PHILOSOPHY 16

Representation Language 16
Data Representation 18
Control Knowledge 20
Explanation Subsystem and Debugging Aids . . . 24
Human Engineering 25
Interface to the External World 26
Database Security 27

III. REPRESENTATION LANGUAGE 28

Example 1 30
Example 2 30
Example 3 32
Example 4 32

IV. DATA REPRESENTATIONS 37

Relational Tables 38
Hierarchical Trees 42
Semantic Nets 46
Knowledge Frames 50
Blackboard 55
Knowledge Frame Segment 55
Action Frame Segment 56
Object List Segment 57
More Complex Representations 58

V. CONTROL KNOWLEDGE 60

Action Frames 61
Production Rules 66
Intermix of Action Frames and Production Rules 66
Multiple Knowledge Sources 70
More Complex Control Structures 72

VI. EXPLANATION SUBSYSTEM AND DEBUGGING AIDS 75

Explanation Subsystem 76
Debugging Aids 78

VII. HUMAN ENGINEERING 79

Natural Language Capability 80
Menu Facility 82
Question-Answering System 82
System Characteristics 86
System Commands 87

VIII. INTERFACE TO THE EXTERNAL WORLD 88

An Example Interface Program 89

IX. DATABASE SECURITY 91

Fundamental Requirements 92
Mandatory and Discretionary Security Protection 94

X. GUESS/1 LIBRARY ROUTINES 98

Blackboard Manipulation Routines 100
Explanation Subsystem Routines 104
GUESS/1 Initialization and Execution Routines 105
Hierarchical Tree Manipulation Routines . . . 107
Human-System Interface Routines 112
Miscellaneous Utility Routines 117
Object-List Manipulation Routines 135
Relational Table Manipulation Routines . . . 138
Semantic Net Manipulation Routines 142
Stack Manipulation Routines 149
Standard SSA_MAP Routines 151
Syntax-Semantic Action Mapping Routines . . . 152
SYSHELP Routines 154
System Security Routines 155
Terminal Input/Output Routines 157
GUESS/1 Library Routines Distribution 159

XI. APPLICATION: THE DRUG INTERACTION EXPERT SYSTEM 161

Introduction 161
System Objectives and Design 163
Conceptual Model 165
Communicative Model 167
Relational Tables 167
Hierarchical Trees 167
Semantic Nets 170
Knowledge Frames 170
Action Frames 170
Production Rules 173
Natural Language Capability 177
Menu-Driven Capability 178
User Authorizations 182
Programmed Model 182
Relational Tables 185
Hierarchical Trees 185
Semantic Nets 185
Knowledge Frames 185
Action Frames 194
Production Rules 194
Natural Language Capability 194
Menu-Driven Facility 203
User Authorizations 203
Filling in Other Details 213
Application System Identification . . . 213
Activation of the Application System . 213
Help Messages 213
Experimental Model 218
Model Results 218

XII. INSTALLING A NEW EXPERT SYSTEM ON GUESS/1 . . . 223

Application Expert System Routines Distribution 224
Creating a GUESS/1 Interpreter With a New Domain-Specific Dictionary 226

XIII. DISCUSSION 228

A General Purpose Expert Systems Shell . . . 228
A Comparison with S.1 233

BIBLIOGRAPHY 240

A TOPICAL INDEX TO GUESS/1 LIBRARY ROUTINES 250

AN ALPHABETICAL INDEX TO GUESS/1 LIBRARY ROUTINES . . . 254

VITA 256

LIST OF FIGURES

Figure		page
1.	A Typical Expert System Architecture	3
2.	A Semantic Net	4
3.	A Production Rule	7
4.	A Comparison of Tools	12
5.	Sketch of a Hierarchical Frame	19
6.	AND/OR Goal Trees Implemented in Frames and Rules	22
7.	Representation of a Mechanism Tree	31
8.	Representation of a Knowledge Frame	33
9.	Representation of a Production Rule	34
10.	Representation of an Action Frame	36
11.	A Relational Table (1)	39
12.	A Relational Table (2)	41
13.	A Hierarchical Tree	43
14.	A Directed Graph	47
15.	Knowledge Frame Structure	51
16.	A Knowledge Frame	53
17.	Hierarchical Knowledge Frames	54
18.	A Semantic Net With Complex Links	59
19.	Action Frame Structure	62
20.	An Action Frame	64
21.	Hierarchical Action Frames	65
22.	Production Rule Structures	67

23. A Production Rule (1) 68
24. A Production Rule (2) 69
25. Intermix of Action Frames and Production Rules . . . 71
26. A Prioritized Agenda 73
27. Prolog Routines for Sorting an Agenda 74
28. Execution Trace 77
29. SSA_map Structure 83
30. A SSA_map 84
31. A Menu 85
32. Examples of Access Request Based on Clearance . . . 96
33. GUESS/1 Library Routines Distribution 160
34. The Life Cycle of Model Development 166
35. Nephrotoxicity of a Drug 168
36. Drug Classification Tree 169
37. A Causal Model 171
38. Drug Properties Knowledge Frame 172
39. An AND/OR Goal Tree to Determine Interactions . . 174
40. Action Frames for Predicting Drug Interactions . . 175
41. Neutralization Rule 176
42. Synonyms and Non-Keywords 179
43. Syntax-Semantic Maps 180
44. Display Menu and Corresponding Actions 181
45. A User Authorization File 183
46. A Login Procedure 184
47. Internal Representation of Relational Tables . . . 186

48. Implementation of Drug Nephrotoxicity Table . . . 187

49. Internal Representation of Hierarchical Trees . . 188

50. Drug Classification Tree 189

51. Internal Representation of Semantic Nets 190

52. A Causal Model 191

53. Internal Representation of Knowledge Frames . . . 192

54. A Drug Properties Frame 193

55. Internal Representation of Action Frames 195

56. An Action Frame to Determine Possible Interactions 196

57. An Action Frame to Determine Acid-Base Interaction 197

58. Internal Representations of Production Rules . . . 198

59. Neutralization Rule 199

60. Neutralization Rule (continued) 200

61. Internal Representations of Synonym and Non-Keyword Declarations 201

62. Synonym and Non-Keyword Declarations 202

63. Internal Representation of SSA_maps 204

64. SSA_maps 205

65. A Typical "No Match" SSA_map 206

66. Internal Representation of a Menu 207

67. A Display Menu 208

68. Internal Representation of Option/Action Rules . . 209

69. Some Option/Action Rules 210

70. Internal Representations of a User Authorization File 211

71. A User Authorization File 212

72. Expert System Identification 214

73. Code to Activate the Application System 215

74. Internal Representation of a HELP Rule 216

75. HELP Messages 217

76. Interactive Session on the Drug Interaction Expert 222

77. Application System Routines Distribution 225

78. Overall Structure of Expert Systems Shell 230

79. Anticipated Structure of Expert Systems Shell . . 231

80. The Ada Programming Environments 232

81. S1 Control Block vs GUESS/1 Action Frame 234

82. S.1 Class Attribute Definition vs GUESS/1 Knowledge Frame 236

83. S.1 Rule vs GUESS/1 Rule 238

THE PEOPLE WITH LIGHT COMING OUT OF THEM

by

William Saroyan

Look at the light shining out of those humble houses.
That light is the light of a happy nation,
a free and growing people,
a people without fear,
a people who love instead of hate,
whose casual everyday humanity
is stronger than any other power in the world...
The best in people from all over the world
is growing here into the first real nation of the world -
the American nation.

To the men and women of vision:

Ask not what the civilization of humankind can do for you,
ask what you can do for the civilization of humankind.

Chapter I

INTRODUCTION

The fear of the Lord is the
beginning of wisdom,
and knowledge of the Holy One
is understanding.

- Solomon, PROVERBS

Expert systems are very useful and are probably the most fruitful products of applied artificial intelligence (AI). There are a number of expert systems that perform diagnosis, data analysis, design, planning, consultation and many other functions that normally require human expertise. Expert systems, however, are very expensive to develop. DENDRAL [Feigenbaum, 1983], for example, took 40 man-years to complete. Powerful construction tools are therefore indispensable to construct, modify and maintain a practical expert system. GUESS/1 is a domain-independent expert systems shell that captures and enhances the strengths of its predecessors while at the same time overcoming their limitations. To cater for the readers who are not familiar with the basic principles of expert systems, we first introduce in brief the expert systems architecture and applications, then we will discuss the motivation for GUESS/1 and a general overview of the system.

1.1 EXPERT SYSTEMS ARCHITECTURE

What is an expert system? An expert system is an embodiment of human expert knowledge and thereby emulates human thought in problem solving in a specific domain. An expert system obtains a skill comparable to the traditional problem solver the "expert" or "consultant." The kernel or core of an expert system consists of a knowledge base (where facts and rules are stored) and an inference engine (where various methods of plausible reasoning are implemented). Other essential components include an explanation subsystem and a knowledge acquisition module. (See Figure 1).

A knowledge base differs from conventional databases in a number of ways. The main differences are data representations and database "creativity."

First, a knowledge base usually employs a comparatively higher level of data representations than a conventional database does. For instance, some pieces of data in a knowledge base are organized and stored as a semantic net. A semantic net is a directed graph whose nodes represent some objects and whose arcs denote relationships among the objects. Figure 2 shows a simple semantic net.

```
human expert
     |              ---------------            -------------
     | knowledge    | Knowledge     | facts and | Knowledge |
     ----------->   | Acquisition   |---------->| Base      |
     ----------->   | Module        |   rules   |           |
     |  data        ---------------            -------------
     |                                                 ^
 database                                            ||
                                                     ||
                                                     v
                    ----------------            -------------
          advice    | Explanation   |  results  | Inference |
   user <----------| Interface     |<----------| Engine    |
                    |               |           |           |
                    ----------------            -------------
```

Figure 1: A Typical Expert System Architecture

```
                 is_a
   ----------------------------------------
   |                                      |
   |                                      v
--------  is_a   -------  is_a   -------------------
| John |-------->| Boy |-------->| Living_Creature |
--------         -------         -------------------
   |
   | loves      --------  loves  --------------
   ------------>| Dogs |-------->| Dog's Food |
                --------         --------------
```

Figure 2: A Semantic Net

We can see from the graph that the relationship "is_a" is transitive. For example, "John is_a Boy and Boy is_a Living_Creature" implies "John is_a Living_Creature." However, the relation "loves" is not necessarily transitive. "John loves Dogs and Dogs loves Dog's Food" does not necessarily mean "John loves Dog's Food." Besides semantic nets, frames and scripts are among the often used knowledge representation structures. A frame is a table-like structure containing associated information about an object. A frame may include information about how to use the frame, about reasonable ranges of values of attributes, their default values and the consequences if they have unexpected values. A script is similar to a frame except that a script normally contains the objects and the sequences of actions related to a concept. A famous example is the restaurant script [Schank, 1977] which captures the normal expectations about a person going into a restaurant. Normally, the person is going to order something to eat and pay the bill. Unexpected events, should they occur, can be easily recognized. An abnormal event may be that "the person is going to bomb the restaurant." Why would a person ever want to bomb a restaurant? There must be reasons for it. Production rules are the favorite means of encapsulating cause-and-effect or condition/action type of knowledge. A

production rule has an IF-THEN format. Figure 3 shows a self-explanatory example of such a rule.

Second, a knowledge base is more "creative" than a conventional data base. A knowledge base actively tries to fill in the missing information either by deduction or by prompting the user for data. Frames, for instance, typically have slots for values to be determined dynamically. Scripts, encapsulating knowledge about normal expectation or behavior, "know" what pieces of information are missing from the input. For example, if "paying the bill" is not mentioned in a story about eating in a restaurant, the restaurant script will seek an answer to the question "did the customer pay the bill?" and may take the default "yes" if a definite answer is unobtainable.

To probe further into the field of expert systems, refer to [Hayes-Roth, 1983; Davis, 1982; Davis, 1977; Duda, 1983; Feigenbaum, 1977; Feigenbaum, 1983; Forsyth, 1983; Gevarter, 1983; Shortliffe, 1976; Stefik, 1982].

```
   IF a person bombs a restaurant, AND
      the person is not insane,
THEN it is very likely that the person is a terrorist.
```

Figure 3: A Production Rule

1.2 EXPERT SYSTEMS APPLICATIONS

The old dictum "if it's applied to real world problems, it's not AI" has finally been disposed of due to the success of a number of expert systems that perform diagnosis, data analysis, design, planning, consultation and many other functions that normally require human expertise. These applications of artificial intelligence have been emerging from the university laboratories out into the industry and government. For instance, XCON (originally named R1 [McDermott, 1982] and developed at Carnegie-Mellon University) is an operational expert system at the Digital Equipment Corporation that configures DEC VAX computer systems from skeletal specifications [Kinnucan, 1984]. An important function of XCON is to make sure that the computer equipment ordered by a customer will fit his/her VAX system, and that the desired purposes can be achieved by purchasing that particular piece of equipment. Results accumulated since 1981 indicate that the frequency of making errors in configuring VAX systems is "substantially" reduced with the use of XCON, and it was reported that the company has saved millions of dollars in labor costs. Many examples of other practical uses of expert systems and their statistical results can be found in the literature [eg. Shortliffe, 1976; Goldstein, 1977; Buchanan, 1978; Duda, 1979; Miller,

1982; Lenat, 1983; Hayes-Roth, 1984]. The government has not lagged behind the industry in applying expert systems technology to practical problems. There are already systems developed for naval task force threat analysis (TECH [Gevarter, 1983), mission planning (KNOBS [Engelman, 1983?]), battlefield weapons assignments (BATTLE [Slagle, 1983]), warfare simulation (SWIRL [Klahr, 1982]) and an unknown number of intelligence and defense applications.

1.3 MOTIVATION FOR GUESS/1

Expert systems can be "money-savers." As mentioned in the previous section, XCON reportedly has saved the Digital Equipment Corporation millions of dollars in labor costs. DENDRAL is another good example. DENDRAL [Buchanan, 1978; Feigenbaum, 1983] is an expert system that identifies molecular structures from a spectrum. The system requires a low-resolution spectrometer which costs $5,000 whereas a trained human requires a high-resolution spectrometer to do the same job which costs $50,000. In other words, DENDRAL plus $5,000 equals a trained human plus $50,000! Expert systems are indeed very promising in reducing labor and equipment costs. However, expert systems are expensive to build due to the lack of software support and knowledge

engineers. It took DENDRAL 40 man-years to complete! Our experience working with expert systems calls for having some powerful construction tools that would help speed up the lengthy process of development and reduce the maintenance cost of expert systems. There are tools available but they either impose too many restraints on a developer or they are not user-friendly [Kinnucan, 1984]. An expert systems workshop held in 1980 did a case study on eight different tools[1]: EMYCIN, KAS, EXPERT, OPS5, ROSIE, RLL, HEARSAY-III and AGE. Results of the investigation [Waterman, 1982] discussed their strengths and weaknesses in the following major categories:

1. human engineering
2. programming environment
3. language generality
4. diversity and flexibility of data representation and control structures
5. ability to track or explain the system's reasoning process

---

[1] In addition to these eight tools, there are a number of commercial products for constructing expert systems: OPS5E, KEE, DUCK, MRS, ART, S.1 and LOOPS, just to mention a few. However, most of these tools have not supported enough implementations to allow useful evaluations. OPS5E (an enhancement of OPS5) and MRS are probably some of the exceptions.

6. other features such as pattern matching capability and ability to handle multiple interacting knowledge sources

Each tool was found to be strong in some categories but weak in others (see Figure 4).

The domain-independent expert systems shell GUESS/1 is our attempt to capture and enhance the strengths of its predecessors while at the same time overcoming their limitations.

1.4 GENERAL OVERVIEW OF GUESS/1

GUESS/1 (General Purpose Expert Systems Shell) is a high-level programming language based on Prolog. Prolog is a general purpose AI language based on logic. GUESS/1 provides a sophisticated programming environment for building expert systems and it has a user-friendly interactive environment. Among its strengths are good human engineering, language generality, diverse data representation and control structures, strong pattern matching capability, simple backtracking mechanism, built-in debugging aid and database construction facility. Knowledge of facts is represented in relational tables, hierarchical trees, semantic nets and structured knowledge frames. The

Tool	Category					
	1	2	3	4	5	6
EMYCIN	S	S		W	S	
KAS		S	W	W	S	
EXPERT	S		W	W		
OPS5	W	W	S	S		S
ROSIE	S	W		W		S
RLL	W	W		S		
HEARSAY-III	W	W		S		S
AGE		W		S	W	

S = Strong, W = Weak, blank = Average

Figure 4: A Comparison of Tools

resulting knowledge base is highly modular and well organized. Control knowledge is encoded in action frames and production rules, which allow implementations of the goal-directed backward-chaining mechanism, the data-driven forward-chaining mechanism and a mixture of both. Action frames provide explicit control flows and help focus in solving a problem. Action frames can trigger production rules which may in turn invoke some other action frames. It is a generalization of CENTAUR's [Aikins, 1983] control structures. For a system built on GUESS/1, knowledge of facts and control knowledge are kept separated so that changing the logic does not affect the data, and vice versa. GUESS/1 also provides data base security facilities, simple interfaces to the conventional programming languages and to the underlying operating system. Moreover, additional features suitable for a problem domain can be implemented using the GUESS/1 primitives and the Prolog language. All in all, GUESS/1 simplifies the task of constructing and maintaining an expert system.

Two operational systems have been developed in GUESS/1. They are the Drug Interaction Expert System and the Pest and Orchard Management Expert. Both systems were rebuilt and improved from their first versions [Roach, 1984; Roach, 1985] which did not use any expert systems building tool at

all. Experience shows that the effort in building and maintaining an expert system is greatly eased by the use of GUESS/1.

Before you attempt to develop your own expert system on GUESS/1, please note the following:

Programming in GUESS/1 requires a basic knowledge of Prolog and some degree of familiarization with the Prolog system at Virginia Tech. Some reference materials you may need are [Clocksin, 1981] and [Roach, 1983].

1.5 A BRIEF READER'S GUIDE

The progression of the following chapters corresponds to the sequence:

- System Overview (Chapters II and III)
- System Components (Chapters IV, V, VI, VII, VIII and IX)
- Language Reference Manual (Chapter X)
- Application and Illustration (Chapter XI)
- System Installation (Chapter XII)
- Conclusion and Discussion (Chapter XIII)

Chapters II and III give a general overview of GUESS/1. The chapters describe what features are available in the system and their underlying rationale and objectives.

Chapters IV, V, VI, VII, VIII and IX describe how GUESS/1 supports expert systems development and maintenance in detail: Data Representation, Control Knowledge, Explanation Subsystem and Debugging Aids, Human Engineering, Interface to the External World and Database Security.

Chapter X is a language reference manual that describes in detail all the available routines in GUESS/1. The routines are grouped into 15 functional categories.

Chapter XI describes an application of GUESS/1 in medicine. The chapter covers the entire development process of a practical expert system from scratch: System Definition and Objectives, Conceptual Model, Communicative Model, Programmed Model, Experimental Model and Model Results.

Chapter XII contains instructions to install a new expert system and the GUESS/1 interpreter on a VAX 11/780 system.

Chapter XIII is the conclusion of GUESS/1 as an expert systems shell. A brief comparison between GUESS/1 and S.1 is included in the chapter because of the many similarities in both systems.

Chapter II

DESIGN PHILOSOPHY

Where there is no vision,
the people perish.

- Solomon, PROVERBS

The major goal of having an expert systems building tool is to assist the development, production and maintenance of expert systems. It is toward this aim that the design philosophy of GUESS/1 was formulated.

2.1 REPRESENTATION LANGUAGE

A high level representation language is much more desirable than a low level language to facilitate system development and maintenance. Programming in HEARSAY-III [Balzer, 1980] remains in general at a basic LISP level, whereas ROSIE [Fain, 1981; Hayes-Roth, 1981; Fain, 1982], at the other extreme uses an English-like syntax. GUESS/1, seeking a compromise between the functional LISP style and the English-like syntax in the design of an efficient high-level language, employs Prolog-like representations for knowledge and controls. In other words, it provides English-like

syntax Prolog macros suited for expert systems. For instance, to create a tree T with root R one would write (TREE_create TREE is T ROOT is R). Prolog [Kowalski, 1979; Chester, 1979; Roach, 1983] was chosen to be the implementation language due to its powerful built-in features, including a pattern-matching capability, automatic backtracking, high-level and flexible data representation, and its intrinsic modularity. Using the idea of unification, the strong pattern-matching power of Prolog contributed to GUESS/1 in many ways, especially in its natural language capability. The automatic backtracking feature in Prolog was useful but sometimes got in the way of implementing various additional control capabilities where backtracking was undesirable. The "cut" operator in Prolog was mostly used to get around this problem. Since GUESS/1 was implemented in Prolog, a system developer can mix GUESS/1 with Prolog within a program. This permits the developer to implement additional features that are not currently supported by GUESS/1.

2.2 DATA REPRESENTATION

Knowledge of facts is represented by relational tables, hierarchical trees, semantic networks, and knowledge frames. Tables, trees and networks are simple representation structures. Frames, on the other hand, are more complex. A knowledge frame [Minsky, 1975] is a table-like data structure containing associated information about an object or a concept. A frame may include information about how to use the frame; it may contain information about reasonable ranges of values of attributes, their default values and the consequences if they have unexpected values. Frames typically have slots for values to be determined dynamically from the database, from the user or from deduction. Instantiating an abstract frame structure creates an instance of a knowledge frame associated with an object. Instances of a frame are kept in a common knowledge blackboard [Lesser, 1977] for later use. Moreover, knowledge frames can be organized in a hierarchy or in a network. Such flexible and powerful data representation schemes facilitate the implementation of causal models. Figure 5 shows a typical knowledge frame in a hierarchical organization.

GUESS/1 provides a set of powerful utilities to construct, update and retrieve information from the database.

```
      A
     / \
    B   C  -----
   / \          |
  D   E         |
                v
-------------------------------------------------
| Author:                                       |
| Knowledge Frame Name:                         |
| Frame Usage:                                  |
| Level in Tree:                                |
|                                               |
| Slots  Default Values  Methods of Retrieval   |
| _____  _______________ ______________________ |
| _____  _______________ ______________________ |
| _____  _______________ ______________________ |
-------------------------------------------------
```

Figure 5: Sketch of a Hierarchical Frame

Knowledge tables, trees, networks and frames can be dynamically created and destroyed. An interactive transfer of knowledge from an expert is achieved by executing a set of pre-programmed knowledge frames which use the GUESS/1 utilities to construct and update the data bases accordingly. Another way to enter knowledge is simply by editing the data in the knowledge containers (table, trees, networks, frames) created on the disk files by the system developer. Additional data representation schemes can be implemented using the GUESS/1 primitives and optionally the Prolog language.

2.3 CONTROL KNOWLEDGE

Control knowledge in GUESS/1 is expressed in structured action frames and production rules. Production rules and frames have been advocated by many researchers [e.g. Van Melle, 1980; Pauker, 1977; Aikins, 1980] for building expert systems. The use of frames forces a problem solving strategy to be explicit and a search to be more focused. It also facilities explanations of the system's behavior. An action frame differs from a knowledge frame in that it represents a part of an active control flow of the system instead of a passive data encapsulation. In addition to

containing information about how to use the frame and about what knowledge frames to instantiate, an action frame may also include a success conditions slot that lists the conditions under which the frame is successfully applied. It may also contain a failure conditions slot that lists the conditions which force a backtrack to its caller. An example of a condition is (SAME ?X A) which means variable ?X has the value A. CENTAUR [Aikins, 1980; Aikins, 1983] uses prototypes or frames that trigger production rules as needed to solve a given problem. Production rules are treated as attached procedures. GUESS/1 extends this idea to allow an intermix of frames and production rules. In other words, action frames can trigger production rules as well as production rules invoking action frames. This relaxation allows more freedom in selecting a control strategy best suited to a problem domain. Different structuring of action frames and production rules, therefore, allows the programmer to represent different control strategies: procedural, recursive, backtracking, goal-directed backward-chaining, data-driven forward-chaining, or any mixture of these mechanisms. Figure 6 shows two and/or goal trees and a production rule.

The action frame C of the first goal tree points to a set of production rules in the middle of its execution. The

```
          A
         / \          [ Goal-directed
        /   \           Backward-chaining ]
       / AND \
      B       C  -----
     / \                |
    /OR \               |
   C     D              |
                        v
        ----------------------------------
        | Author:                        |
        | Action Frame Name:             |
        | Frame Usage:                   |
        | Level in Tree:                 |
        | Action Type (AND/OR):          |
        | Success Conditions:            |
        | Failure Conditions:            |
        |                                |
        | Action   Method of Achievement |
        | ______   ______________________ |
        | ______   ______________________----|---
        | ______   ______________________ |  |
        |                                |  |
        ----------------------------------  |
                                            |
    ----------------------------------------
    |
    |        [ Data-driven
    |          Forward-chaining ]
    |
    -----> IF   <condition 1> and
                <condition 2> and
                 ...
           THEN <action 1> and
                <action 2> and
                 ...    |
                        |
                        ---------> I
                                  / \
                                 /OR \
  [ Goal-directed               /     \
    Backward-chaining ]        J       K
                              / \      \
                             /OR \      \
                            /     \      \
                           L       M      N
```

Figure 6: AND/OR Goal Trees Implemented in Frames and Rules

production rules in the set are waiting for the provision of some particular input data that will trigger them at any moment. One of the rules is fired when the conditions are met by the input data items, and the rule invokes a new frame (I) in a new context (the second goal tree). The result is a combination of goal-driven backward-chaining and data-driven forward-chaining strategies. Furthermore, multiple knowledge sources are able to communicate and interact with each other through the use of a common knowledge blackboard. For reasons of efficiency and modularity, the knowledge blackboard is divided into segments storing different types of global data. For instance, an action frame blackboard segment contains findings, hypotheses or pseudo-interrupt signals generated by action frames only; and a knowledge frame blackboard segment stores only the instances of knowledge frames. Additional control structures can be implemented using the GUESS/1 primitives and optionally the Prolog language.

2.4 EXPLANATION SUBSYSTEM AND DEBUGGING AIDS

An expert system cannot be widely accepted without an ability to trace and explain its reasoning process. The action frames with production rules in GUESS/1 present a focused problem solving approach to a given problem. The reasoning process is automatically traced by the system. A "real-time" trace is displayed to the user when the system command TRACE is entered. The command NOTRACE turns the option off. A step-by-step bottom-up trace has not yet been implemented. Nevertheless, the idea is to relieve a system developer from the burden of implementing an "artificial" explanation subsystem. The built-in explanation subsystem follows automatically every line of reasoning of the system and displays the trace to the user upon request. The subsystem can be used as a debugging aid during the development of an expert system. Low level debugging is achieved by toggling the special flags in the system which would allow a rule-by-rule or statement-by-statement trace at the Prolog level.

2.5 HUMAN ENGINEERING

GUESS/1 has built-in natural language capability, menu driven facilities, a simple question-answering system, and utilities to alter system characteristics. Pattern-matching rules are employed for the recognition of natural language dialogue expressions [Colby, 1974]. Domain-specific words are translated into word class names, and simple patterns composed of word class names correspond to response functions. A built-in spelling checker and corrector detects misspelled words and attempts to correct them before any transformations are made to an input query. Menu driven facilities, however, bypass the transformations. The menu facilities and the question-answering system are often used together. The question-answering system prompts the user with pre-stored questions and expects simple responses (a word or a phrase) which may vary within some expected ranges. For example, the system builder may specify that any integer greater than 5 and less than 10 is an acceptable input. In addition, system characteristics such as the default system prompt and system messages can be altered or suppressed. System input/output is generally case-insensitive, that is, the user can type in mixed lower- and upper-case characters and all characters are automatically converted to their uppercase equivalence. For example,

'menu,' 'MeNu' and 'MENU' have the same meaning to the system. The only exception is when the user is prompted with a number of choices (eg. choose either 'a,' 'A' or 'aZ'). Additional human/system interfaces can be implemented using the GUESS/1 primitives and optionally the Prolog language.

2.6 INTERFACE TO THE EXTERNAL WORLD

Interfaces to other programming languages, database systems, and the underlying operating system are important parts of an expert systems shell. GUESS/1 provides simple interface mechanisms. For example, a system developer can write a number crunching Fortran program and link it to the system. Interface between GUESS/1 and the operating system allows interprocess communications, access to mail boxes and so on within the GUESS/1 environment. Currently, our GUESS system is interfaced with the VMS operating system on a VAX 11/780. A communication link to the conventional databases is essential for real world applications. Access from GUESS/1 to the FBI criminal records, for instance, would have to be efficient. Efficiency is achieved by linking the external DBMS subroutines to GUESS/1 at the source code level.

2.7 DATABASE SECURITY

Optional database security can be enforced for an expert system built on GUESS/1. Security protection becomes necessary when an expert system is used in national defense, military or financial purposes which often involve sensitive information. The two built-in basic protection mechanisms are the mandatory security protection and the discretionary (need-to-know) protection [DoD, 1983]. Each database object (a table, a tree, a network or a frame) in the system is labeled with a hierarchical classification level (for example: top secret, secret, confidential or unclassified) and a non-hierarchical category (for example: read, write or read and write). The system is able to handle any arbitrary number of classification levels and categories. Each user is assigned a clearance-tuple composed of a security level and a category, and a set of need-to-know object identifiers. Access to a data object is granted only if (1) the user clearance security level is greater than or equal to the data object classification security level, (2) the user category is a superset of the data object category, and (3) the user has the need-to-know to the data object. In order to reinforce the protection mechanisms, GUESS/1 provides login procedures, mechanisms to log transactions, and facilities to trace and monitor system activities.

Chapter III

REPRESENTATION LANGUAGE

To be is to be perceived.

- George Berkeley

A high level representation language is much more desirable than a low level language to facilitate system development and maintenance. Programming in HEARSAY-III [Balzer, 1980] remains in general at a basic LISP level, whereas ROSIE [Fain, 1981; Hayes-Roth, 1981; Fain, 1982], at the other extreme uses an English-like syntax. GUESS/1, seeking a compromise between the functional LISP style and the English-like syntax in the design of an efficient high-level language, employs Prolog-like representations for knowledge and controls. In other words, it provides English-like syntax Prolog macros suited for expert systems. For instance, to create a tree T with root R one would write (TREE_create TREE is T ROOT is R).

Prolog [Kowalski, 1979; Chester, 1979; Roach, 1983] was chosen to be the implementation language due to its powerful features, including built-in pattern-matching capability, automatic backtracking, high-level and flexible data

representation, and its intrinsic modularity. Using the idea of unification, the strong pattern-matching power of Prolog contributed to GUESS/1 in many ways, especially in its natural language capability. The automatic backtracking facility provided in Prolog was useful but sometimes got in the way of implementing various additional control capabilities where backtracking was undesirable. The "cut" operator in Prolog was mostly used to get around this problem. Since GUESS/1 was implemented in Prolog, a system developer can mix GUESS/1 with Prolog within a program. This permits the developer to implement additional features that are not currently supported by GUESS/1.

To give you a flavor of the GUESS/1 representation language, a few examples drawn from the domain of pharmacology and drug interactions are shown in the following. A system (the Drug Interaction Expert [Roach, 1984; Roach, 1985]) has been built using GUESS/1 to predict interactions when two drugs are administered together to a patient and to suggest corrective actions to alleviate detrimental interactions, if any.

The exact language syntax depends on the version of GUESS and Prolog you are using, and may therefore differs slightly from the examples given below. Please refer to the chapter on "GUESS/1 Library Routines" for the exact language syntax for your version of GUESS and Prolog.

3.1 EXAMPLE 1

Drug interaction mechanisms can be organized naturally in a hierarchical tree structure. The five classes of mechanisms are chemicophysical, pharmacodynamic, pharmacokinetic, physiological and "unknown". These are the second level nodes in the MECHANISM tree. Each class of mechanism can be subdivided into many different subtypes and each subtype can be further refined. Figure 7 shows the actual representation of a partial mechanism tree.

3.2 EXAMPLE 2

Acidity, pKa, water solubility, nephrotoxicity, et cetera constitute the concept of "drug properties". A knowledge frame is used to encapsulate these attributes of a drug. Figure 8 shows the actual representation of a partial "Drug Properties" frame. The "Drug Properties" frame, when

```
(TREE "Drug Interaction Mechanisms"
      (     (MECHANISM          Chemicophysical
                                Pharmacodynamic
                                Pharmacokinetic
                                Physiological
                                "Unknown"       )
            (Chemicophysical    Electrostatic
                                Chelation       )
            (Electrostatic      In-Vitro
                                In-Vivo         )
             ....
       )
)
```

Figure 7: Representation of a Mechanism Tree

instantiated, fills in the slots by searching the databases, by asking the user or by taking the default values. Then it writes an instance of itself to the common knowledge blackboard to be kept for later use.

3.3 EXAMPLE 3

If a clinician asks if there is an electrostatic interaction between two drugs and how much neutralization will result, then instantiate knowledge frame "Drug Properties" and trigger action frame "Electrostatic Interaction" to determine such interaction and the amount of neutralization. Figure 9 shows the actual representation of a production rule to accomplish the request.

3.4 EXAMPLE 4

If one drug is acidic and another drug is basic, then a mixture of these two drugs will cause an electrostatic reaction. The result is that one drug will dominate and the other drug will lose all its effectiveness. In some rare circumstances, however, both drug will neutralize each other completely. The degree of neutralization can be computed using the drug's pKa's and the patient's body pH. An action

```
(Knowledge Frame "Drug Properties")
->
(Frame Usage is "Drug Properties Frame")
(Get Current Object of Interest  ?drug)
(or (Lookup Table
            Table is Acidity
            Key   is ?drug
            Value is ?ans)
    (Ask User
         Question is "What is the acidity of"
         Object   is ?drug
         Expect   (acid base neutral unknown)
         Answer   is ?ans)
    (Default of  ?ans  is  Unknown)
)
 ...
 ...
(Update Blackboard
        Knowledge Frame is "Drug Properties"
        Object          is ?drug
        Content         is (Acidity ?ans
                            pKa     ?pka
                             ...)
)
```

Figure 8: Representation of a Knowledge Frame

```
[ IF clause ]
(User_Ask_For Electrostatic Interaction
 of ?drug1 ?drug2)
->
[ THEN clauses ]
(Cleanup Blackboard)
(Add Object of Interest is ?drug1)
(Add Object of Interest is ?drug2)
(Instantiate Knowledge Frame "Drug Properties"
             Object is ?drug1)
(Instantiate Knowledge Frame "Drug Properties"
             Object is ?drug2)
(Activate Action Frame
          "Electrostatic Interaction")
```

Figure 9: Representation of a Production Rule

frame (Figure 10) is created to determine the electrostatic interaction. The action frame triggers a rule to compute the amount of neutralization and it records the findings on the common knowledge blackboard.

```
(Action Frame "Electrostatic Interaction")
->
(Frame Usage is
       "Determine Electrostatic Interaction")
(Get Object of Interest ?drug1)
(Get Object of Interest ?drug2)
(Lookup Blackboard
        Knowledge Frame is "Drug Properties"
        Object          is ?drug1
        Key             is Acidity
        Value           is ?ans1)
(Lookup Blackboard
        Knowledge Frame is "Drug Properties"
        Object          is ?drug2
        Key             is Acidity
        Value           is ?ans2)
(Same Set (acid base) (?ans1 ?ans2))
(Lookup Blackboard
        Knowledge Frame is "Drug Properties"
        Object          is ?drug1
        Key             is pKa
        Value           is ?p1)
(Lookup Blackboard
        Knowledge Frame is "Drug Properties"
        Object          is ?drug2
        Key             is pKa
        Value           is ?p2)
(Compute Amount of Neutralization ?n
        from ?p1 ?p2)
(Update Blackboard
        Action Frame is
               "Electrostatic Interaction"
        Content      is
               (Drug1          ?drug1
                Drug2          ?drug2
                Interaction    Electrostatic
                Neutralization ?n)
)
```

Figure 10: Representation of an Action Frame

Chapter IV

DATA REPRESENTATIONS

To exist is to stand out,
away from the background.

- Frank Herbert, CHILDREN OF DUNE

Knowledge of facts is represented by relational tables, hierarchical trees, semantic nets, and knowledge frames. Tables, trees and networks are simple representation structures. Frames, on the other hand, are little more complex. GUESS/1 provides a set of powerful utilities to construct, update and retrieve information from the database. Tables, trees, networks and frames can be dynamically created and destroyed. An interactive transfer of expertise is achieved by executing a set of pre-programmed knowledge frames which use the GUESS/1 utilities to construct and update the databases accordingly. Another way to enter knowledge into the system is simply by editing the data in the knowledge containers (table, trees, networks, frames) created on the disk files by the system developer. Additional data representation schemes can be implemented using the GUESS/1 primitives and optionally the Prolog language.

The GUESS/1 database is comprised of five basic types of objects:

- Relational Tables: storing tabulated information.
- Hierarchical Trees: storing hierarchical information.
- Semantic Nets: storing non-hierarchical relational information.
- Knowledge Frames: storing high level meta-knowledge.
- Blackboard: storing information for data communication purposes.

The following sections are devoted to the discussion of each data representation.

4.1 RELATIONAL TABLES

A relational table is a set of key/value-tuples grouped together under a defined relation. Each key/value-tuple is composed of a key and a value associated with the key. The only rule to adhere to is that every key in a table MUST BE UNIQUE to that particular table.

Figure 11 shows an example of a relational table.

Homogeneity of data types is not required; in other words, two seemingly different types of data can appear in the same table. In Figure 11, the value (GPA) column is all numeric.

Key: Student Name	Value: Grade Point Average
John	4.0
Tom	3.0
David	3.3
Mary	3.8
Ann	4.0

Figure 11: A Relational Table (1)

We can, for example, enter a person name Jennifer and its associated grade point average being "unknown, suspect to be higher than 3.0." Moreover, a value can be a list of items. Figure 12 shows a difference Student/GPA table for the Fall, Winter and Spring Quarters of the academic year.

GUESS/1 provides built-in library routines to create, manipulate and destroy relational tables. The following are the available routines. Read the chapter on "GUESS/1 Library Routines" for details about their usage. To save the newly-created or recently-updated relational tables permanently, use the library routine (SAVE KNOWLEDGE_BASE).

To look up the value associated with a given key in a relational table.

```
(TABLE_LOOKUP
        TABLE   IS   <table_name>
        KEY     IS   <key_name>
        VALUE   IS   <value>)
```

To add a new item to a relational table.

```
(TABLE_ADD
        TABLE IS  <table_name>
        KEY   IS  <key_name>
        VALUE IS  <value>)
```

Key: Student Name	Value: Grade Point Average
John	(4.0, 4.0, 4.0)
Tom	(3.9, 3.5, 3.0)
David	(3.0, 3.1, 3.3)
Mary	(4.0, 3.8, 3.8)
Ann	(4.0, 4.0, 4.0)

Figure 12: A Relational Table (2)

```
To create a new relational table.

(TABLE_CREATE
       TABLE IS  <table_name>)

(TABLE_CREATE
       TABLE    IS  <table_name>
       SECURITY IS  <security>
       CATEGORY IS  <category>)
```

```
To destroy a relational table.

(TABLE_DESTROY
       TABLE IS  <table_name>)
```

```
To delete an old item from a relational table.

(TABLE_ERASE
       TABLE IS  <table_name>
       KEY   IS  <key>
       VALUE IS  <value>)
```

4.2 HIERARCHICAL TREES

A tree reflects the natural hierarchy of certain types of knowledge. A tree is n-ary and uni-directional. Each node in a tree contains a piece of datum (eg. a concept or an object). Figure 13 shows an example of a hierarchical tree.

```
                    Living Organisms
                           |
                    ----------------
                    |               |
                 Animals          Plants
                    |
            -----------------
            |                |
     Higher Animals   Lower Animals
```

Figure 13: A Hierarchical Tree

GUESS/1 provides built-in library routines to create, manipulate and destroy hierarchical trees. The following are the available routines. Read the chapter on "GUESS/1 Library Routines" for details about their usage. To save the newly-created or recently-updated hierarchical trees permanently, use the library routine (SAVE KNOWLEDGE_BASE).

To look up the parent node given one or all of its successor nodes, or to look up the children nodes given their predecessor node.

```
(TREE_LOOKUP
        TREENAME  IS  <tree_name>
        PARENT    IS  <parent_id>
        CHILDREN  IS  <children_id>)
```

To traverse a tree and collect all the nodes starting from a given node.

```
(TREE_EXPAND
        TREENAME  IS  <tree_name>
        NODE      IS  <starting_node>
        TREE      IS  <expanded_tree>)
```

To add a new item to a tree.

```
(TREE_ADD
      TREENAME IS  <tree_name>
      BRANCH   IS  <parent> <newchild> [ <treeid> ] )
```

To add to a tree a new top branch connecting a new root to the old root.

```
(TREE_UPDATE
      TREENAME IS  <tree_name>
      OLDROOT  IS  <old_root>
      NEWROOT  IS  <new_root>)
```

To create a new tree.

```
(TREE_CREATE
      TREENAME IS  <tree_name>
      ROOT     IS  <root>)

(TREE_CREATE
      TREENAME IS  <tree_name>
      ROOT     IS  <root>
      SECURITY IS  <security>
      CATEGORY IS  <category>)
```

To destroy a tree.

```
(TREE_DESTROY
      TREENAME IS  <tree_name>
```

To delete an old item from a tree.

```
(TREE_ERASE
      TREENAME IS  <tree_name>
      PARENT   IS  <parent>
      CHILD    IS  <child>)
```

4.3 SEMANTIC NETS

Semantic nets come with two flavors: directed and undirected graphs. The nodes in a graph denote some objects or concepts and the arcs in the graph represent some relationships among the objects or concepts. Every relationship is uni-directional in a directed graph and it is bi-directional in an undirected graph. Figure 14 shows an example of a directed graph.

GUESS/1 provides built-in library routines to create, manipulate and destroy semantic nets. The following are the available routines. UNET refers to undirected graphs whereas DNET refers to directed graphs. Read the chapter on "GUESS/1 Library Routines" for details about their usage. To save the newly-created or recently-updated semantic nets permanently, use the library routine (SAVE KNOWLEDGE_BASE).

To find all the neighbors of a given node in an undirected graph.

```
(UNET_RETRIEVE
                NETNAME  IS  <name>
                NODE     IS  <node>
                NEIGHBOR IS  <neighbor>)
```

```
                    is_a
     ------------------------------------------
     |                                        |
     |                                        v
--------    is_a    -------    is_a    -------------------
| John |---------> | Boy |---------->| Living_Creature |
--------            -------           -------------------
     |
     |   loves      --------  loves  --------------
     --------------->| Dogs |-------->| Dog's Food |
                     --------          --------------
```

Figure 14: A Directed Graph

To find all the nodes pointed to in a certain relationship by the given node in a directed graph. If the relation is not given as an input, it will find all the nodes pointed to by the given node and their relationships to the given node.

```
(DNET_RETRIEVE
              NETNAME  IS  <name>
              NODE     IS  <node>
              RELATION IS  <relation>
              NEIGHBOR IS  <neighbor>)
```

To find the relationship between two given nodes in a directed graph. If there is no relation between them, it returns NIL.

```
(DNET_RETRIEVE
              NETNAME   IS  <name>
              NODE1     IS  <node1>
              NODE2     IS  <node2>
              RELATION  IS  <rel>
              DIRECTION IS  <dir>)
```

To add a new node and a new arc to the graph.

```
(DNET_ADD
      NETNAME  IS  <name>
      ARC      IS  <node1> <relation> <node2>)

(UNET_ADD
      NETNAME  IS  <name>
      ARC      IS  <node1> <node2>)
```

To create a new graph.

```
(DNET_CREATE
      NETNAME  IS  <name>)

(UNET_CREATE
      NETNAME  IS  <name>)

(DNET_CREATE
      NETNAME  IS  <name>
      SECURITY IS  <security>
      CATEGORY IS  <category>)

(UNET_CREATE
      NETNAME  IS  <name>
      SECURITY IS  <security>
      CATEGORY IS  <category>)
```

To destroy an old graph.

```
(DNET_DESTROY
      NETNAME  IS  <name>)

(UNET_DESTROY
      NETNAME  IS  <name>)
```

To delete an old item from the graph.

```
(DNET_ERASE
      NETNAME  IS  <name>
      ARC      IS  <node1> <relation> <node2>)

(UNET_ERASE
      NETNAME  IS  <name>
      ARC      IS  <node1> <node2>)
```

4.4 KNOWLEDGE FRAMES

A knowledge frame [Minsky, 1975] is a table-like data structure containing associated information about an object or a concept. A frame may include information about how to use the frame; it may contain information about reasonable ranges of values of attributes, their default values and the consequences if they have unexpected values. Frames typically have slots for values to be determined dynamically from the database, from the user or from deduction. Instantiating an abstract frame structure creates an instance of a knowledge frame associated with an object. Instances of a frame are kept in a common knowledge blackboard for later use. Moreover, knowledge frames can be organized in a hierarchy or in a network. Such flexible and powerful data representation schemes facilitate the implementations of causal models and default reasoning. Figure 15 shows the general structure of a knowledge frame.

The KF NAME slot contains the identification of the frame; the USAGE slot describes what the frame is about; the LEVEL# slot is the level in a tree-like structure when the knowledge frames are organized hierarchically. The LEVEL# is particularly useful for explanation and debugging purposes.

KF NAME: _____
USAGE: _____
LEVEL#: _____

AUTHOR: _____
DATE: _____

SLOTS	DEFAULT VALUES	METHODS OF RETRIEVAL
_____	_____	_____
_____	_____	_____
_____	_____	_____

Figure 15: Knowledge Frame Structure

The METHODS OF RETRIEVAL entries refer to the proper methods that are to be employed to fill in the slots. The basic methods are:

1. look up a relational table
2. look up a hierarchical tree
3. look up a semantic net
4. look up the blackboard
5. ask the user
6. take on the default values stored in the DEFAULT VALUES slot
7. invoke some other knowledge frames

There must exist one or more methods for each METHODS OF RETRIEVAL slot. The methods are attempted sequentially in the specified order until one of them successfully finds a value to fill in the slots. It is recommended that the last resort is always to take on the default values in case no other method works.

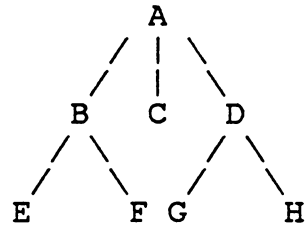
Figure 16 illustrates the use of a knowledge frame. Following in Figure 17 shows the organization of knowledge frames in a hierarchical manner.

KF NAME: a person
USAGE: to obtain the relevant information about a person
LEVEL#: Any

AUTHOR: Newton S. Lee
DATE: January 1985

SLOTS	DEFAULT VALUES	METHODS OF RETRIEVAL
Name	X	1. Ask 2. Take default value
Age	Unknown	1. Look up table 2. Ask 3. Take default value
Sex	Male or Female	1. Look up table 2. Ask 3. Take default value

Figure 16: A Knowledge Frame



KF NAME: A
USAGE: Illustration
LEVEL#: Any

AUTHOR: Newton S. Lee
DATE: January 1985

SLOTS	DEFAULT VALUES	METHODS OF RETRIEVAL
B	none	invoke knowledge frame B
C	none	look up semantic net
D	none	invoke knowledge frame D

Figure 17: Hierarchical Knowledge Frames

4.5 BLACKBOARD

Communication and sharing of data among the knowledge frames, production rules and action frames[2] are accomplished by dynamic update and lookup from a global accessible area known as the blackboard [Lesser, 1977]. There is only one global blackboard and it is fragmented into three segments:

- knowledge frame segment
- action frame segment
- object list segment

The following sections describe the usage for each of the three segments and the built-in library routines available to manipulate these segments.

4.5.1 Knowledge Frame Segment

The knowledge frame segment is reserved for knowledge frames to write information onto the blackboard. The written information can be retrieved by some other knowledge frames, action frames or production rules. The following are the GUESS/1 built-in library routines to perform update and lookup operations on this blackboard segment. For more details, read the chapter on "GUESS/1 Library Routines."

---

[2] Production rules and action frames are described in the chapter "Control Knowledge."

```
To update or look up information from the knowledge
frame segment.

  (BLACKBOARD_UPDATE
          KF      IS  <KF_name>
          OBJECT  IS  <object_id>
          CONTENT IS  <contents>)

  (BLACKBOARD_LOOKUP
          KF      IS  <KF_name>
          OBJECT  IS  <object_id>
          KEY     IS  <key>
          VALUE   IS  <value>)
```

4.5.2 Action Frame Segment

The action frame segment is reserved for action frames to write information onto the blackboard. The written information can be retrieved by some other action frames, knowledge frames or production rules. The following are the GUESS/1 library routines to perform update and lookup operations on this blackboard segment.

```
To update or look up information from the action frame
segment.

  (BLACKBOARD_UPDATE
          AF      IS  <AF_name>
          OBJECT  IS  <object_id>
          CONTENT IS  <contents>)

  (BLACKBOARD_LOOKUP
          AF      IS  <AF_name>
          OBJECT  IS  <object_id>
          KEY     IS  <key>
          VALUE   IS  <value>)
```

4.5.3 Object List Segment

The object list segment stores the objects of interest which are being referenced by the knowledge frames, action frames and production rules. The following are the GUESS/1 library routines to perform update and lookup operations on this blackboard segment.

To update and look up objects of interest from the object list segment.

```
(ADD         OBJECT_OF_INTEREST   <object>)
(GETALL      OBJECT_OF_INTEREST   <object>)
(GETFIRST    OBJECT_OF_INTEREST   <object>)
(GETNEXT     OBJECT_OF_INTEREST   <object>)
(GETCURRENT  OBJECT_OF_INTEREST   <object>)
```

4.6 MORE COMPLEX REPRESENTATIONS

More complex knowledge representation schemes can be implemented using the GUESS/1 primitives and optionally the Prolog language. Suppose we want to build a semantic net in which the links can be expanded into property frames, the general structure will look like Figure 18 . The programmer can manipulate the structure using the standard knowledge frame and semantic network manipulation routines provided in GUESS/1.

```
      |                 |
      v                 v
    -----  link_a_b   -----
<---| a |------------>| b |----->
    -----       .     -----
                .
                .
                .
                .
                .
               -------------------------
               | KF NAME: link_a_b       |
               | USAGE: property frame   |
               | LEVEL: 0                |
               | SLOTS: _________________ |
               -------------------------
```

Figure 18: A Semantic Net With Complex Links

Chapter V

CONTROL KNOWLEDGE

It is not good to have zeal
  without knowledge,
nor to be hasty and miss the way.

- Solomon, PROVERBS

Control knowledge in GUESS/1 is expressed in structured action frames and production rules. Production rules and frames have been advocated by many researchers [e.g. Van Melle, 1980; Pauker, 1977; Aikins, 1980] for building expert systems. The use of frames forces a problem solving strategy to be explicit and a search to be more focused. It also facilities explanations of the system's behavior. The combined use of action frames and production rules allows the programmer to represent various control strategies: procedural, recursive, backtracking, goal-directed backward-chaining, data-driven forward-chaining, or any combination of these mechanisms. Additional control structures can be implemented using the GUESS/1 primitives and optionally the Prolog language.

5.1 ACTION FRAMES

An action frame differs from a knowledge frame in that it represents a part of an active control flow of the system instead of a passive data encapsulation. In addition to containing information about how to use the frame and about what knowledge frames to instantiate, an action frame may also include a success conditions slot that lists the conditions under which the frame is successfully applied. It may also contain a failure conditions slot that lists the conditions which force a backtrack up to its caller. An example of a condition is (SAME ?X A) which means variable ?X has the value A. Figure 19 shows the structure of an action frame.

The AF NAME slot contains the identification of the action frame; the USAGE slot describes the use of this frame; the LEVEL# slot is the level in a tree-like structure when the action frames are organized hierarchically. The LEVEL# slot is particularly useful for explanation and debugging purposes. The ACTION TYPE is either AND or OR in a hierarchical AND/OR goal tree. The conditions under which an action frame will succeed in pursuing a goal and will fail in doing so are listed in the SUCCESS COND slot and the FAILURE COND slot respectively.

AF NAME: _____
USAGE: _____
LEVEL#: _____
ACTION TYPE: _____
SUCCESS COND: _____
FAILURE COND: _____

AUTHOR: _____
DATE: _____

ACTIONS	METHODS OF ACHIEVEMENT
_____	_____
_____	_____
_____	_____

Figure 19: Action Frame Structure

The ACTIONS slots list a sequence of actions or goals needed to be performed when the action frame is invoked. The METHODS OF ACHIEVEMENT entries tell how each and every action is going to be carried out. The basic methods to achieve a goal are:

- invoke other action frames
- invoke some production rules
- instantiate some knowledge frames
- look up the blackboard
- invoke GUESS/1 built-in library routines

Figure 20 illustrates the use of action frames.

Employing the philosophy of "divide and conquer," an AND/OR goal tree is a very powerful way to solve a problem at hand. Action frames can be organized in a hierarchical fashion. Figure 21 illustrates the use of action frames to implement an AND/OR goal tree.

In short, the use of action frames forces a problem solving strategy to be explicit and a search to be more focused. It also facilitates explanations of the system's behavior.

AF NAME: Acid-base reaction
USAGE: Determine acid-base reaction
LEVEL#: 2
ACTION TYPE: AND
SUCCESS COND: There is an acid-base reaction
FAILURE COND: There is no acid-base reaction

AUTHOR: Newton S. Lee
DATE: January 1985

ACTIONS	METHODS
1. Get the names of the two chemicals	Ask the user
2. Make sure one is acidic, one is basic	Look up "Chemical Properties" tables and retrieve their pKa's and pH's
3. Determine amount of neutralization	Invoke neutralization rule

Figure 20: An Action Frame

```
                GOAL TREE                          LEVEL

          Establish world peace
          /          |          \                    1
         / <------ AND ------> \
        /           |           \
  Engage in     Deter         Discourage
  Arms talks    terrorism     wars                   2
                                /        \
                               /<- AND ->\
                              /          \
                        Economic       Peaceful
                        sanction       protest       3
```

AF NAME: world peace
USAGE: To establish world peace
LEVEL#: 1
ACTION TYPE: AND
SUCCESS COND: Most nations involve in the actions to establish world peace
FAILURE COND: Most nations do not involve in the actions to establish world peace

AUTHOR: Newton S. Lee
DATE: January 1985

ACTIONS	METHODS
1. Engage in arms talks	Invoke action frame "Arms Talks"
2. Deter terrorism	Invoke action frame "Counter-terrorism"
3. Discourage wars	Invoke action frame "Actions Against Wars"

Figure 21: Hierarchical Action Frames

5.2 PRODUCTION RULES

Production rules are often used to encapsulate cause-and-effect or condition/action type of knowledge. A production rule has an IF-THEN format. Similar to an action frame, a production rule can:

- invoke action frames
- invoke other production rules
- instantiate some knowledge frames
- look up the blackboard
- invoke GUESS/1 built-in library routines
- invoke standard Prolog functions

The two general structures of a production rule are shown in Figure 22 Following it in Figure 23 and 24 are two simple examples. Please note that comments are preceded by a semicolon (;).

5.3 INTERMIX OF ACTION FRAMES AND PRODUCTION RULES

CENTAUR [Aikins, 1980; Aikins, 1983] uses prototypes or frames that trigger production rules as needed to solve a given problem. Production rules are treated as attached procedures. GUESS/1 extends this idea to allow an intermix of frames and production rules. In other words, action frames can trigger production rules as well as production

```
; IF clause                ( <identification> <parameters> )
( <condition> )            ->
->                         ; IF clauses
; THEN clauses             ( <conditions> )
( <actions> )              ; THEN clauses
                           ( <actions> )
```

Figure 22: Production Rule Structures

```
; IF clause
(User_Ask_For Acid-Base Reaction of ?Chemical1
              and ?Chemical2)
->
; THEN clauses
(Add_Object_of_Interest is  ?Chemical1)
(Add_Object_of_Interest is  ?Chemical2)
(Invoke Action Frame "Determine Acid-Base Reaction")
(Generate Answer)
```

Figure 23: A Production Rule (1)

```
(Determine Rationale)
->
[ IF clauses ]
(Person X bombs an embassy)
(Person X is not insane)
[ THEN clauses ]
(Conclude Person X is a terrorist)
```

Figure 24: A Production Rule (2)

rules invoking action frames. This relaxation allows more freedom in selecting a control strategy best suited to a problem domain. Different structuring of action frames and production rules, therefore, allows the programmer to represent different control strategies: procedural, recursive, backtracking, goal-directed backward-chaining, data-driven forward-chaining, or any mixture of these mechanisms. Figure 25 shows two and/or goal trees and a production rule. The action frame C of the first goal tree points to a set of production rules in the middle of its execution. The production rules in the set are waiting for the provision of some particular input data which will trigger them at any moment. One of the rules is fired when the conditions are met by the input data items, and the rule invokes a new frame (I) in a new context (the second goal tree). The result is a combination of goal-driven backward-chaining and data-driven forward-chaining strategies.

5.4 MULTIPLE KNOWLEDGE SOURCES

GUESS/1 supports the implementation of multiple knowledge sources by providing a global blackboard as a mean of data communication among the interacting knowledge sources. The blackboard has been described in detail in the chapter on

```
        A
       / \          [ Goal-directed
      /   \           Backward-chaining ]
     / AND \
    B       C  -----
   / \              |
  /OR \             |
 C     D            |
                    v
      ----------------------------------
      | Author:                        |
      | Action Frame Name:             |
      | Frame Usage:                   |
      | Level in Tree:                 |
      | Action Type (AND/OR):          |
      | Success Conditions:            |
      | Failure Conditions:            |
      |                                |
      | Action   Method of Achievement |
      | ______   ______________________ |
      | ______   ______________________----|---
      | ______   ______________________ |  |
      |                                |  |
      ----------------------------------  |
                                          |
   ----------------------------------------
   |
   |        [ Data-driven
   |          Forward-chaining ]
   |
   ----->  IF   <condition 1> and
                <condition 2> and
                 ...
           THEN <action 1> and
                <action 2> and
                 ...    |
                        |
                        ---------> I
                                  / \
                                 /OR \
                                /     \
                               J       K
                              / \       \
                             /OR \       \
                            /     \       \
                           L       M       N
[ Goal-directed
  Backward-chaining ]
```

Figure 25: Intermix of Action Frames and Production Rules

"Data Representation." An action frame, for example, can write findings, hypotheses or pseudo-interrupt signals onto the blackboard. Some other frames will interpret the information on the blackboard and react accordingly. Should conflicting information be posted on the blackboard, a "judgement" frame or rule will eventually make a final decision or suggest some conflict resolutions.

5.5 MORE COMPLEX CONTROL STRUCTURES

More complex control structures can be implemented using the GUESS/1 primitives and optionally the Prolog language. Suppose we want to build a prioritized agenda system, we can use the built-in common knowledge blackboard to keep a list of all the names of the action frames with their associated priorities. A few Prolog rules may be necessary to manipulate and maintain the list. Figure 26 shows some routines to manipulate the GUESS/1 blackboard object list segment for an agenda system. Following it in Figure 27 are the Prolog sorting routines based on the Quicksort algorithm.

To initialize the agenda:

```
( (INITIALIZE AGENDA)
  ->
  (OBJECTLIST_ERASE) )
```

To insert an item into the agenda:

```
( (ADD *af_name *priority TO AGENDA)
  ->
  (ADD OBJECT_OF_INTEREST (*af_name *priority))
  (SORT AGENDA) )
```

To retrieve the top prioritized item from the agenda:

```
( (RETRIEVE *af_name *priority FROM AGENDA)
  ->
  (GETFIRST OBJECT_OF_INTEREST *priority)
  (GETNEXT OBJECT_OF_INTEREST *af_name) )
```

To sort the agenda by priorities:

```
( (SORT AGENDA)
  ->
  (GETALL OBJECT_OF_INTEREST *agenda)
  (QSORT *agenda nil *new_agenda)
  (OBJECTLIST_ERASE)
  (ADD OBJECT_OF_INTEREST *new_agenda) )
```

Figure 26: A Prioritized Agenda

```
;
; Quick Sort Algorithm:
;

( (QSORT (*priority *af . *l) *r0 *r)
  ->
  (PARTITION *l *priority *af *l1 *l2)
  (QSORT *l2 *r0 *r1)
  (QSORT *l1 (*af *priority . *r1) *r) )

( (QSORT nil *x *x) )


( (PARTITION (*priority *af . *l)
    *priority_2 *af_2
    (*priority *af . *l1) *l2)
  ->
  (<= *priority *priority_2)
  (cut)
  (PARTITION *l *priority_2 *af_2 *l1 *l2) )

( (PARTITION (*priority *af . *l)
    *priority_2 *af_2 *l1
    (*priority *af . *l2))
  ->
  (> *priority *priority_2)
  (cut)
  (PARTITION *l *priority_2 *af_2 *l1 *l2) )

( (PARTITION nil * * nil nil) )
```

Figure 27: Prolog Routines for Sorting an Agenda

Chapter VI

EXPLANATION SUBSYSTEM AND DEBUGGING AIDS

A life that is not examined is
not worth living.

- Socrates

An expert system cannot be widely accepted without an ability to trace and explain its reasoning process. The action frames and production rules in GUESS/1 present a focused problem solving approach to a given problem. The reasoning process is automatically traced by the system. A "real-time" trace is displayed to the user when the system command TRACE is entered. The command NOTRACE turns the option off. The built-in explanation subsystem automatically follows every line of reasoning of the system and displays the trace to the user upon request. The subsystem can be used as a debugging aid during the development of an expert system. Low level debugging is achieved by toggling some special flags in the system that allow a rule-by-rule or statement-by-statement trace at the Prolog level.

6.1 EXPLANATION SUBSYSTEM

In order for the system to trace its reasoning process automatically, each knowledge frame, action frame and production rule should always invoke the GUESS/1 library routine USAGE. The syntax of USAGE is (USAGE <description> LEVEL <level number>). The description is a string of characters identifying the frame or rule and the use of such a frame or rule. The level number refers to the level of depth in an AND/OR goal tree. When the user activates the trace, the descriptions and level numbers are displayed in "real-time" while the frames and rules are being executed. Figure 28 is a trace of the "execution" of the AND/OR goal tree shown in Figure 21

Optionally a frame or rule can invoke GUESS/1 built-in library routine SYSMSG. The syntax is (SYSMSG <message>). The message is a string of text to be displayed to the user when the system option MSG is activated.

```
[1] Establish world peace
[2]     Engage in arms talks
[2]     Deter terrorism
[2]     Discourage wars
[3]         Economic sanction
[3]         Peaceful protest
```

Figure 28: Execution Trace

6.2 DEBUGGING AIDS

In addition to the GUESS/1 built-in library routines USAGE and SYSMSG, the programmer can debug the program in a relatively low level by toggling some special flags in the system. They include (set sflag t) for selective tracing and (set tflag t) for complete detailed trace. The debugging facilities are fully documented in the Virginia Tech Prolog manual [Roach, 1983].

Chapter VII

HUMAN ENGINEERING

> The human question is not how many can
> possibly survive with the system,
> but what kind of existence is possible
> for those who do survive.
>
> - Frank Herbert, DUNE

GUESS/1 has a built-in natural language capability, a menu driven facility, a simple question-answering system, and numerous utilities to alter system characteristics. The goals are to make the system user-friendly, easy to use and easy to customize. Pattern-matching rules are employed for the recognition of natural language dialogue expressions [Colby, 1974]. Domain-specific words are translated into word class names, and simple patterns composed of word class names correspond to response functions. A built-in spelling checker and corrector[3] detects misspelled words and attempts to correct them before any transformations are made to an input query. Menu driven facilities, however, bypass the natural language parser and spelling correction. The menu facilities and the question-answering system are often used together. The question-answering system prompts the user

[3] The author thanks Roger Ehrich for providing the kernel of the spelling corrector.

with pre-stored questions and expects simple responses (a word or a phrase) which may vary within some expected ranges. For example, the system builder may specify that any integer greater than 5 and less than 10 is an acceptable input. In addition, system characteristics such as the default system prompt and system messages can be altered or suppressed. System input/output is generally case-insensitive, that is, the user can type in mixed lower- and upper-case characters and all characters are automatically converted to their uppercase equivalence. For example, 'menu,' 'MeNu' and 'MENU' have the same meaning to the system. The only exception is when the user is prompted with a number of choices (eg. choose either 'a,' 'A' or 'aZ'). Additional human/system interfaces can be implemented using the GUESS/1 primitives and optionally the Prolog language.

7.1 NATURAL LANGUAGE CAPABILITY

The natural language capability contributes to the input string processor and the syntax-semantic mapping routines. The processor performs the following sequence of actions everytime an input string is entered into the system:

1. substitute every occurrence of <control_z> to the word "STOP."
2. convert all lowercase characters to uppercase.
3. remove all punctuation marks.
4. perform spelling corrections.
5. translate domain-specific words into word class names, that is, replace words by their pre-defined synonyms.
6. remove irrelevant or extra words (termed non-keywords).
7. save processed string in a system buffer.
8. send processed string to the syntax-semantic action mapping routines.

In order for the input string processor to work properly, the programmer must provide a domain-specific dictionary for the spelling checker and corrector, a list of word class names (or synonyms), and a list of non-keywords.

If the spelling corrector is unable to correct a misspelled word, the above process is terminated at step 4 and the user is notified of the fatal error. After an input string is processed successfully, it is sent to the syntax-semantic action mapping routines. A syntax-semantic action map (SSA_map) represents a mapping between a simple pattern

composed of word class names (the processed string) and some response functions. In other words, one can determine the semantics from the syntax alone. Actions triggered by a SSA_map may include the invocation of action frames, knowledge frames, production rules, GUESS/1 library routines and so forth. Figure 29 shows the general structure of a syntax-semantic action mapping routine. Following it in Figure 30 is a simple example.

7.2 MENU FACILITY

GUESS/1 provides a powerful natural language capability as well as a useful menu facility. The two can also be intermixed. To implement a menu-driven interactive input/output, the programmer has to construct a menu which is to be displayed to the user when the system command MENU is entered. Sub-menus can be implemented using production rules. Figure 31 is an example menu.

7.3 QUESTION-ANSWERING SYSTEM

GUESS/1 provides a built-in library routine ASK_user which constitutes a very powerful and easy-to-implement interactive input/output interface. The following shows the syntax of the routine ASK_user (more details can be found in the chapter on "GUESS/1 Library Routines."):

```
( SSA_map  <Syntax>)
->
( <Semantic Actions> )
```

Figure 29: SSA_map Structure

```
( SSA_map  *Person  GO RESTAURANT )
->
( Add Object_of_Interests  *Person )
( Invoke Action Frame "Restaurant Script" )
```

Figure 30: A SSA_map

```
(MENU  is
 " a) The Sound of Music
   b) I Left My Heart in San Fransico
   c) Evergreen ")
```

Figure 31: A Menu

```
(ASK_USER   QUESTION IS      <text>
            OBJECT IS        <object name>
            EXPECT [RANGE]   <acceptable answers>
            USELESS          <answer that is useless>
            ANSWER IS        <answer>)
```

7.4 SYSTEM CHARACTERISTICS

GUESS/1 was designed such that its system characteristics can be tailored by the programmer and by the user easily. For instance, the system prompt which by default is a percent sign (%) can be changed to any character or any string by the library routine SET PROMPT. For instance, (SET PROMPT "QUERY> ") alters the system prompt from % to QUERY>. Full-screen terminal input/output is also supported by GUESS. Special built-in routines such as CLEARSCREEN are provided for the VT100, TAB, HP2621 and HP2641 terminals. Only the normal terminal I/O routines should be used for other types of terminals.

7.5 SYSTEM COMMANDS

When a user is running the GUESS/1 system, there are many commands available to alter the system characteristics and perform other functions. The system commands are:

- CLEAR - Clearscreen
- CPU - Show elapsed CPU time
- <CTRL-Z> - Stop the system
- ECT - Show elapsed clock time
- HELP - Display help messages
- MAIL - Invoke VAX MAIL
- MENU - Invoke MENU
- MSG - Set SYSTEM MESSAGE on
- NOMSG - Set SYSTEM MESSAGE off
- NOTRACE - Set TRACE off
- REPEAT - Repeat last query
- RN - Generate a random number
- SAY - Invoke VAX SAY
- SPELL - Invoke spelling corrector
- STATUS - Show system status
- STOP - Stop the system
- TIME - Show current date and time
- TRACE - Set TRACE on
- USER - Show users

Chapter VIII

INTERFACE TO THE EXTERNAL WORLD

When two persons walk together,
one is the teacher of the other.

- Confucius

Interfaces to other programming languages, database systems, and the underlying operating system are important parts of an expert systems shell. GUESS/1 provides simple interface mechanisms. For example, a system developer can write a number crunching Fortran program and link it to the system. Interface between GUESS/1 and the operating system allows interprocess communications, access to mail boxes and so on within the GUESS/1 environment. Currently, our GUESS/1 system is interfaced with the VMS operating system on a VAX 11/780. A communication link to the conventional databases is essential for real world applications. Access from GUESS/1 to the FBI criminal records, for instance, has to be efficient. Efficiency is achieved by linking the external DBMS subroutines to GUESS/1 at the source code level.

8.1 AN EXAMPLE INTERFACE PROGRAM

Interfaces to external programming systems is well discussed in details in the Virginia Tech Prolog manual [Roach, 1984]. The following is an example interface program by which a random number generation routine is attached to the GUESS/1 system.

```
[INHERIT('CS:[roach.hc4x6]HC')]
MODULE NEWHC (INPUT,OUTPUT);

[GLOBAL]
FUNCTION f_user (id : int;  fun : string;  args : list) : list;

(* ------------------------------------------------------------ *)
(* A mixed congruential random number generator                 *)
(*   Z(i) = 314159269 * Z(i-1) + 453806245   MOD 2**31-1        *)
(* NB: it changes the seed (args)                               *)
(* ------------------------------------------------------------ *)

FUNCTION u_rand (VAR  args : list;  fun : string) : list;
CONST   D2P31M = 2147483647.0D0;
VAR
        quotient, z:    DOUBLE;
        seed,intquot:   INTEGER;
BEGIN
        if (args = nil) then error (idarg, NONUM)
        else if cklist(args¬.car, ANUM) then z := args¬.car¬.numval
        else error(idarg, NONUM);

        z := 314159269.0D0 * z + 453806245.0D0;
        quotient := z / D2P31M;
        intquot := TRUNC(quotient);
        z := z - intquot * D2P31M;
        seed := TRUNC(z);
        z := z / D2P31M;

        args¬.car¬.numval := seed;
        u_rand := getnum (z);

END;    (* u_rand *)

BEGIN    f_user

    { switching function }
    CASE id OF
        0:  BEGIN
                mkufun ('g$rand\0          ', false, 1);
                f_user := nil;
            END;
        1:  f_user := u_rand (args, fun);

        OTHERWISE error (fun, NOFUN);
    END;
END;      f_user

END.    { MODULE }
```

Chapter IX

DATABASE SECURITY

The name of the Lord is a strong
tower;
the righteous run to it and are
safe.

- Solomon, PROVERBS

GUESS/1 provides facilities to construct, update and retrieve information from the data structures in a knowledge base. Relational tables, trees, semantic nets, frames and production rules can be dynamically created and destroyed. As a user manipulates the knowledge base through a set of GUESS/1 utility routines, security protection can be enforced. The following sections describe GUESS/1 supporting the conventional protective measures required for a trusted database system, and the GUESS/1 knowledge base access control mechanisms, namely the mandatory and the discretionary security protection.

9.1 FUNDAMENTAL REQUIREMENTS

"A secure system is one that will control, through use of specific security features, access to information such that only properly authorized individuals, or processes operating on their behalf, will have access to read, write, create, or delete information." [DoD] There are six fundamental requirements derived from this basic statement of objectives [DoD, 1983]. GUESS/1 deals only with the first four requirements which specify what needs to be provided to control access to information. The other two deals with how one can obtain credible assurances that this is accomplished in a trusted computer system. The following are the four basic requirements that were taken into consideration in the design of GUESS.

- Requirement 1 - SECURITY POLICY - There must be an explicit and well-defined security policy enforced by the system. GUESS/1 adopts the policies of mandatory and discretionary security protection. Read next section for details.
- Requirement 2 - MARKING - Access control labels must be associated with objects. An object in a GUESS/1 knowledge base can be a relational table, a tree, a semantic net, a frame, a script or a production rule. Every such object is labeled with an appropriate access

control tag. The control tag of an object is examined by GUESS/1 routines whenever an access to the object is requested. The tag is used to compare against the user's clearance-tuple to determine if access should be granted or denied. GUESS/1 is also responsible for generating an appropriate access control tag to label a newly-created object in the knowledge base.

- Requirement 3 - IDENTIFICATION - Individual subjects must be identified. GUESS/1 provides a login procedure that requires a user to identify himself/herself. An identification is usually confirmed by a password.
- Requirement 4 - ACCOUNTABILITY - Audit information must be selectively kept and protected so that actions affecting security can be traced to the responsible party. GUESS/1 has a variety of built-in selective trace facilities. Transactions can be automatically traced after a user has logged onto the system. The audit information, including any attempt to violate the security protection enforced on the knowledge base, is kept in a system file inaccessible by any user.

9.2 MANDATORY AND DISCRETIONARY SECURITY PROTECTION

The mandatory security protection provides access controls "based directly on a comparison of the individual's clearance or authorization for the information and the classification or sensitivity designation of the information being sought." [DoD] Each and every database object in GUESS/1 is labeled with a hierarchical classification level and a non-hierarchical category. Examples of a classification level are SECRET and CONFIDENTIAL. The standard relationships among the levels are "greater than (>)," "equal to (=)," and "less than (<)" which are all transitive. For instance, TOP SECRET > SECRET > CONFIDENTIAL > UNCLASSIFIED defines the constant interrelationships among TOP SECRET, SECRET, CONFIDENTIAL and UNCLASSIFIED. SECRET, in this case, is greater than UNCLASSIFIED. Examples of a category are (READ) and (READ WRITE APPEND). Categories are related by the set concept. Category A is a "subset" of B and category B is a "superset" of A if and only if every element in A is present in B. For instance, (READ) is a subset of (READ WRITE APPEND) and (READ WRITE) is a superset of (WRITE). An object can assume a pre-defined label and it can be dynamically re-labeled. On the other hand, every user is assigned a clearance-tuple composed of a security level and a category. A clearance-

tuple is static and cannot be altered by any individual but the system manager. Under the mandatory security policy, access to a knowledge base object is granted only if

1. The user clearance security level is greater than or equal to the object classification security level.
2. The user clearance category is a superset of the object classification category.

GUESS/1 is capable of supporting an arbitrary number of classification levels and categories, thereby facilitates transportability. Figure 32 shows some examples of access request based on clearance.

In addition to the direct comparison of the individual's clearance and the object classification, the discretionary (need-to-know) security policy provides an extra layer of access controls. Under this policy a user is granted access to a knowledge base object only if he/she has the "need-to-know," that is, only if the object or resource is really relevant to the problem at hand. Therefore each and every user is given a list of object identifiers of which the user has the need to know (or to use in the case of resources). Objects that not included in the list are "invisible" (or inaccessible) to the user. An empty list implies no access to the knowledge base. Therefore the knowledge base is

```
Classification: (UNCLASSIFIED, {READ})
     Clearance: (UNCLASSIFIED, {READ WRITE})
Access Request: granted

Classification: (TOP_SECRET, {READ WRITE})
     Clearance: (CONFIDENTIAL, {READ WRITE})
Access Request: denied

Classification: (UNCLASSIFIED, {READ WRITE})
     Clearance: (TOP_SECRET, {WRITE})
Access Request: denied
```

Figure 32: Examples of Access Request Based on Clearance

itself a closed system. On the contrary, a list containing the pre-defined system keyword EVERYTHING implies an unrestricted access to the knowledge base under the discretionary security policy.

The two built-in mandatory and discretionary security protection mechanisms contribute to an explicit and well-defined security policy enforced by GUESS. Any attempted security violation is automatically recorded by the system and the user is warned as a consequence.

Chapter X

GUESS/1 LIBRARY ROUTINES

I think; therefore I am.

- Descartes

GUESS/1 library routines are grouped into 15 functional categories. They are:

- Blackboard Manipulation Routines
- Explanation Subsystem Routines
- GUESS/1 Initialization and Execution Routines
- Hierarchical Tree Manipulation Routines
- Human-System Interface Routines
- Miscellaneous Utility Routines
- Object-List Manipulation Routines
- Relational Table Manipulation Routines
- Semantic Net Manipulation Routines
- Stack Manipulation Routines
- Standard SSA_map Routines
- Syntax-Semantic Action Mapping Routines
- SYSHELP Routines
- System Security Routines
- Terminal Input/Output Routines

Concerning the use of the GUESS/1 library routines, please observe the following:

1. Unless otherwise explicitly stated in the description of a library routine in this chapter, a GUESS/1 library routine always returns true as a value.

2. All GUESS/1 library routines are CAPITALIZED and should remain so when you use them. It is important because the GUESS/1 interpreter is case-sensitive.

3. No library routine can be redefined or overridden because the GUESS/1 routines are always loaded into memory before the application program routines.

10.1 BLACKBOARD MANIPULATION ROUTINES

BLACKBOARD_ERASE

Erase the entire blackboard. This operation cannot be undone.

```
(BLACKBOARD_ERASE)
```

BLACKBOARD_LOOKUP (1)

Look up in the blackboard information written by an action frame (AF).

```
(BLACKBOARD_LOOKUP   AF    IS  *afname
                     KEY   IS  *key
                     VALUE IS  *val)

Input Parameters :   *afname - AF name
                     *key    - search key value
Output Parameters:   *val    - associated value
```

BLACKBOARD_LOOKUP (2)

Look up in the blackboard information written by a knowledge frame (KF).

```
(BLACKBOARD_LOOKUP   KF     IS  *kfname
                     OBJECT IS  *obj
                     KEY    IS  *key
                     VALUE  IS  *val)

Input Parameters :   *kfname - KF name
                     *obj    - object identifier
                     *key    - search key value
Output Parameters:   *val    - associated value
```

BLACKBOARD_UPDATE (1)

Update the blackboard. An action frame (AF) writes information (CONTENT) onto the blackboard. The information is structured as a relational table.

```
(BLACKBOARD_UPDATE   AF      IS  *afname
                     CONTENT IS  *contents)

Input Parameters:    *afname   - AF name
                     *contents - contents
```

```
Example:

(BLACKBOARD_UPDATE
            AF      IS  "Acid-Base Reactions"
            CONTENT IS (  (Chemical_A acidic)
                          (Chemical_B basic)
                          (Neutralization 100%)
                       )
)
```

BLACKBOARD_UPDATE (2)

Update the blackboard. A knowledge frame (KF) writes information (CONTENT) about an object (OBJECT) onto the blackboard. The information is structured as a relational table.

```
(BLACKBOARD_UPDATE   KF      IS  *kfname
                     OBJECT  IS  *obj
                     CONTENT IS  *contents)

Input Parameters:    *kfname   - KF name
                     *obj      - object id
                     *contents - contents
```

```
Example:

(BLACKBOARD_UPDATE
            KF      IS  "A Person"
            OBJ     IS  "X"
            CONTENT IS (  (name    X)
                          (age    21)
                          (sex  male)
                       )
)
```

10.2 EXPLANATION SUBSYSTEM ROUTINES

SYSMSG

Print the given message if the system message display flag (MSG) is ON. The message is ignored if MSG is OFF.

```
(SYSMSG  *message)

Input Parameters:   *message - a string, an atom, or
                               a list of atoms
```

USAGE

Indicate the usage of a knowledge frame, an action frame or a production rule. The description is printed if the TRACE flag is on. The level number indicates the level of the frame or rule in the context of an AND/OR goal tree. If ANY is given as the level number, the current level number takes on the level number of the calling frame or rule.

```
(USAGE  *desp  LEVEL  *level)

Input Parameters:   *desp  - description
                    *level - level number
```

10.3 GUESS/1 INITIALIZATION AND EXECUTION ROUTINES

GUESS/1

Execute GUESS/1.

```
(GUESS)
```

GETCOMMAND

Get the GUESS/1 first level commands.

```
(GETCOMMAND)
```

GETTERMTYPE

Get the terminal type from the file GTERM.DAT.

```
(GETTERMTYPE)
```

SHOWCOMMAND

Display the system commands.

```
(SHOWCOMMAND)
```

START

Start execute GUESS/1 and perform the proper initializations.

```
(START)
```

10.4 HIERARCHICAL TREE MANIPULATION ROUTINES

TREE_ADD

Add an new item to a tree. It prints error message if the given parent does not exist.

```
(TREE_ADD
      TREENAME IS  *name
      BRANCH   IS  *parent  *newchild  [ *treeid ] )

Input Parameters:  *name     - name of a tree
                   *parent   - parent identification
                   *newchild - new descendant(s)
                   *treeid   - new tree name,
                               specified only if the
                               newchild is the root
                               of another new tree
```

TREE_CREATE (1)

Create a new tree. It assumes that the security level for the new tree is UNCLASSIFIED and that its category is (READ). It prints an error message if the new tree already exists.

```
(TREE_CREATE
          TREENAME IS  *name
          ROOT     IS  *root)

Input Parameters:   *name - name of a new tree
                    *root - root of the new tree
```

TREE_CREATE (2)

Create a new tree. It prints an error message if the new tree already exists.

```
(TREE_CREATE
          TREENAME IS  *name
          ROOT     IS  *root
          SECURITY IS  *sec
          CATEGORY IS  *cat)

Input Parameters:   *name - name of a new tree
                    *root - root of the new tree
```

TREE_DESTROY

Destroy an old tree. It prints an error message if the given tree does not exist.

```
(TREE_DESTROY
          TREENAME IS  *name)

Input Parameter:   *name - name of a tree
```

TREE_ERASE

Erase a leaf node (terminal node) from a tree. Care must be taken as not to delete a non-terminal which would result in a disconnected tree. It prints an error message if the given parent or child does not exist.

```
(TREE_ERASE
            TREENAME IS  *name
            PARENT   IS  *parent
            CHILD    IS  *child)

Input Parameters:     *name - name of a tree
                      *parent - node id
                      *child - leaf node
```

TREE_EXPAND

Traverse a knowledge tree from a given node, that is, to collect all the descendants of the given node.

```
(TREE_EXPAND    TREENAME IS  *name
                NODE     IS  *node
                TREE     IS  *tree)

Input Parameters:  *name - tree name
                   *node - node name
Output Parameter:  *tree - expanded tree
```

TREE_LOOKUP

Look up some parents or some children from a hierarchical tree. If the parent is given, it looks for the children. If the children are given, it looks for the parent. If both the parent and the children are given, it checks for the existence of such a parent-children relationship.

```
(TREE_LOOKUP   TREENAME IS  *name
               PARENT   IS  *parent
               CHILDREN IS  *children)

Input Parameters :   *name     - tree name
                     *parent   - parent node
                     *children - children node
Output Parameters:   *parent   - parent node
                     *children - children node
```

TREE_UPDATE

Update an old tree. It creates a new root and forms a new branch between the old root and the new root. It prints an error message if the given root or the given tree does not exist.

```
(TREE_UPDATE
          TREENAME IS  *name
          OLDROOT  IS  *old
          NEWROOT  IS  *new)

Input Parameters:  *name - tree name
                   *old  - old root name
                   *new  - new root name
```

10.5 HUMAN-SYSTEM INTERFACE ROUTINES

ASK_USER (1)

Ask user a question. The body of the question should always be a string of characters or an atom. The object of interest is an atom to be concatenated to the end of the question body (with a blank in between). The object of interest can be "nil." The list of acceptable answers should be a list of atoms such as (yes no unknown). The USELESS answer is a specific atom among the acceptable answers. If the user chooses the USELESS answer, ASK_USER will return nil and cause a backtrack to its caller. NOTE: ASK_USER I/O IS CASE-SENSITIVE. FOR EXAMPLE: (yes no) IS DIFFERENT FROM (Yes No).

```
(ASK_USER   QUESTION IS  *ques
            OBJECT   IS  *obj
            EXPECT       *anslist
            USELESS      *useless
            ANSWER   IS  *ans)

Input Parameters:   *ques     - question body
                    *obj      - object of interest
                    *anslist  - acceptable answers
                    *useless  - useless answer
Output Parameter:   *ans      - answer from the user
```

Example:

```
(ASK_USER  QUESTION IS "Do you know anything about"
           OBJECT   IS GUESS/1
           EXPECT   (yes no unknown)
           USELESS  unknown
           ANSWER   IS  *ans)
```

generates

```
QUESTION: Do you know anything about GUESS/1?
CHOOSE  : yes / no / unknown
ANSWER  ?
```

ASK_USER (2)

Ask user a question. It is similar to the first ASK_USER routine except that the acceptable answer is now a range of values rather than a few discrete choices. The range of values can be integers, real numbers, single characters, or character strings. The range is specified by the list (GE *lower_limit LE *upper_limit). The list is interpreted as: any input "Greater than or Equal to" the *lower_limit and "Lower than or Equal to" the *upper_limit is acceptable. NOTE: ASK_USER IS CASE-SENSITIVE. FOR EXAMPLE: (yes no) IS DIFFERENT FROM (Yes No).

```
(ASK_USER    QUESTION IS  *ques
             OBJECT   IS  *object
             EXPECT RANGE *range
             USELESS      *useless
             ANSWER   IS  *ans)

Input Parameters:   *ques    - question body
                    *obj     - object of interest
                    *range   - acceptable answers
                    *useless - useless answer
Output Parameter:   *ans     - answer from the user
```

```
Examples:

(ASK_USER  QUESTION IS    "Enter a real number"
           OBJECT   IS    nil
           EXPECT RANGE   (GE -5.87 LT 10.86)
           USELESS        1.00
           ANSWER   IS    *ans)

(ASK_USER  QUESTION IS    "Enter a string"
           OBJECT   IS    nil
           EXPECT RANGE   (GE aa LT za)
           USELESS        uz
           ANSWER   IS    *ans)
```

g$spell

Given a string of characters, it returns a new string with all the possible spelling corrections made. It prints all the misspelled words and their correct spellings.

```
(g$spell  *input_string)

Input Parameter:   *input_string - a string
```

```
Example:

(:=  *outstr  (g$spell  "HOW ARE YU"))
```

MENU

Set up a display menu. The menu is displayed when the user enters the system command MENU.

```
(MENU IS  *menu)

Input Parameter:   *menu - a string (menu)
```

```
Example:

(MENU IS
 " 1] My Fair Lady
   a] The Sound of Music
   X] The Good, The Bad and The Ugly ")
```

10.6 MISCELLANEOUS UTILITY ROUTINES

APPENDA

Append an atom to the beginning of a given list.

```
(APPENDA  *atom  *list  *newlist)

Input Parameters :   *atom    - an atom
                     *list    - an old list
Output Parameters:   *newlist - a new list
```

CHECK_WORD

Convert a given word to its uppercase equivalence and perform spelling correction.

```
(CHECK_WORD  *in  *out)

Input Parameters :   *in  - input word
Output Parameters:   *out - processed word
```

DECREMENT

Decrement the internal counter by a given amount.

```
(DECREMENT COUNTER  *val)

Input Parameters:   *val - amount to be subtracted
```

DEFAULT_OF

Set a default value. If the system message flag MSG is on, it prints a notification message whenever DEFAULT_OF is invoked.

```
(DEFAULT_OF  *val  IS  *val)

Input Parameters :    *val (2nd) - default value
Output Parameters:    *val (1st) - variable to take on
                                   the default value
```

DELETE_MEMBER

Delete an item from a given list. It returns the same list if the given item is not in the list.

```
(DELETE_MEMBER  *item  *list  *newlist)

Input Parameters:     *item    - an item
                      *list    - a list of items
Output Parameter:     *newlist - a new list of items
```

DIFFERENT_SET

Return true if the two given sets are different, return nil otherwise.

```
(DIFFERENT_SET  *set1  *set2)

Input Parameters:   *set1 - first set
                    *set2 - second set
```

DISPLAY_HR_MIN_SEC

Display the given time in hr:min:sec format.

```
(DISPLAY_HR_MIN_SEC  *hr  *min  *sec)

Input Parameters:   *hr - hour
                    *min - minute
                    *sec - second
```

GET PROMPT

Obtain the current system prompt.

```
(GET PROMPT  *prompt)

Output Parameter:   *prompt - system prompt
```

GETELEMENT

Get the n'th element from a given list. It returns nil if the pointer value *ptr is less than 1 or if the pointer value is greater than the length of the given list.

```
(GETELEMENT  *list  *ptr  *elem)

Input Parameters :   *list - a list
                     *ptr  - pointer to n'th element
Output Parameters:   *elem - n'th element
```

g$stratm

Given a string of characters, it returns a list of atoms (words).

```
(g$stratm  *input_string)

Input Parameter:   *input_string - a string
```

```
Example:

(:= *list (g$stratm  "Virginia and California"))

returns a list:

(Virginia  and  California)
```

INCREMENT COUNTER

Increment the internal counter by a given amount.

```
(INCREMENT COUNTER  *val)
Input Parameter:   *val - amount to be added
```

INCREMENT QUERY#

Increment the internal counter which keeps track of the total number of input queries.

```
(INCREMENT QUERY#)
```

INCREMENT SYSCMD#

Increment the internal counter which keeps track of the total number of system command calls.

```
(INCREMENT SYSCMD#)
```

INDENT

Indent (print tab characters) to a given level. No carriage return is issued.

```
(INDENT  *level)

Input Parameters:   *level - level of indentation
```

LISTLEN

Return the length of a given list.

```
(LISTLEN  *list  *len)

Input Parameter :   *list - a list
Output Parameter:   *len - length of the list
```

LPRINT

Line print. It prints an atom if an atom is given. If a list of atoms is given, it numbers all the atoms in the list and prints them, one per line.

```
(LPRINT  *list/atom)

Input Parameter:   *list/atom - a list or an atom
```

MEMBER

Return true if a given item is a member of a given list, return nil otherwise.

```
(MEMBER  *item  *list)

Input Parameters:  *item - an item
                   *lits - a list
```

NOT_MEMBER

Return true if a given item is not a member of a given list, return nil otherwise.

```
(NOT_MEMBER  *item  *list)

Input Parameters:  *item - an item
                   *lits - a list
```

NUMERIC

Return true if the given item is a number (integer or real), return nil otherwise.

```
(NUMERIC  *number)

Input Parameter:   *number - anything
```

PRECISION

Reset the number of digits after the decimal point.

```
(PRECISION  *precision)
(PRECISION  DEFAULT)

Input Parameter:   *precision - the number of digits
                                after the decimal
                                point (Default is 8)
```

PRINT_LIST

Print all the elements in a given list, one per line with numbering.

```
(PRINT_LIST  *list)

Input Parameter:   *list - a list of elements
```

RESET CPU

Reset CPU time to 0.

```
(RESET CPU)
```

RESET QUERY#

Reset the query counter to 0.

```
(RESET QUERY#)
```

RESET SYSCMD#

Reset the system command call counter to 0.

```
(RESET SYSCMD#)
```

RESET TIME

Reset the elapsed clock time to 0.

```
(RESET TIME)
```

RET_QUESTION

Retrieve the stored query.

```
(RET_QUESTION  *que  *ssa)

Output Parameters:   *que - question to be retrieved
                     *ssa - ssa to be retrieved
```

RN_SEED

Supply an initial seed for the built-in random number generator.

```
(RN_SEED  *seed)

Input Parameter:   *seed - initial seed for RNG
```

RN_SEED RANDOM

Generate an initial seed for the built-in random number generator. The initial seed is a function of the current clock time.

```
(RN_SEED  RANDOM)
```

RN

Generate a random number (between 0 and 1) from the built-in random number generator.

```
(RN  *rn)

Output Parameter:   *rn - random number
```

SAME_SET

Return true if the two given sets are the same, return nil otherwise.

```
(SAME_SET  *set1  *set2)

Input Parameters:   *set1 - first set
                    *set2 - second set
```

SAVE KNOWLEDGE_BASE

Save the knowledge base in a permanent file (knowledge.bse). The file is automatically loaded back into memory the next time you execute GUESS/1.

```
(SAVE KNOWLEDGE_BASE)
```

SAVE_QUESTION

Save the input query and the ssa in a system buffer. System command calls are not saved.

```
(SAVE_QUESTION  *que  *ssa)

Input Parameters:   *que - query to be saved
                    *ssa - ssa to be saved
```

SET COUNTER

Set the internal counter to a given value.

```
(SET COUNTER  *val)

Input Parameter:   *val - initial value
```

SET PROMPT

Set the system prompt to a given string.

```
(SET PROMPT  *prompt)

Input Parameter:   *prompt - new system prompt
```

SET SYSMSG OFF

Turn off the system message display.

```
(SET SYSMSG OFF)
```

SET SYSMSG ON

Turn on the system message display.

```
(SET SYSMSG ON)
```

SET TERMINAL ECHO

Set the terminal to echoing input characters from the keyboard.

```
(SET TERMINAL ECHO)
```

SET TERMINAL NOECHO

Set the terminal such that it does not echo input characters from the keyboard.

```
(SET TERMINAL NOECHO)
```

SET TRACE OFF

Turn TRACE off.

```
(SET TRACE OFF)
```

SET TRACE ON

Turn TRACE on.

```
(SET TRACE ON)
```

SHOW APPLICATION

Display the application identification.

```
(SHOW APPLICATION)
```

SHOW COUNTER

Display the content of the internal counter.

```
(SHOW COUNTER)
```

SHOW CPU

Display the elapsed CPU time.

```
(SHOW CPU)
```

SHOW DATETIME

Display the current date and time.

```
(SHOW DATETIME)
```

SHOW FLAGS

Display the system flags.

```
(SHOW FLAGS)
```

SHOW TIME

Display the elapsed clock time.

```
(SHOW TIME)
```

SUBSET_OF

Return true if set2 is a subset of set1, return nil otherwise.

```
(SUBSET_OF  *set1  IS  *set2)

Input Parameters:   *set1 - a list
                    *set2 - a list
```

SUCCEED

Always return true.

```
(SUCCEED)
```

SUPERSET_OF

Return true if set2 is a superset of set1, return nil otherwise.

```
(SUPERSET_OF  *set1  IS  *set2)

Input Parameters:   *set1 - list
                    *set2 - list
```

SYSCOMMAND

Return true if the input string is a system command, return nil otherwise.

```
(SYSCOMMAND  *command)
Input Parameters:   *command - input string
```

SYSTEM

Send a system command to the underlying operating system.

```
(SYSTEM  *command)
Input Parameters:   *command - an OS command
```

10.7 OBJECT-LIST MANIPULATION ROUTINES

ADD

Add an object of interest or a list of objects to the front of the current object list. The currency pointer to the object list is reset to "undefined".

```
(ADD  OBJECT_OF_INTEREST  *obj)

Input Parameter:   *obj - an object of interest, or
                          a list of objects
```

GETALL

Obtain the whole object list. The currency pointer to the object list remains unchanged.

```
(GETALL  OBJECT_OF_INTEREST  *objlist)

Output Parameter:   *objlist - object list
```

GETCURRENT

Get the current object from the object list pointed to by the currency pointer. The GETCURRENT must not be issued if neither a GETFIRST nor a GETNEXT operation has previously been issued to establish the currency pointer. If the currency pointer is pointing to the end of the list, *obj returns a nil.

```
(GETCURRENT  OBJECT_OF_INTEREST  *obj)
Output Parameter:   *obj - object of interest
```

GETFIRST

Get the first object from the object list. The currency pointer is set to point to the beginning of the object list. If the object list is empty, *obj returns a nil.

```
(GETFIRST  OBJECT_OF_INTEREST  *obj)
Output Parameter:   *obj - object of interest
```

GETNEXT

Get the next object from the object list. The currency pointer is set to point to the next object in the object list. If the end of the list is encountered, *obj returns a nil.

```
(GETNEXT  OBJECT_OF_INTEREST  *obj)
Output Parameter:   *obj - object of interest
```

OBJECTLIST_ERASE

Erase the entire OBJECTLIST. This operation cannot be undone.

```
(OBJECTLIST_ERASE)
```

10.8 RELATIONAL TABLE MANIPULATION ROUTINES

TABLE_ADD

Add a new item to a table. It prints an error message if the given item already exists or the given table does not exist.

```
(TABLE_ADD
        TABLE IS  *tbl
        KEY   IS  *key
        VALUE IS  *val)

Input Parameters:     *tbl - table name
                      *key - key name
                      *val - value
```

TABLE_CREATE (1)

Create a new table. It assumes that the security level for the new table is UNCLASSIFIED and that its category is (READ). It prints an error message if the given table already exists.

```
(TABLE_CREATE
          TABLE IS  *tbl)

Input Parameter:      *tbl - table name
```

TABLE_CREATE (2)

Create a new table. It print an error message if the given table already exists.

```
(TABLE_CREATE
            TABLE    IS  *tbl
            SECURITY IS  *sec
            CATEGORY IS  *cat)

Input Parameters:   *tbl - table name
                    *sec - security level
                    *cat - category
```

TABLE_DESTROY

Destroy an old table. It prints an error message if the given table does not exist.

```
(TABLE_DESTROY
            TABLE IS  *tbl)

Input Parameter:    *tbl - table name
```

TABLE_ERASE

Erase an item from a table. It prints an error message if the item or the given table does not exist.

```
(TABLE_ERASE
          TABLE IS  *tbl
          KEY   IS  *key
          VALUE IS  *val)

Input Parameters:     *tbl - table name
                      *key - key name
                      *val - value
```

TABLE_LOOKUP

Look up information from a relational table.

```
(TABLE_LOOKUP   TABLE IS  *tbl
                KEY   IS  *key
                VALUE IS  *val)

Input Parameters :    *tbl - table name
                      *key - key name
Output Parameters:    *val - associated value
```

```
Example:

(TABLE_LOOKUP   TABLE IS  "Student/GPA"
                KEY   IS  John
                VALUE IS  *GPA)
```

10.9 SEMANTIC NET MANIPULATION ROUTINES

DNET_ADD

Add an arc to a directed graph. It prints an error message if the given arc already exists.

```
(DNET_ADD
        NETNAME  IS  *name
        ARC      IS  *node1  *rel  *node2)

Input Parameters:    *name  - net name
                     *node1 - node1
                     *rel   - relation name (-->)
                     *node2 - node2
                     (relation  node1 --> node2)
```

DNET_CREATE (1)

Create a new directed graph. The security level and the category for the new graph are assumed to be UNCLASSIFIED and (READ) respectively. It prints an error message and returns nil if the given semantic net already exists.

```
(DNET_CREATE
          NETNAME  IS  *name)

Input Parameter:     *name - net name
```

DNET_CREATE (2)

Create a new directed graph. It prints an error message and return nil if the given network already exists.

```
(DNET_CREATE
          NETNAME  IS  *name
          SECURITY IS  *sec
          CATEGORY IS  *cat)

Input Parameters:    *name - net name
                     *sec  - security level
                     *cat  - category
```

DNET_DESTROY

Destroy an old directed graph. It prints an error message and return nil if the given network does not exist.

```
(DNET_DESTROY
          NETNAME  IS  *name)

Input Parameter:     *name - net name
```

DNET_ERASE

Erase an arc from a directed graph. It prints an error message and return nil if the given arc dos not exist.

```
(DNET_ERASE
        NETNAME  IS  *name
        ARC      IS  *node1  *rel  *node2)

Input Parameters:    *name  - net name
                     *node1 - node1
                     *rel   - relation name (-->)
                     *node2 - node2
                     (relation  node1 --> node2)
```

DNET_RETRIEVE (1)

Find all the nodes pointed to by the given node in a directed graph. If the relation is not given, it will try to find all such nodes and their relationships to the given node.

```
(DNET_RETRIEVE
              NETNAME  IS  *name
              NODE     IS  *node
              RELATION IS  *relation
              NEIGHBOR IS  *neighbor)


Input Parameters:     *name     - net name
                      *node     - node id
                      *relation - a relation
Output Parameters:    *neighbor - a list of neighbors
                      *relation - a list of relations
```

DNET_RETRIEVE (2)

Find the relationship between two given nodes in a directed graph. It gives the direction of the relationship as either "->" or "<-". If there is no relation between the two given nodes, it returns a "nil" relationship.

```
(DNET_RETRIEVE
              NETNAME   IS  *name
              NODE1     IS  *node1
              NODE2     IS  *node2
              RELATION  IS  *rel
              DIRECTION IS  *dir)

Input Parameters:     *name  - name of a network
                      *node1 - node id
                      *node2 - node id
Output Parameters:    *rel   - a relation
                      *dir   - a direction
                               ('->' or '<-')
```

UNET_ADD

Add an arc to an undirected graph. It prints an error message if the given arc already exists.

```
(UNET_ADD
      NETNAME  IS  *name
      ARC      IS  *node1  *node2)

Input Parameters:   *name - net name
                    *node1 - node1
                    *node2 - node2
```

UNET_CREATE (1)

Create a new undirected graph. The security level and the category for the new graph are assumed to be UNCLASSIFIED and (READ) respectively. It prints an error message and return nil if the given network already exists.

```
(UNET_CREATE
          NETNAME  IS  *name)

Input Parameter:     *name - net name
```

UNET_CREATE (2)

Create a new undirected graph. It prints an error message and return nil if the given network already exists.

```
(UNET_CREATE
          NETNAME  IS  *name
          SECURITY IS  *sec
          CATEGORY IS  *cat)

Input Parameters:     *name - net name
                      *sec  - security level
                      *cat  - category
```

UNET_DESTROY

Destroy an old undirected graph. It prints an error message and return nil if the given network does not exist.

```
(UNET_DESTROY
          NETNAME  IS  *name)

Input Parameter:     *name - net name
```

UNET_ERASE

Erase an arc from an undirected graph. It prints an error message and return nil if the given arc does not exist.

```
(UNET_ERASE
         NETNAME  IS  *name
         ARC      IS  *node1  *node2)

Input Parameters:     *name  - net name
                      *node1 - node1
                      *node2 - node2
```

UNET_RETRIEVE

Find all the neighbors of a given node in an undirected graph.

```
(UNET_RETRIEVE
                NETNAME  IS  *name
                NODE     IS  *node
                NEIGHBOR IS  *neighbor)

Input Parameters:      *name     - net name
                       *node     - node id
Output Parameters:     *neighbor - a list of neighbors
```

10.10 STACK MANIPULATION ROUTINES

INIT_STACK

Initialize STACK to nil.

```
(INIT_STACK)
```

POP

Pop the top element off the STACK.

```
(POP)
```

PUSH

Push an item onto the STACK.

```
(PUSH *item)
Input Parameter:   *item - item to be pushed
```

TOP_OF_STACK

Return the top element on the STACK. The STACK remains unchanged.

```
(TOP_OF_STACK)
```

```
Example:
(:=  *element  (TOP_OF_STACK))
```

10.11 STANDARD SSA_MAP ROUTINES

The standard SSA_MAP routines are included here for your references only.

```
(SSA_MAP  (CLEAR))        Clearscreen.
(SSA_MAP  (CPU))          Show elapsed CPU time.
(SSA_MAP  (ECT))          Show elapsed clock time.
(SSA_MAP  (HELP))         Display help messages.
(SSA_MAP  (MAIL))         Invoke VAX MAIL.
(SSA_MAP  (MENU))         Invoke MENU.
(SSA_MAP  (MSG))          Set SYSTEM MESSAGE on.
(SSA_MAP  (NOMSG))        Set SYSTEM MESSAGE off.
(SSA_MAP  (NOTRACE))      Set TRACE off.
(SSA_MAP  (REPEAT))       Repeat last query.
(SSA_MAP  (RN))           Generate random number.
(SSA_MAP  (SAY))          Invoke VAX SAY.
(SSA_MAP  (SPELL))        Invoke spelling corrector.
(SSA_MAP  (STATUS))       Show system status.
(SSA_MAP  (STOP))         Stop the system.
(SSA_MAP  (TIME))         Show current time/date.
(SSA_MAP  (TRACE))        Set TRACE on.
(SSA_MAP  (USER))         Show users.
```

10.12 SYNTAX-SEMANTIC ACTION MAPPING ROUTINES

GET_QUESTION

Read an input string and process it by removing punctuation marks and non-keywords, performing spelling corrections, substituting words for their synonyms, saving the processed string, and so forth.

```
(GET_QUESTION  *question)

Output Parameter:   *question - query or command
```

REMOVE_NONKEYWORDS

Remove non-keywords from the input string.

```
(REMOVE_NONKEYWORDS  *instr  *outstr)

Input Parameter :   *instr  - input raw string
Output Parameter:   *outstr - processed string
```

REMOVE_PUN

Remove punctuation marks from the input string.

```
(REMOVE_PUN  *instr  *outstr)

Input Parameter:   *instr  - input raw string
Output Parameter:  *outstr - processed string
```

SUB_CONTROL_Z

Remove <control_z> from the input string and substitute it for the word "STOP."

```
(SUB_CONTROL_Z  *instr  *outstr)

Input Parameter:   *instr  - input raw string
Output Parameter:  *outstr - processed string
```

SUBSTITUTE_SYNONYMS

Substitute words for their synonyms in the input string.

```
(SUBSTITUTE_SYNONYMS  *instr  *outstr)

Input Parameter :   *instr  - input raw string
Output Parameter:   *outstr - processed string
```

10.13 SYSHELP ROUTINES

SYSHELP

Display help messages about the system commands.

```
(SYSHELP)
```

10.14 SYSTEM SECURITY ROUTINES

CLASSIFICATION

Compare the user clearance against the system object classification to determine if the requested access should be granted or denied. A system object classification is given as a pair: (security level, categories). The possible security levels are TOP_SECRET, SECRET, CONFIDENTIAL and UNCLASSIFIED, and the possible categories are READ, WRITE, READ and WRITE. Access is allowed only if user clearance >= system object classification, i.e. user security level >= system object security level, and the user CATEGORY IS a superset of the system object category. It prints a warning message if access is denied.

```
(CLASSIFICATION  *security_level  *category)

Input Parameters:   *security_level - security level
                    *category       - categories
```

LOGIN

Allow a user to logon to the system with a proper user identification and password. If the identification and/or the password are incorrect, an error message is issued.

```
(LOGIN)
```

LOGOUT

Allow a user to logout.

```
(LOGOUT)
```

NEED_TO_KNOW

Checks the user need-to-know set to determine if a requested access should be granted. Access is allowed only if the system object sought is an element of the user need-to-know set. A warning message is printed if the access is denied.

```
(NEED_TO_KNOW  *object)
Input Parameter:   *object - object identification
```

10.15 TERMINAL INPUT/OUTPUT ROUTINES

CLEARSCREEN

Clear the screen. It works on the VT100, TAB, HP 2621 and HP 2641 terminals. If CLEARSCREEN is invoked for other types of terminals, a warning message is printed.

```
(CLEARSCREEN)
```

HOLD_MORE

Print a MORE... message and wait for user to press the return key.

```
(HOLD_MORE)
```

SET CURSOR

Put the cursor at a given row and column on the screen. It works on the VT100, TAB, HP 2621 and HP 2641 terminals. If SET CURSOR is invoked for other types of terminals, a warning message is printed.

```
(SET CURSOR  *row  *col)

Input Parameters:   *row - row number
                    *col - column number
```

SET TERM$TYPE

Set the internal flag term$type to a given terminal type.

```
(SET TERM$TYPE *term$type)

Input Parameters:   *term$type - terminal type
```

10.16 GUESS/1 LIBRARY ROUTINES DISTRIBUTION

The GUESS/1 library routines are distributed in 15 files[4], each of which corresponds to a functional group described above. Figure 33 shows the distribution of the routines.

---

[4] The file GSTART2.HC is the same as GSTART.HC except that it contains the system security initialization routines.

File Name	Functional Group
GBLACKBD.HC	Blackboard Manipulation Routines
GIO.HC	Human-System Interface Routines
GMENU.HC	Menu-Driven Capability Routines
DNETWORK.HC	Semantic Nets Manipulation Routines
GOBJECT.HC	Object-List Manipulation Routines
GSECURITY.HC	System Security Routines
GSSAMAP.HC	Syntax-Semantic Action Mapping Routines
GSTACK.HC	Stack Manipulation Routines
GSTART.HC	GUESS/1 Initialization/Execution Routines
GSTART2.HC	GUESS/1 Initialization/Execution Routines
GSTDSSA.HC	Standard SSA_MAP Routines
GSYSHLP.HC	SYSHELP Routines
GTABLE.HC	Relational Table Manipulation Routines
GTERMIO.HC	Terminal Input/Output Routines
GTREE.HC	Hierarchical Tree Manipulation Routines
GUTILITY.HC	GUESS/1 Utility Routines

Figure 33: GUESS/1 Library Routines Distribution

Chapter XI

APPLICATION: THE DRUG INTERACTION EXPERT SYSTEM

Knowledge, you see,
  has no uses without purpose,
but purpose is what builds
  enclosing walls.

- Frank Herbert, CHILDREN OF DUNE

11.1 INTRODUCTION

The need for a system to aid medical practitioners by identifying, cataloging and recalling possible drug interactions is evident with the multitude of drugs in use today. Manifestations of interactions and therapeutic failures are often considered a result of the disease or an effect of a single agent. Reports of potentially dangerous combinations administered to hospital patients have ranged from 7% to 52% [Hussar, 1980; Morrelli & Melmon, 1978]. There are a number of good reasons too, that a clinician may choose to use multiple drug therapy, including multiple problems associated with a single disease (ie., hypertension and hyperglycemia with diabetes mellitus), patients with multiple diseases (ie., a diabetic with an infection), and the possibility that a single therapeutic objective may best

be reached when more than one drug is used (ie., analgesia with aspirin and codeine). The Drug Interaction Expert System (DIES-II) was developed to identify interactions that are both adverse and beneficial. DIES-II is an improvement over the first generation drug interaction expert system built by the authors [Roach, Lee, Wilcke & Ehrich, 1985; Ehrich, Wilcke, Lee & Roach, 1984]. Some of the advantages of DIES-II are better human engineering, more sophisticated problem-solving techniques, and ease of modification and expansion of the knowledge base.

DIES-II draws conclusions about drug interactions from a store of knowledge based on certain biochemical properties of the drugs involved (ie. pKa, lipid and aqueous solubility, binding characteristics and metabolism) and physiological conditions of the patient (ie. urine pH, age, condition of liver and species). This knowledge is made available to the medical practitioner through a natural language interface. The system, already able to provide consultant information in a limited sense, may also be used as a simple database since considerable information is stored for specific drugs and classes of drugs. A clinician may thus ask for all information related to a specific drug and receive back information on the drug's acidity, pKa, water solubility, protein binding characteristics,

metabolism and so forth for determining possible interactions. The system can also give reasonably extensive help to novices who have never used it before. The help and question asking facilities are sufficient so that the system can be termed "user-friendly."

In this chapter, we will go through the entire development process of DIES-II which illustrates the use of GUESS/1.

11.2 SYSTEM OBJECTIVES AND DESIGN

A good design is essential in building a successful expert system for practical use. As a rule of thumb, the very first step in any design is the definition of the objectives. Here are the objectives for DIES-II:

1. The system should be able to tell all possible interactions between two given drugs.
2. The system should be able to suggest corrective actions to avoid or alleviate the adverse effects caused by the interactions of two given drugs.
3. The system should be able to retrieve the properties of a given drug.
4. The system should be able to answer simple queries about the classification and subclasses of a given drug.

With these clear objectives in mind, we are ready to proceed. To build an expert system is to construct a model of a human expert in some specific problem solving domain. The life cycle of model development can be divided into six basic phases [Balci, 1983; Nance, 1981]. They are:

1. System Definition and Objectives - The objectives of developing a model and the definitions of the system.
2. Conceptual Model - "A model which exists in the mind of the modeler. The form of the conceptual model is influenced by the system, the perceptions of the system held by the modeler, and the objectives of the study." [Nance, 1981]
3. Communicative Model - "A model representation which can be communicated to other humans, can be judged or compared against the system and the study objectives by more than one human." [Nance, 1981]
4. Programmed Model - "A model representation that admits execution by a computer to produce results." [Nance, 1981]
5. Experimental Model - "A programmed model incorporating an executable description of the experimental design." [Balci, 1983]
6. Model Results - "The outcomes from a single execution of the experimental model or those results produced to satisfy a single test scenario." [Nance, 1981]

The process of model development is, however, by no means sequential. It is iterative in nature and bounces back and forth between the phases during the development (see Figure 34).

Although this life cycle has been developed primarily for simulation models, it is also very applicable to the construction of expert systems. In this section, we have stated the system objectives. The next step is to construct a conceptual model for DIES-II.

11.3 CONCEPTUAL MODEL

We want the system to be a medical consultant or expert on drug interactions. Therefore, we consulted with some human experts in the field and extracted their knowledge about drug interactions. A conceptual model of DIES-II was gradually formulated. In order to simplify the model, several assumptions have been made:

- Only two drugs at a time are to be considered.
- The physical condition of the patient is more or less normal. For instance, the body pH is around 7.4, and there is no recent major physical injury.
- The normal route of administration is being practised.

```
---------------------      1      --------------
| SYSTEM DEFINITION |/---------\| CONCEPTUAL |
|  AND OBJECTIVES   |\---------/|    MODEL   |
---------------------             --------------
    /\                                 /\
    ||  6                              ||  2
    \/                                 \/
-----------                    -----------------
|  MODEL  |                    | COMMUNICATIVE |
| RESULTS |                    |     MODEL     |
-----------                    -----------------
    /\                                 /\
    ||  5                              ||  3
    \/                                 \/
----------------       4         --------------
| EXPERIMENTAL |/--------------\| PROGRAMMED |
|     MODEL    |\--------------/|    MODEL   |
----------------                --------------
```

LEGENDS:

1 = Model Formulation
2 = Model Representation
3 = Programming
4 = Experimental Design
5 = Experimentation
6 = Re-definition

Figure 34: The Life Cycle of Model Development

11.4 COMMUNICATIVE MODEL

Knowledge extracted from the human experts is then organized and represented as relational tables, hierarchical trees, semantic nets, knowledge frames, action frames and production rules. Decisions are made about what specific kinds of knowledge are to be included in the system, and about what strategies are to be used to solve the problems stated as the system objectives.

11.4.1 Relational Tables

Information about acidity, pKa, water solubility, nephrotoxicity, protein binding and so on that are related to the properties of a given drug are best stored in some relational tables. Figure 35 shows a table about the nephrotoxicity of a drug.

11.4.2 Hierarchical Trees

Information about drug classifications, specific examples of a class of drug and drug interaction mechanisms are best stored in some hierarchical trees. Figure 36 shows a partial drug classification tree.

TABLE NAME: nephrotoxicity of a drug
USAGE: store the nephrotoxicity of drugs
KEY: drug name
VALUE: yes / no / unknown
SECURITY: UNCLASSIFIED
CATEGORY: (READ)

AUTHOR: Newton S. Lee
DATE: January 1985

CONTENTS:

Drug	Nephrotoxicity
AMINOGLYCOSIDES	yes
AMPHOTERICIN-B	unknown
ANTIBIOTICS	unknown
ANTIFUNGALS	yes
CEPHALOSPORINS	no
CHLORAMPHENICOL	yes
LINCOSAMIDES	yes
MACROLIDES	yes
PENICILLINS	unknown
POLYMYXINS	yes
TETRACYCLINES	yes

Figure 35: Nephrotoxicity of a Drug

```
TREE NAME:    drug classification
USAGE:        store the drug classifications
ROOT:         antibiotics
SECURITY:     UNCLASSIFIED
CATEGORY:     (READ)

AUTHOR:       Newton S. Lee
DATE:         January 1985

CONTENTS:

                  ANTIBIOTICS
                 /            \
     PENICILLINS               CEPHALOSPORINS
    /           \             /             \
ORIGINAL     BETA-         1ST-          2ND-
PENICILLINS  LACTAMASE     GENERATION    GENERATION
             PENICILLINS   CEPHAL.       CEPHAL.
```

Figure 36: Drug Classification Tree

11.4.3 Semantic Nets

Causal models are best represented by some directed graphs. Figure 37 shows a causal model for the cause-effect relationships among concentration, clearance, blood flow and protein-binding.

11.4.4 Knowledge Frames

Information about the properties of a specific drug or a class of drugs is a good candidate for the knowledge frame representation. A knowledge frame fills in its slot by drawing information from relational tables, hierarchical trees, and semantic nets. Figure 38 shows a drug properties knowledge frame.

11.4.5 Action Frames

The use of AND/OR goal trees is an effective way to solve a decomposable problem. One of the major functions of DIES-II is to predict all possible interactions between two given drugs. The problem can be broken down into determining many different types of interactions such as acid-base reaction, protein-bound interaction, nephrotoxic interaction and so forth. Figure 39 shows a partial AND/OR goal tree for

```
GENERAL NET:   concentration
USAGE:         clearance, blood flow and protein-
               binding relationships
SECURITY:      UNCLASSIFIED
CATEGORY:      (READ)

AUTHOR:        Newton S. Lee
DATE:          January 1985

CONTENTS:

            CONCENTRATION
                  |
                  | inversely_proportional_to
                  v
            TOTAL CLEARANCE
                  |
        ---------------------------
        |                          |
        | proportional_to          | proportional_to
        |                          |
        v                          v
    HEPATIC CLEARANCE        RENAL CLEARANCE
            |
            | proportional_to
            |
            v
          BLOOD FLOW
```

Figure 37: A Causal Model

KF NAME: properties of a drug
USAGE: find all the properties of a drug
LEVEL#: Any

AUTHOR: Newton S. Lee
DATE: January 1985

SLOTS	DEFAULT VALUES	METHODS OF RETRIEVAL
1. Acidity	unknown	tbl,ask,dval
2. pKa	unknown	tbl,ask,dval
3. Water Solubility	50%	tbl,ask,dval
4. Nephrotoxicity	unknown	tbl,ask,dval
5. Protein Binding	50%	tbl,ask,dval
6. Properties towards Microsomal Enzymes	unknown	tbl,ask,dval

LEGENDS:

tbl = look up information from relational tables
ask = ask the user for information
dval = take on the default value

Figure 38: Drug Properties Knowledge Frame

determining all the possible interactions between two given drugs.

In order to predict all the possible drug interactions, the system first obtains the properties of the drugs, then determines the possible interactions based on the known properties, and finally summarizes the findings. The tough problem is to determine all the possible interactions based on the drug properties. The problem is broken down into many subgoals. Some of the subgoals are, as shown in Figure 39, to determine acid-base interaction, to determine nephrotoxic interaction, and to determine protein-bound interaction.

Figure 40 shows the action frame representations of the top-level goal "determine interactions of two given drugs" and the subgoal "determine acid-base interaction."

11.4.6 Production Rules

The action frame "Determine Acid-Base Interaction" invokes a production rule to compute the amount of neutralization of an acidic drug and a basic drug. Figure 41 shows a neutralization rule.

```
                 Determine Interactions
                 of Two Given Drugs
                   /         |         \
                  /  <------AND----->   \
                 /          |           \
                /           |            \
   Obtain Properties  Determine        Report
   of the Drugs       Interactions     all
                      Based on         Interactions
                      Properties
                     /      |      \
                    /  <---AND--->   \
                   /       |          \
                  /        |           \
          Determine    Determine    Determine
          Acid-Base    Nephrotoxic  Protein-Bound
          Interaction  Interaction  Interaction
```

Figure 39: An AND/OR Goal Tree to Determine Interactions

AF NAME: determine interactions of two given drugs
USAGE: determine all possible interactions
LEVEL#: 1
ACTION TYPE: AND
SUCCESS COND: there are some interactions
FAILURE COND: none

AUTHOR: Newton S. Lee
DATE: January 1985

ACTIONS	METHODS OF ACHIEVEMENT
1. Obtain Properties of the Drugs	Instantiate Knowledge Frame "Properties of a Drug"
2. Determine possible interactions	Invoke Action Frame "Interactions Based on Drug Properties"
3. Report all interactions	Invoke Action Frame "Report All Interactions"

AF NAME: determine acid-base interactions
USAGE: determine acid-base interactions
LEVEL#: 3
ACTION TYPE: AND
SUCCESS COND: there is acid-base interaction
FAILURE COND: none

AUTHOR: Newton S. Lee
DATE: January 1985

ACTIONS	METHODS OF ACHIEVEMENT
1. Obtain properties of the drugs	Look up information from the blackboard
2. See if they are acid-base pair	Invoke library routine SAME_set
3. Store findings	Write information on the blackboard
4. Determine amount of neutralization	Invoke a production rule to compute the amount of neutralization

Figure 40: Action Frames for Predicting Drug Interactions

```
RULE NAME:    neutralization rule
USAGE:        to compute the amount of neutralization
              based on the pKa's of two drugs
LEVEL#:       Any
AUTHOR:       Newton S. Lee
DATE:         January 1985

ASSUMPTIONS (IF CLAUSES):
1. The two drugs are in equimolar concentrations in
   the blood plasma
2. The body pH of the patient is approximately 7.4

CALCULATIONS (THEN CLAUSES):
  Drug 1 pH = pKa1
  Drug 2 pH = pKa2
  Take difference of each from 7.4
  Drug with pH nearest 7.4 will be completely
  neutralized
  Drug with pH farthest from 7.4 will suffer
  loss of activity
  [ 7.4 - abs(differences) ] / 7.4  * 100%
```

Figure 41: Neutralization Rule

11.4.7 Natural Language Capability

The system should be able to understand simple queries such as "what are the possible interactions between drug_A and drug_B?" and "what are the properties of drug_A?." To make use of the built-in natural language capability, we have to recall how an input string is processed by the system and that the processed string is sent to a set of syntax-semantic action routines (re-read the chapter on "Human Engineering" if necessary). The first step is to identify the following entities:

1. Keywords - Words that are of significance in the meaning of a query. For example: the word 'go' in the query 'where did you go?' The keywords are also known as domain-specific words.
2. Key Objects - Objects that assume important roles in a query. They are the focus of attention. For example: the object 'John' in the query 'when did John go?' The objects are also known as objects of interest.
3. Non-keywords - Words that are of no significance in the meaning of a query and can thus be discarded. For example: the words 'are' and 'about' in the query 'what are you talking about?'

4. Synonyms - words that are of the same or of similar meaning. For example: the word 'speak' is a synonym of 'talking' in the query 'why are you talking about?' The synonyms are also known as the words class names.

Figure 42 are some of the synonyms and non-keywords for DIES-II.

Having identified these entities, the second step is to design a set of syntax-semantic action maps (SSA_maps). The syntax portion of a SSA_map is a list of keywords and key objects. The keywords are fixed atoms but the key objects are usually variables. The semantic action portion of a SSA_map can invoke any frames, production rules and library routines. Figure 43 shows the SSA_maps along with some example queries.

11.4.8 Menu-Driven Capability

The alternative to the natural language interface is the menu-driven capability. Two steps are involved. First we need to set up a display menu, and next we need to specify what actions are to be performed for each option described in the menu. Figure 44 shows an example of a display menu and the actions that correspond to each option.

SYNONYMS

Interaction	=	Interactions, Reactions, Reaction
Properties	=	Property, Characteristics, Characteristic
Class	=	Classes, Type, Types, Kind, Kinds
Subclass	=	Subclasses, Subtypes, Subtype, Example, Examples
Actions	=	Action, Procedure, Procedures

NON-KEYWORDS

What, Is, Are, The, Of, Between, And, Drug, Drugs, Which, Do, Does, Belong, Belongs, To

Figure 42: Synonyms and Non-Keywords

```
SYNTAX:  (CLASS  *A)
ACTION:  Activate AF("determine class of drug")
QUERY :  What class of drugs does *A belong to?

SYNTAX:  (CORRECTIVE ACTIONS)
ACTION:  Activate AF("find corrective actions")
QUERY :  What are the corrective actions?

SYNTAX:  (INTERACTION  *A  *B)
ACTION:  Activate AF("determine drug interactions")
QUERY :  What are the interactions of *A and *B?

SYNTAX:  (PROPERTIES  *A)
ACTION:  Activate AF("find properties of a drug")
QUERY :  What are the properties of *A?

SYNTAX:  (SUBCLASS  *A)
ACTION:  Activate AF("determine subclass of drug")
QUERY :  What are the subclasses of *A?

Legend:  *A = name of a drug
         AF = action frame
```

Figure 43: Syntax-Semantic Maps

```
MENU:

1] Interactions between two drugs
2] Corrective actions for adverse interactions
3] Properties of a drug
4] Classification of a drug
5] Subclass/examples of a drug

WHICH?

Option     Actions
------     -------
  1        Ask for the names of the two drugs and
           activate AF("determine class of drug")
  2        Activate AF("find corrective actions")
  3        Ask for the name of the drug and activate
           AF("determine properties of a drug")
  4        Ask for the name of the drug and activate
           AF("determine classification of a drug")
  5        Ask for the name of the drug and activate
           AF("determine subclasses of a drug")
```

Figure 44: Display Menu and Corresponding Actions

11.4.9 User Authorizations

Since DIES-II is for general use, no user authorization is required. If you are not designing an expert system that requires security protections, you may skip the following material.

For the moment, let's pretend that we are working on a classified project. A user authorization file must be present in order to give the appropriate types of access to the specific users. The file contains the identifications, the passwords, the clearances, and the need-to-know sets for all the users of the classified system. A user must supply his/her correct identification and password in order to logon to the system. Figure 45 shows two records in a user authorization file and following it in Figure 46 is a typical login procedure to a classified system.

11.5 PROGRAMMED MODEL

Having constructed a reasonable communicative model for DIES-II, the remaining task is to program the model so that it runs on a computer and produces results for us. The task of "translating" a communicative model to a programmed model becomes relatively easy with the use of GUESS/1. The following sections present the implementation of DIES-II.

Identification	Password	Clearance	Need-to-Know
System	VPI&SU	TOP_SECRET (READ WRITE)	EVERYTHING
User	VPI	UNCLASSIFIED (READ)	TABLE_A TREE_X FRAME_2

Figure 45: A User Authorization File

```
[ Responses typed by the user are underlined.  Note
  that the password typed by the user is not echoed. ]

Welcome to Virginia Tech GUESS/1
: help

Legal commands at this point are:

  login   - initiate a new session
  logout  - break the connection
  time    - show time
  user    - show user

Any problem or question should be addressed
to Newton Lee, login NEWTON at node VPICS1,
address VPI&SU, tel. 961-5853

: login

Your name: newton
Identification code: user
Password:

Application Ready
DATETIME = 10-JAN-1985 21:23:09

Welcome to the Classified Expert System
```

Figure 46: A Login Procedure

11.5.1 Relational Tables

Figure 47 shows the internal representation of relational tables in GUESS/1. Following it in Figure 48 is the drug nephrotoxicity table.

11.5.2 Hierarchical Trees

Figure 49 shows the internal representation of hierarchical trees. Following it in Figure 50 is the drug classification tree.

11.5.3 Semantic Nets

Figure 51 shows the internal representation of semantic nets. Following it in Figure 52 is a portion of a causal model implemented using semantic nets.

11.5.4 Knowledge Frames

Figure 53 shows the internal representation of knowledge frames. Following it in Figure 54 is the drug properties frame.

```
(TABLE <table name> <security-level> <category>
       (    (<key>  <value>)
            (<key>  <value>)
             ....
       )
)
```

Figure 47: Internal Representation of Relational Tables

```
TABLE NAME:    nephrotoxicity of a drug
USAGE:         store the nephrotoxicity of drugs
KEY:           drug name
VALUE:         yes / no / unknown
SECURITY:      UNCLASSIFIED
CATEGORY:      (READ)

AUTHOR:        Newton S. Lee
DATE:          January 1985

(  (TABLE "nephrotoxicity" UNCLASSIFIED (READ)
          (
                (AMINOGLYCOSIDES      yes)
                (AMPHOTERICIN-B       unknown)
                (ANTIBIOTICS          unknown)
                (ANTIFUNGALS          yes)
                (CEPHALOSPORINS       no)
                (CHLORAMPHENICOL      yes)
                (LINCOSAMIDES         yes)
                (MACROLIDES           yes)
                (PENICILLINS          unknown)
                (POLYMYXINS           yes)
                (TETRACYCLINES        yes)
          )
)  )
```

Figure 48: Implementation of Drug Nephrotoxicity Table

```
(TREE <tree name> <security-level> <category>
      (  (<parent node>
                <children node> <tree id>
                ... )
         (<parent node>
                <children node> <tree id>
                ... )
         ...
         (<leaf node>
                nil)
         ...
      )
)
```

<u>Where</u>

<tree id> = "internal", if children node is on the same tree.
<tree id> = <tree name> of the other tree, if children node is on the other tree.

Figure 49: Internal Representation of Hierarchical Trees

```
TREE NAME:    drug classification
USAGE:        store the drug tree
ROOT:         antibiotics
SECURITY:     UNCLASSIFIED
CATEGORY:     (READ)

AUTHOR:       Newton S. Lee
DATE:         January 1985

(  (TREE "drug classification" UNCLASSIFIED (READ)
      (
        (ANTIBIOTICS
             PENICILLINS internal
             CEPHALOSPORINS internal)
        (PENICILLINS
             ORIGINAL-PENICILLINS internal
             BETA-LACTAMASE-PENICILLINS internal)
        (CEPHALOSPORINS
             1ST-GENERATION-CEPHALOSPORINS internal
             2ND-GENERATION-CEPHALOSPORINS internal)
        (ORIGINAL-PENICILLINS
             nil)
        (BETA-LACTAMASE-PENICILLINS
             nil)
        (1ST-GENERATION-CEPHALOSPORINS
             nil)
        (2ND-GENERATION-CEPHALOSPORINS
             nil)
      )
)  )

NOTE: "nil" signifies "leave node"
```

Figure 50: Drug Classification Tree

For a directed graph:

```
(GNET  <net name>  <security level>  <category>
       (  ( <node1>  <relationship>  <node2> )
          ( <node1>  <relationship>  <node2> )
          ...
       )
)
```

For a non-directed graph:

```
(SNET  <net name>  <security level>  <category>
       (  ( <node1>  <node2> )
          ( <node1>  <node2> )
          ...
       )
)
```

Figure 51: Internal Representation of Semantic Nets

```
NET NAME:      concentration
USAGE:         clearance, blood flow and protein-
               binding relationships
SECURITY:      UNCLASSIFIED
CATEGORY:      (READ)

AUTHOR:        Newton S. Lee
DATE:          January 1985

Note:          A --(is_related_to)--> B
               is encoded as
               (A  is_related_to  B)


(  (GNET "concentration"  UNCLASSIFIED  (READ)
   (
   (concentration  inverse_to  total_clearance)
   (total_clearance proportional_to hepatic_clearance)
   (total_clearance  proportional_to  renal_clearance)
   (hepatic_clearance  proportional_to  blood_flow)
   (renal_clearance  proportional_to  filtration)
   (filtration  proportional_to  blood_flow)
   (blood_flow  inverse_to  vascular_resistance)
   (vascular_resistance  inverse_to vessel_size)
   (vessel_size  proportional_to  temperature)
   (vessel_size  proportional_to  dilation_agents)
   )  )  )
```

Figure 52: A Causal Model

```
(KF <KF_name>)
    ->   (USAGE <usage description> LEVEL <level#>)

         <KF body>
         ...
         ...
```

Figure 53: Internal Representation of Knowledge Frames

```
KF NAME:          properties of a drug
USAGE:            find all the properties of a drug
LEVEL#:           Any

AUTHOR:           Newton S. Lee
DATE:             January 1985

(  (KF "properties of a drug")
     ->
       (USAGE "find out the properties of a drug"
        LEVEL ANY)
       (GETCURRENT  OBJECT_OF_INTEREST  *a)
       (or  (TABLE_LOOKUP
                TABLE IS "acidity of a drug"
                KEY   IS *a
                VALUE IS *a_a)
            (ASK_USER
                QUESTION IS "what is the acidity of"
                OBJECT   IS *a
                EXPECT   (acid base unknown)
                USELESS  unknown
                ANSWER   IS *a_a)
            (DEFAULT_OF  *a_a  IS  unknown)
       )
       (cut)
       (or  (TABLE_LOOKUP
                TABLE IS "pKa of a drug"
                KEY   IS *a
                VALUE IS *p_a)
            (ASK_USER
                QUESTION IS "what is the pKa of"
                OBJECT   IS *a
                EXPECT   RANGE (GE 1.0 LE 14.0)
                USELESS  unknown
                ANSWER   IS *p_a)
            (DEFAULT_OF  *p_a  IS  unknown)
       )
       (cut)
       (BLACKBOARD_UPDATE
           KF      IS  "properties of a drug"
           OBJECT  IS  *a
           CONTENT IS  ( (acidity  *a_a) (pKa  *p_a) )
       )  )
```

Figure 54: A Drug Properties Frame

11.5.5 Action Frames

Figure 55 shows the internal representation of action frames. Following it in Figure 56 and 57 are the action frames "determine interactions of two given drugs" and "determine acid-base interactions."

11.5.6 Production Rules

Figure 58 shows the three basic internal representations of production rules. The choice of an appropriate representation depend on your application. Following it in Figure 59 and 60 are the neutralization rules.

11.5.7 Natural Language Capability

Figure 61 shows the internal representations of synonyms and non-keywords declarations. Following it in Figure 62 are the declarations in DIES-II.

Figure 63 is the internal representation of SSA_maps. Following it in Figure 64 are some SSA_map routines in DIES-II. A "No Match" SSA_map for handling the unrecognizable queries should be present at the very bottom of all the

```
(AF <AF_name>)
    ->
          (USAGE <usage description> LEVEL <level#>)

          <AF body>
          ...
          ...
```

Figure 55: Internal Representation of Action Frames

```
AF NAME:          determine interactions of two given
                  drugs
USAGE:            determine all possible interactions
LEVEL#:           1
ACTION TYPE:      AND
SUCCESS COND:     there are interactions
FAILURE COND:     none

AUTHOR:           Newton S. Lee
DATE:             January 1985

(  (AF "determine interaction of drug A and B")
      ->
     (USAGE "to determine interactions between
             two given drugs"  LEVEL 1)
     (GETFIRST  OBJECT_OF_INTEREST  *a)
     (KF "properties of a drug")
     (GETNEXT  OBJECT_OF_INTEREST  *b)
     (KF "properties of a drug")
     (AF "determine interactions based on properties")
     (AF "report all possible interactions")
     (cut)  )
```

Figure 56: An Action Frame to Determine Possible Interactions

```
AF NAME:          acid-base interactions
USAGE:            determine acid-base interactions
LEVEL#:           3
ACTION TYPE:      AND
SUCCESS COND:     there is acid-base interaction
FAILURE COND:     none

AUTHOR:           Newton S. Lee
DATE:             January 1985

(  (AF "acid-base interaction")
      ->
     (USAGE "determine acid-base interaction" LEVEL 3)
     (GETFIRST  OBJECT_OF_INTEREST  *a)
     (GETNEXT  OBJECT_OF_INTEREST  *b)
     (BLACKBOARD_LOOKUP
          KF     IS  "properties of a drug"
          OBJECT IS  *a
          KEY    IS  acidity
          VALUE  IS  *v_a)
     (BLACKBOARD_LOOKUP
          KF     IS  "properties of a drug"
          OBJECT IS  *b
          KEY    IS  acidity
          VALUE  IS  *v_b)
     (SAME_SET  (*v_a  *v_b)  (acid  base))
     (BLACKBOARD_UPDATE
          AF      IS  "acid-base interaction"
          CONTENT IS  ( (interaction  acid-base) ))
     (BLACKBOARD_LOOKUP
          KF     IS  "properties of a drug"
          OBJECT IS  *a
          KEY    IS  pKa
          VALUE  IS  *pKa1)
     (BLACKBOARD_LOOKUP
          KF     IS  "properties of a drug"
          OBJECT IS  *b
          KEY    IS  pKa
          VALUE  IS  *pKa2)
     (PR  "compute amount of neutralization"
          *a *pKa1 *b *pKa2)
     (cut)  )
```

Figure 57: An Action Frame to Determine Acid-Base Interaction

```
; IF clause
(PR  <rule identification>  <variable parameters>)
   ->
          ; THEN clauses
          <function body>
          .....
)


; IF clause
(<rule header>  <variable parameters>)
   ->
          ; THEN clauses
          <rule body>
          .....
)


; Rule Identification
(<rule header>  <variable parameters>)
   ->
          ; IF clauses
          <conditions>
          .....
          ; THEN clauses
          <actions>
)
```

Figure 58: Internal Representations of Production Rules

```
RULE NAME:    neutralization rule
USAGE:        to compute the amount of neutralization
              based on the pKa's of two drugs
LEVEL#:       Any
AUTHOR:       Newton S. Lee
DATE:         January 1985

ASSUMPTIONS:
1. The two drugs are in equimolar concentrations in
   the blood plasma
2. The body pH of the patient is approximately 7.4

CALCULATIONS:
  Drug 1 pH = pKa1
  Drug 2 pH = pKa2
  Take difference of each from 7.4
  Drug with pH nearest 7.4 will be completely
  neutralized
  Drug with pH farthest from 7.4 will suffer
  loss of activity
  [ 7.4 - abs(differences) ] / 7.4  * 100%

(  (PR  "compute amount of neutralization"
         *drug1  *pKa1  *drug2  *pKa2)
      ->
     (USAGE  "compute amount of neutralization"
      LEVEL  ANY)
     (NUMERIC  *pKa1)
     (NUMERIC  *pKa2)
     (!  (and (<  *pKa1  7.4) (<  *pKa2  7.4)) )
     (!  (and (>  *pKa1  7.4) (>  *pKa2  7.4)) )
     (==  *body-pH  7.4)
     (==  *pH1  *pKa1)
     (==  *pH2  *pKa2)
     (:=  *d1  (abs  (-  *body-pH  *pH1)))
     (:=  *d2  (abs  (-  *body-pH  *pH2)))
     ....
     ....  )
```

Figure 59: Neutralization Rule

```
(  (PR  "compute amount of neutralization"
        *drug1  *pKa1  *drug2  *pKa2)
     ->
    (NUMERIC  *pKa1)
    (NUMERIC  *pKa2)
    (or     ((<  *pKa1  7.4) (<  *pKa2  7.4))
            ((>  *pKa1  7.4) (>  *pKa2  7.4))
    )
    (print  "No neutralization.")  )


(  (PR  "compute amount of neutralization"
        *drug1  *pKa1  *drug2  *pKa2)
     ->
    (or     (! (NUMERIC  *pKa2))
            (! (NUMERIC  *pKa1))
    )
    (print  "Unable to compute.")  )
```

Figure 60: Neutralization Rule (continued)

```
(SYNONYM
     (    ( <word1>   <synonym11> .... <synonym1n> )
          ( <word2>   <synonym21> .... <synonym2n> )
          ....
     )
)

(NONKEYWORDS
     ( <word1> <word2> .... <wordn> )
)
```

Figure 61: Internal Representations of Synonym and Non-Keyword Declarations

```
(  (SYNONYM
      (  (INTERACTION     INTERACTIONS REACTIONS
                          REACTION)
         (PROPERTIES      PROPERTY CHARACTERISTICS
                          CHARACTERISTIC)
         (CLASS           CLASSES TYPE TYPES KIND
                          KINDS)
         (SUBCLASS        SUBCLASSES SUBTYPE SUBTYPES
                          EXAMPLE EXAMPLES)
         (ACTIONS         ACTION PROCEDURE PROCEDURES)
      )
)  )


(  (NONKEYWORDS
      (WHAT IS ARE THE OF BETWEEN AND
       DRUG DRUGS
       WHICH DO DOES BELONG BELONGS TO)
)  )
```

Figure 62: Synonym and Non-Keyword Declarations

regular SSA_map routines. Figure 65 shows a typical "No Match" SSA_map.

11.5.8 Menu-Driven Facility

Figure 66 shows the internal representation of a display menu. Following it in Figure 67 is the display menu for DIES-II. Each option described in the menu must be associated with some actions. Figure 68 shows the internal representation of option/action rules. Following it in Figure 69 are two option/action rules in DIES-II.

11.5.9 User Authorizations

Again since DIES-II is for general use, no user authorization is required. If you are not designing an expert system that requires security protections, you may skip the following material.

Figure 70 shows the internal representations for the user identifications, passwords, clearances and the need-to-know sets. Following it in Figure 71 are some examples.

```
(  (SSA_MAP  <syntax>)
           ->   <semantic actions>  )
```

Figure 63: Internal Representation of SSA_maps

```
(SSA_MAP  (CLASS  *A))

! answer questions about the classification of
! a given drug

Input Parameter:   *A - a drug name

(  (SSA_MAP  (CLASS  *A))
      ->
          (cut)
          (OBJECTLIST_ERASE)
          (ADD  OBJECT_OF_INTEREST  *A)
          (AF  "determine class of drug")  )


(SSA_MAP  (INTERACTION  *A  *B))

! answer questions about the all possible
! interactions of two given drugs

Input Parameters:   *A - drug A
                    *B - drug B

(  (SSA_MAP  (INTERACTION  *A  *B))
      ->
          (cut)
          (BLACKBOARD_ERASE)
          (OBJECTLIST_ERASE)
          (ADD  OBJECT_OF_INTEREST  *A)
          (ADD  OBJECT_OF_INTEREST  *B)
          (AF  "determine drug interaction")  )
```

Figure 64: SSA_maps

```
(SSA_MAP  *question)

! no match

Input Parameter:   *question - user question

(  (SSA_MAP  *question)
      ->
     (cut)
     (print "Sorry.
             I don't understand your question.")  )
```

Figure 65: A Typical "No Match" SSA_map

```
(MENU IS "<option 1> : <description 1>
          <option 2> : <description 2>
          .
          .
          <option n> : <description n>")
```

Figure 66: Internal Representation of a Menu

```
(  (MENU IS

" 1] Interaction between two drugs
 2] Corrective actions for adverse interactions
 3] Properties of a drug
 4] Classification of a drug
 5] Subclass/examples of a drug")  )
```

Figure 67: A Display Menu

```
(   (MENU "<option #>")
          ->    <actions>)
```

Figure 68: Internal Representation of Option/Action Rules

```
(   (MENU "1")
        ->  (ASK_USER  QUESTION IS "what is drug A"
                       OBJECT   IS nil
                       EXPECT   anything
                       USELESS  nil
                       ANSWER   *A)
            (ASK_USER  QUESTION IS "what is drug B"
                       OBJECT   IS nil
                       EXPECT   anything
                       USELESS  nil
                       ANSWER   *B)
            (CHECK_WORD  *A  *D1)
            (CHECK_WORD  *B  *D2)
            (SSA_MAP  (INTERACTION  *D1  *D2))  )

(   (MENU "2")
        ->  (SSA_MAP  (CORRECTIVE ACTIONS))  )
```

Figure 69: Some Option/Action Rules

```
(PASSWORD_OF  <identification code> IS  <password>)
(CLEARANCE_OF  <identification code> IS  <clearance>)
(NEEDTOKNOW_OF  <identification code> IS  <objects>)

where

<identification code>
                = any string of characters
<password>      = any string of characters
<clearance>     = ( <security_level>  <category> )
<objects>       = ( <obj1> <obj2> .... )

and

<security_levels>
                = TOP_SECRET, SECRET, CONFIDENTIAL,
                  or UNCLASSIFIED

<category>      = (READ WRITE), (READ), or (WRITE)

<obji>          = the name of an object
                  (eg. of a relational table)
```

Figure 70: Internal Representations of a User Authorization File

```
! System Manager:

(  (PASSWORD_OF   system  IS
                  VPI&SU)  )
(  (CLEARANCE_OF  system  IS
                  (TOP_SECRET (READ WRITE)))  )
(  (NEEDTOKNOW_OF system  IS
                  EVERYTHING)  )

! User_A:

(  (PASSWORD_OF   user_A    IS
                  VPI)  )
(  (CLEARANCE_OF  user_A    IS
                  (UNCLASSIFIED  (READ)))  )
(  (NEEDTOKNOW_OF user_A    IS
                  ("TABLE A" "TREE X" "FRAME 2")  )
```

Figure 71: A User Authorization File

11.5.10 Filling in Other Details

The major tasks in building the DIES-II have been discussed in length, what is left undone are some minor details for completing the system. These are described here.

11.5.10.1 Application System Identification

The expert system identification is displayed whenever the user first invokes the system. Figure 72 shows how to set up a proper system identification.

11.5.10.2 Activation of the Application System

In order for GUESS/1 to activate your expert system, the code shown in Figure 73 is included in the program APP.APP that comes with the GUESS/1 package.

11.5.10.3 Help Messages

Help messages should generally be included in every application system. Figure 74 shows the internal representation of a HELP rule. Following it in 75 is the help messages for DIES-II.

Syntax:

```
(APPLICATION NAME IS <identification>)
```

Example:

```
( (APPLICATION NAME IS
          "The Drug Interaction Expert System")  )
```

Figure 72: Expert System Identification

```
;
; (APPLICATION)
;
; Activate Application System
;

(  (APPLICATION)
        ->
        (GET_QUESTION  *question)
        (SSA_MAP  *question)  )
```

Figure 73: Code to Activate the Application System

```
( (HELP)  ->  <help messages> )
```

Figure 74: Internal Representation of a HELP Rule

```
(   (HELP)
     ->
    (cut)
    (print " ** THE DRUG INTERACTION EXPERT SYSTEM **")
    (print "")
    (print " This system is capable of answering"
    (print " questions of the following types:")
    (print "")
    (print " 1) What are the interactions between
    (print "    drug A and drug B")
    (print " 2) What are the corrective actions")
    (print " 3) What are the properties of A")
    (print " 4) What class of drug does A belong to")
    (print " 5) What are the subclasses of A")
    ....
    ....
```

Figure 75: HELP Messages

11.6 EXPERIMENTAL MODEL

There is no special condition under which DIES-II is to be observed or experimented with. The system is still in the experimental stage. Therefore, the experimental model is more or less the same as the programmed model.

If you are developing an experimental model for your expert system, the first step is to define a set of conditions under which the system is to be observed or experimented with, and the second step is to state how the observation or experimentation is to be carried out. Please refer to [Balci, 1983].

11.7 MODEL RESULTS

DIES-II currently contains 72 drug classes, each with 1 to 3 examples for a current total of 155 specific drugs. Extensive information is available on the characteristics of eight drugs: aspirin, diphenhydramine, penicillin, sulfisoxazole, atropine, chlorpromazine, morphine and digoxin. Figure 76 shows an interactive log session of the execution of DIES-II. Responses typed by the user are underlined. Commentaries by the author are enclosed in square brackets.

```
Application Ready
DATETIME = 10-JAN-1985 20:35:46

Welcome to the Drug Interaction Expert System

% help

 *** THE DRUG INTERACTION EXPERT SYSTEM ***

 This system is capable of answering questions of the following type

 1) What are the interactions between A and B
 2) What are the corrective actions
 3) What are the properties of A
 4) What class of drug does A belong to
 5) What are the subclasses of A

 Examples:
 1) What are the interactions between antifungals and tetracyclines?
 2) What are the corrective actions?
 3) What are the properties of antifungals?
 4) What class of drug does oxycycline belong to?
 5) What are the subclasses of tetracyclines?

 To see how the system comes up with the answer, type TRACE at %
 prompt, follow it by a query or REPEAT command.  NOTRACE can be
 used to disable TRACE.

 ------------------------------------------------------------
 | System Commands:                                         |
 |                                                          |
 | CLEAR          -  clear the screen                       |
 | CPU            -  show elapsed cpu time                  |
 | <CTRL-Z>       -  stop the system                        |
 | ECT            -  show elapsed clock time                |
 | MAIL           -  use VAX/VMS mailing system             |
 | MSG            -  set system messages on                 |
 | NOMSG          -  set system messages off                |
 | NOTRACE        -  set problem solving trace off          |
 | REPEAT         -  repeat last query                      |
 | RN             -  generate a random number               |
 | SAY            -  say a message to a user                |
 | SPELL <word>   -  confirm the spelling of a word         |
 | STATUS         -  show current status (MSG, TRACE)       |
 | STOP           -  stop the system                        |
 | TIME           -  show current date and time             |
 | TRACE          -  set problem solving trace on           |
 | USER           -  show interactive users on system       |
 ------------------------------------------------------------
```

```
% clear

% What are the interactions between antifungals and tetracyclines?

TETRACYCLINES loses 100% of its activity
ANTIFUNGALS predominates and loses 91.89% of its activity
** INTERACTION(1) ** acid-base
** INTERACTION(2) ** nephrotoxic
** INTERACTION(3) ** protein-bound

[ The system reports all possible interactions. ]

% trace
OK

% repeat

Q: WHAT ARE THE INTERACTIONS BETWEEN ANTIFUNGALS AND TETRACYCLINES
[1] |   to determine interactions between two given drugs
[1] |   find out the properties of a given drug
[1] |   find out the properties of a given drug
[2] |           to determine interactions based on drug's properties
[3] |                   determine acid-base interaction
[3] |                   compute amount of neutralization
TETRACYCLINES loses 100% of its activity
ANTIFUNGALS predominates and loses 91.89% of its activity
[3] |                   determine nephrotoxic interaction
[3] |                   determine protein-bound interaction
[3] |                   determine microsomal interaction
[3] |                   determine microsomal2 interaction
[2] |           report all possible interactions

[ The system displays the trace while it is working along an AND/OR
  goal tree for determining all possible interactions between two
  given drugs. ]

** INTERACTION(1) ** acid-base
** INTERACTION(2) ** nephrotoxic
** INTERACTION(3) ** protein-bound

% notrace
OK

% what are the corrective actions?

Interaction(1): acid-base
Corrective action(1): separate time of administration
Interaction(2): nephrotoxic
Corrective action(2): withdraw one drug
Interaction(3): protein-bound
```

Corrective action(3): lower dosage of one or both drugs

[The system suggests corrective actions for adverse interactions.]

% What are the properties of antifungals?

acidity acid
pKa 4
solubility 80
nephrotoxicity yes
protein-bound 60
microsomal-enzymes inhibitor

% What class of drugs does oxycycline belong to?

CLASS: TETRACYCLINES

% What are the subclasses of tetracyclines?

SUBCLASS:
(1) TETRACYCLINE
(2) OXYCYCLINE
(3) DOXYCYCLINE

% What are the interactions between antifunga and tetracylines?

ANTIFUNGA = ANTIFUNGAL
TETRACYLINES = TETRACYCLINES

[The system corrects misspelled words.]

TETRACYCLINES loses 100% of its activity
ANTIFUNGAL predominates and loses 91.89% of its activity
** INTERACTION(1) ** acid-base
** INTERACTION(2) ** nephrotoxic
** INTERACTION(3) ** protein-bound

% what are the interactions between ampicillin and tetracyclines?

QUESTION: what is the pKa of AMPICILLIN?
CHOOSE : 1 < = ans < = 14 OR ans = unknown
ANSWER ? 1.23

QUESTION: what is the solubility of AMPICILLIN?
CHOOSE : 0 < = ans < = 100 OR ans = unknown
ANSWER ? 85

QUESTION: what is the % protein-bound of AMPICILLIN?
CHOOSE : 0 < = ans < = 100 OR ans = unknown
ANSWER ? 85

[The system asks the user for information on drug properties when it cannot find those information from its knowledge base.]

```
TETRACYCLINES loses 100% of its activity
AMPICILLIN predominates and loses 54.46% of its activity
** INTERACTION(1) ** acid-base
** INTERACTION(2) ** nephrotoxic
** INTERACTION(3) ** protein-bound
** INTERACTION(4) ** bleeding

% what are the corrective actions?

Interaction(1): acid-base
Corrective action(1): separate time of administration
Interaction(2): nephrotoxic
Corrective action(2): withdraw one drug
Interaction(3): protein-bound
Corrective action(3): lower dosage of one or both drugs
Interaction(4): bleeding
Corrective action(4): withdraw both drugs

% stop

DATETIME = 10-JAN-1985 20:42:38
ELAPSED TIME = 00:06:52
CPU = 00:01:02
TOTAL NUMBER OF QUERIES = 8
TOTAL NUMBER OF SYSCMD CALLS = 6
```

Figure 76: Interactive Session on the Drug Interaction Expert

Chapter XII

INSTALLING A NEW EXPERT SYSTEM ON GUESS/1

The horse is made ready for the
day of battle,
but victory rests with the Lord.

- Solomon, PROVERBS

All the programs, command files and data files mentioned in this chapter are located in CS:[NEWTON.GUESSPACK] on VAX1. Please copy all the files from [NEWTON.GUESSPACK] to your directory by issuing the VAX/VMS command $ copy cs:[newton.guesspack]*.* *.*

Five steps to install a new expert system on GUESS/1:

1. Follow the guidelines in the chapter on "Application: The Drug Interaction Expert" to develop your new expert system.
2. Follow the instructions in section 12.1 "Application Expert System Routines Distribution."
3. Follow the instructions in section 12.2 "Creating a GUESS/1 Interpreter With a New Domain-Specific Dictionary."

4. Execute the program COMPILE.COM by issuing the VAX/VMS command

   $ @compile

   It compiles your expert system code. Your expert system is now residing in the file GUESS.MOD in your directory.

5. To run your new expert system, simply execute the program GUESS.COM by issuing the VAX/VMS command

   $ @guess

NOTE: If you modify your expert system, you have to redo step 4. If you modify your dictionary, you have to redo steps 3 and 4.

12.1 APPLICATION EXPERT SYSTEM ROUTINES DISTRIBUTION

Your application expert system routines must be grouped into 13 functional categories as shown in Figure 77

File Name	Functional Group
APP.AF	Action Frames
APP.APP	Application Identification
APP.HC	Additional Features to GUESS/1
APP.HLP	Help Messages
APP.KF	Knowledge Frames
APP.MEN	Menu-Driven Facility
APP.NET	Semantic Nets
APP.PR	Production Rules
APP.SSA	Syntax-Semantic Action Maps
APP.SYN	Synonyms and Non-Keywords Definitions
APP.TBL	Relational Tables
APP.TRE	Hierarchical Trees
APP.USE	User Authorization File

Figure 77: Application System Routines Distribution

12.2 CREATING A GUESS/1 INTERPRETER WITH A NEW DOMAIN-SPECIFIC DICTIONARY

Six steps to create a new GUESS/1 interpreter that contains a new domain-specific dictionary for your expert system:

1. Create a text file named WORD.DAT which contains all the domain-specific words in UPPERCASE, one word per line. Each word must begin in column 1 and should not exceed 25 characters.
2. Append to the text file WORD.DAT the following system commands (one command per line, in UPPERCASE):

   CLEAR, CPU, ECT, HELP, MAIL, MENU, MSG, NOMSG, NOTRACE, REPEAT, RN, SAY, SPELL, STATUS, STOP, TIME, TRACE, USER
3. Sort the text file WORD.DAT by issuing the VAX/VMS command

   ```
   $ sort /key=(position:1,size:25) word.dat word.dat
   ```
4. Execute the program MKDAT.EXE by issuing the VAX/VMS command

   ```
   $ run mkdat
   ```

   The program generates a data file. It also issues messages for duplicate entries in WORD.DAT and displays the total number of distinct words. You will need this number for the next step.

5. Edit the program CORRECT.FOR by issuing the VAX/VMS command

   $ edit/edt correct.for

   and change all the occurrences of ??? to the total number of distinct words.

6. Execute the program GENGUESS.COM by issuing the VAX/VMS command

   $ @genguess

   It compiles the CORRECT.FOR program and creates a new GUESS/1 interpreter that contains your new dictionary. NOTE: The new interpreter occupies a fairly large disk space.

The new GUESS/1 interpreter is now residing in the file GUESS.EXE in your directory.

Chapter XIII

DISCUSSION

Wisdom is supreme;
  therefore get wisdom.
Though it costs all you have,
  get understanding.

- Solomon, PROVERBS

13.1 A GENERAL PURPOSE EXPERT SYSTEMS SHELL

First, GUESS/1 is a general purpose tool because it offers many domain-independent knowledge representation schemes and control structures for building expert systems. Moreover, GUESS/1 permits a system developer to implement additional features that are not currently supported by the system. In some problem domains, more complex knowledge representations and control strategies may be desirable. They can be implemented using the GUESS/1 primitives and the Prolog language.

Second, GUESS/1 is an expert systems shell because it is an interface between the application expert systems and the Prolog language. The overall structure is shown in Figure 78 . Since GUESS/1 is completely domain-independent and can

be tailored for many different applications, we anticipate that some domain-specific shells will be implemented in GUESS/1, changing the overall structure to Figure 79 . An analogy can be drawn from the ongoing development in the conventional programming world [DoD, 1980] (See Figure 80).

Two expert systems in two different domains (pharmacology and drug interactions, and pest and orchard management) have been successfully implemented in GUESS/1, showing that the shell has some generality and usability. As for most academic and commercial expert system building tools, GUESS/1 has not supported enough implementations to allow useful evaluations. However, an extensive use of GUESS/1 in the near future will demonstrate its strengths and weaknesses and thus contribute to its improvement in many different aspects. There are deficiencies in GUESS/1 that need to be dealt with. The limitations include inability to handle inexact reasoning and to deal with the concept of time. Future work will enhance the built-in control strategies to include constraint satisfaction, default reasoning, agenda and planning systems, so that a system developer does not have to implement these powerful features using the GUESS/1 primitives and the Prolog language. All in all, the goal is to make GUESS/1 user-friendly, powerful, easy to use and easy to customize.

```
--------------------
| Expert Systems   |
--------------------
|      GUESS/1     |
--------------------
|      Prolog      |
--------------------
| Operating System |
--------------------
```

Figure 78: Overall Structure of Expert Systems Shell

```
--------------------
|  Expert Systems  |
--------------------
|  Domain-Specific |
|      Shells      |
--------------------
|      GUESS/1     |
--------------------
|      Prolog      |
--------------------
| Operating System |
--------------------
```

Figure 79: Anticipated Structure of Expert Systems Shell

```
-----------------------------------------
|       Application Ada Programs        |
-----------------------------------------
|  Ada Programming Support Environments |
|                (APSEs)                |
-----------------------------------------
|       Common APSE Interface Set       |
|                (CAIS)                 |
-----------------------------------------
|           The Ada Language            |
-----------------------------------------
|           Operating System            |
-----------------------------------------
```

Figure 80: The Ada Programming Environments

13.2 A COMPARISON WITH S.1

GUESS/1 and S.1 [Teknowledge, 1984], a commercial expert systems building tool, share some common philosophies in knowledge representation and control. S.1 knowledge base objects are comprised of (1) control blocks, (2) classes with their attributes and values, and (3) rules. Control blocks are used to represent imperative knowledge that dictates the flow of a consultation by specifying when instances are created, when attributes are determined, how values are determined and what information is displayed. The steps in a control block represent the high-level order of steps to take to solve a given problem. Control blocks in S.1 resemble GUESS/1 action frames. Figure 81 shows an S.1 control block [Teknowledge, 1984] and a GUESS/1 action frame for diagnosing and repairing a car.

In Figure 81, the S1 control block invokes other control blocks (initial.checks, display.recommendations and display.systems). Similarly, the GUESS/1 action frame activates other action frames ("initial checks," "display recommendations" and "display systems") to perform the desired functions. Problem solving strategies in S.1 and GUESS/1 can be organized hierarchically.

S.1 Control Block:

```
DEFINE CONTROL.BLOCK diagnose_and_repair_car
 ::INVOCATION  top.level
 ::TRANSLATION "diagnose and repair a car"
 ::BODY
   begin vars c:car;
    display spaces(15)
      ! "Welcome to the Car Repair Advisor." !
       new.line();
    create.instance car called c;
    determine symptoms[c];
    invoke initial.checks(c);
    determine systems[c];
    determine cause.of.problem[c];
    determine recommendations[c];
    invoke display.recommendations(c);
    determine final.system.check[c];
    invoke display.systems(c)
   end
END.DEFINE
```

GUESS/1 Action Frame:

```
(  (AF "diagnose and repair a car")
      ->
     (USAGE "diagnose and repair a car" LEVEL 1)
     (ADD  OBJECT_OF_INTEREST  car)
     (MESSAGE "Welcome to the Car Repair Advisor.")
     (KF "determine symptoms")
     (AF "initial check")
     (KF "determine systems")
     (KF "determine cause of problem")
     (KF "determine recommendations")
     (KF "determine extra recommendations")
     (AF "display recommendations")
     (KF "determine final system check")
     (AF "display system")
     (cut)  )
```

Figure 81: S1 Control Block vs GUESS/1 Action Frame

The S1 control block in Figure 81 also tries to determine the values of the attributes (symptoms, systems, cause.of.problem, recommendations, extra.recommendations, final.system.check) defined on the class object (car). In S.1, a class is an object, an event or a concept. The properties of a class are termed attributes. Attributes, therefore, contain information about an instance of a class. In the same Figure, the GUESS/1 action frame instantiates knowledge frames ("determine symptoms," "determine systems," "determine cause of problem," "determine recommendations," "determine extra recommendations" and "determine final system check") to determine the values of the attributes of the car. Class attributes in S.1 resembles knowledge frames in GUESS/1. Figure 82 shows a class attribute definition in S.1 and a GUESS/1 knowledge frame on determining the symptoms of a car.

As shown in Figure 82, the S.1 class attribute definition declares legal means and determination means for each attribute. The system can "query user" or "try rules." The order in which the system attempts to determine an attribute is specified in the definition. Similarly, the GUESS/1 knowledge frame allows the specification of means to determine values for each attribute. In GUESS/1, the system can query user, try rules, look up information from the

S.1 Attribute Definition

```
DEFINE ATTRIBUTE symptoms
 ::DEFINED.ON                car
 ::TYPE                      text
 ::MULTIVALUED               true
 ::MULTIVALUED.SEEK.TYPE     all.values
 ::LEGAL.VALUES              symptom.values
 ::LEGAL.MEANS               {query.user}
 ::DETERMINATION.MEANS       {query.user}
 ::PROMPT                    "What is wrong with "
                             ! instance.trans(car) ! "? "
 ::TRANSLATION    "the initial symptoms about the car "
```

GUESS/1 Knowledge Frame

```
(   (KF "determine symptoms")
      ->
      (USAGE "determine symptoms of a car" LEVEL ANY)
      (GETCURRENT  OBJECT_OF_INTEREST  *car)
      (ASK_USER
           QUESTION IS "What is wrong with "
           OBJECT   IS *car
           EXPECT   {any legal symptom values}
           USELESS  unknown
           ANSWER   IS *answer)
)
```

Figure 82: S.1 Class Attribute Definition vs GUESS/1 Knowledge Frame

databases, take the pre-defined default values or instantiate other knowledge frames.

Judgment and factual knowledge in S.1 are represented by rules. Rules are grouped together by class attribute definitions. In GUESS/1, production rules are used for similar purposes and the rules are grouped by knowledge frames and action frames. Figure 83 shows a S.1 rule and a GUESS/1 rule for the cranking test of a car.

The S.1 rule shown in Figure 83 says that if there is dead silence or if the engine just clicks or is sluggish during cranking, then the battery should be charged. The GUESS/1 rule following it has exactly the same meaning. Certainty factors are employed in S.1 rules for judgemental knowledge. The current version of GUESS/1, however, is not capable of handling uncertainties.

Although GUESS/1 and S.1 resemble each other in some of their design philosophies, GUESS/1 provides more diversity and flexiblity in structuring knowledge and control. For example, GUESS/1 action frames allow an easy implementation of AND/OR goal trees with automatic backtracking, but S.1 control blocks support only strict AND trees with no backtracking. GUESS/1 provides high-level knowledge structures such as semantic nets and relational tables, but

S.1 Rule

```
DEFINE RULE rule100
 ::APPLIED.TO    c.car
 ::PREMISE       cranking.test[c] is in
                {dead.silence, clicks, sluggish.cranking}
 ::CONCLUSION    preliminary.suggestions[c] =
                 "Please charge the battery."
```

GUESS/1 Rule

```
(  (PR "cranking test")
     ->
     (USAGE "rule100: cranking test" LEVEL ANY)
     ; PREMISE
     (PERFORM "cranking test" RESULT IS *result)
     (MEMBER_OF *result
             (dead_silence clicks sluggish_cranking))
     ; CONCLUSION
     (SUGGESTIONS "Please charge the battery")  )
```

Figure 83: S.1 Rule vs GUESS/1 Rule

S.1 provides only the primitives - classes and their attributes and associated values.

Dissimilarities between GUESS/1 and S.1 are very apparent in the programming style (see Figures 81, 82 and 83) and in the expert systems design methodology. GUESS/1 applies the life cycle of model development (see chapter XI) to building expert systems whereas S.1 follows the conventional procedures: select domain, prepare plans, acquire and refine knowledge, encode knowledge, execute and test, and acquire and encode more knowledge. Another major difference is that S.1 has a virtual machine whereas GUESS/1 runs on top of a Prolog interpreter. The S.1 virtual machine in a sense imposes a restraint on the system developer. On the contrary, GUESS/1 allows a developer to implement additional features using the GUESS/1 primitives and the Prolog language. However, it requires the developer to have some experience in programming in Prolog, which is the main disadvantage of not having a virtual GUESS/1 machine.

BIBLIOGRAPHY

Adams, J. B., "A Probability Model of Medical Reasoning and the MYCIN Model," Mathematical Biosciences 32, 177-186 (1976).

Aikins, J. S., "Prototypes and Production Rules: A Knowledge Representation for Computer Consultations," Report No. STAN-CS-80-814, Computer Science Department, Stanford University, Stanford, California (1980).

Aikins, J. S., "Prototypical Knowledge for Expert Systems," Artificial Intelligence 20 (1983), 63-210.

American Pharmaceutical Association, Evaluations of Drug Interactions, 2nd Ed. American Pharmaceutical Association, Washington, D.C. (1976).

Appelt, D. E., "A Planner for Reasoning About Knowledge and Action," Proc. National Conference on Artificial Intelligence, (1980) 131-133.

Arons, H., "Expert Systems in the Simulation Domain," Mathematics and Computers in Simulation XXV (1983) 10-16.

Balci, O., "The Life Cycle of Model Development and Model Development Environments," Technical Report CS83022-R, Department of Computer Science, Virginia Polytechnic Institute and State University, 1983.

Balzer, R., L. Erman, P. London, C. Williams, "HEARSAY-III: A Domain-Independent Framework for Expert Systems," Proc. National Conference on Artificial Intelligence, 1980.

Bandler, W., "Representation and Manipulation of Knowledge in Fuzzy Expert Systems," Proc. Workshop on Fuzzy Sets and Knowledge-Based Systems, 1983.

Barstow, D. R., N. Aiello, R. Duda, L. Erman, C. Forgy, R. Greiner, D. B. Lenat, P. London, J. McDermott, H. P. Nii, S. Weiss, "Languages and Tools for Building Expert Systems," in F. Hayes-Roth, D. A. Waterman, and D. R. Lenat (eds.), Building Expert Systems, Addison-Wesley, 1983.

The BDM Corporation, "A Survey of Methods to Represent Uncertainty in Data Bases," 7915 Jones Branch Drive, McLean, VA 22102, October 31, 1984.

Bobrow, D. G., T. Winograd, "An Overview of KRL, A Knowledge Representation Language," Cognitive Science, Vol. 1, No. 1, 1977.

Bobrow, D. G., T. Winograd, "KRL: Another Perspective," Cognitive Science, Vol. 3, No. 1, 1979.

Boose, J. H., "Personal Construct Theory and the Transfer of Human Expertise," Proc. National Conference on Artificial Intelligence, (1984) 27-33.

Buchanan, B. G., E. A. Feigenbaum, "DENDRAL and Meta-DENDRAL: Their Applications Dimensions," Artificial Intelligence, 1978, 11(1,2), 5-24.

Burton, R., "Semantic Grammar: An Engineering Technique for Constructing Natural Language Understanding Systems," Report No. 3453, Bolt, Beranek and Newman, Boston (1976).

Chester, D., "Implementing Production Rules in Prolog," Department of Computer and Information Sciences, University of Delaware, Newark, Delaware, USA 19711.

Clocksin, W. F., C. S. Mellish, Programming in Prolog, Springer-Verlag, New York (1981).

Colby, K. M., R. C. Parkison, B. Faught, "Pattern-Matching Rules for the Recognition of Natural Language Dialogue Expressions," Department of Computer Science, Stanford University, 1974.

Cuena, J., "The Use of Simulation Models and Human Advice to Build an Expert System for the Defense and Control of River Floods," Proc. 8th International Joint Conference on Artificial Intelligence, (1983) 246-249.

Darvas, F., J. Futo, P. Szeredi, "Logic-Based Program System for Predicting Drug Interaction," Int. J. Bio-Medical Computing 9, (1978) 259-271.

Davis, R., "Applications of Meta-Level Knowledge to the Construction, Maintenance and Use of Large Knowledge Bases," Memo AIM-283, Ph.D. dissertation, Department of Computer Science, Stanford University, Stanford, California (1974).

Davis, R., "Expert systems: Where are we? And where do we go from here?" The AI Magazine, 3-22, Spring (1982).

Davis, R., B. G. Buchanan, E. H. Shortliffe, "Production Rules as a Representation for a Knowledge-based Consultation Program," Artificial Intelligence, Vol. 8, No. 1, 1977.

Davis, R., "Interactive Transfer of Expertise: Acquisition of New Inference Rules," Proc. International Joint Conference on Artificial Intelligence, 1977.

Davis, R., J. J. King, "An Overview of Production Systems," in Machine Intelligence 8, E. Elcock & D. Michie (Eds.), Horwood, Chichester, England, 1977.

Davis, R., "Applications of Meta Level Knowledge to the Construction, Maintenance and Use of Large Knowledge Bases," in Knowledge-Based Systems in Artificial Intelligence, R. Davis & D. B. Lenat (Eds.), McGraw-Hill, New York, 1982.

Davis, R., D. B. Lenat, Knowledge-Based Systems in Artificial Intelligence, New York: McGraw-Hill, 1982.

DoD Computer Security Center, "Department of Defense Trusted Computer System Evaluation Criteria," CSC-STD-001-83, Library No. S225,711, 15 August 1983.

Doyle, J., "A Truth Maintenance System," Artificial Intelligence 12 (1979) 231-272.

Duda, R. O., P. E. Hart, K. Konolige, R. Reboh, "A Computer-based Consultant for Mineral Exploration," Technical Report, SRI International, Sept. 1979.

Duda, R. O., E.H. Shortliffe, Expert Systems Research, Science 220, 261-268 (1983).

Ehrich, M., J. R. Wilcke, S. Lee, J. W. Roach, "An Expert System for Prediction of Drug Interactions Involving Antibiotics," Proc. 24th Interscience Conference on Antimicrobial Agents and Chemotherapy, 1984.

Engelman, C., J. K. Millen, E. A. Scarl, "KNOBS: An Integrated AI Interactive Planning Architecture," The MITRE Corporation, Bedford, MA 01730.

Fahlman, S., NETL: A System for Representing and Using Real World Knowledge, M.I.T. Press, Cambridge, MA (1979).

Fain, J., D. Gorlin, F. Hayes-Roth, S. Rosenschein, H. Sowizral, "The ROSIE Language Reference Manual," N-1647-ARPA, The Rand Corporation, Santa Monica, California, 1981.

Fain, J., F. Hayes-Roth, H. Sowizral, D. Waterman, "Programming in ROSIE: An Introduction by Means of Examples," N-1646-ARPA, The Rand Corporation, Santa Monica, California, 1982.

Faught, W., K. Colby, L. Parkison, "The Interaction of Inferences, Affects and Intentions in a Model of Paranoia," Memo-AIM253, Department of Computer Science, Stanford University, Stanford, California (1974).

Feigenbaum, E. A., "The Art of Artificial Intelligence: I. Themes and Case Studies of Knowledge Engineering," Proc. Fifth International Joint Conference on Artificial Intelligence, MIT, 1977.

Feigenbaum, E. A., "Knowledge Engineering: The Applied Side of Artificial Intelligence," Heuristic Programming Project, Computer Science Department, Stanford University, 1983.

Findler, N. V., J. E. Brown, R. Lo, H. Y. You, "A module to estimate numerical values of hidden variables for expert systems," Int. J. Man-Machine Studies (1983) 18, 323-335.

Forsyth, R., "The architecture of expert systems," in Expert Systems: Principles and Case Studies, R. Forsyth (Ed.), Chapman and Hall Computing, 1983.

Forsyth, R., "Fuzzy reasoning systems," in Expert Systems: Principles and Case Studies, R. Forsyth (Ed.), Chapman and Hall Computing, 1983.

Gevarter, W. B., "Expert Systems: Limited but Powerful," IEEE Spectrum, pp 39-45, August 1983.

Goldstein, I. P., D. G. Bobrow, "Descriptions for a Programming Environment," Proc. National Conference on Artificial Intelligence, (1980) 187-189.

Goldstein, I. P., R. B. Roberts, "NUDGE, A Knowledge-based Scheduling Program," IJCAI 5, 1977, pp. 257-263.

Gomez, F., B. Chandrasekaran, "Knowledge Organization and Distribution for Medical Diagnosis," IEEE Transactions on System, Man, and Cybernetics, Vol. SMC-11, No. 1, January 1981, 34-42.

Greiner, R., D. B. Lenat, "A Representation Language Language," Proc. National Conference on Artificial Intelligence, (1980) 165-169.

Haas, N., G. G. Hendrix, "An Approach to Acquiring and Applying Knowledge," Proc. National Conference on Artificial Intelligence, (1980) 235-239.

Hart, P. E., R. O. Duda, M. T. Einaudi, "A Computer-based Consultation System for Mineral Exploration," Technical Report, SRI International, May 1978.

Hayes, P., G. Mouradian, "Flexible Parsing," Am. J. Comput. Ling. 7, 232-242 (1981).

Hayes-Roth, B., F. Hayes-Roth, "A Cognitive Model of Planning," Cognitive Science 3, (1979) 275-310.

Hayes-Roth, F., "Knowledge-Based Expert Systems," IEEE Computer, October 1984, pp 263-273.

Hayes-Roth, F., D. Gorlin, S. Rosenschein, H. Sowizral, D. Waterman, "Rationale and Motivation for ROSIE," N-1648-ARPA, The Rand Corporation, Santa Monica, California, 1981.

Hayes-Roth, F., P. Klahr, D. J. Mostow, "Knowledge Acquisition, Knowledge Programming, and Knowledge Refinement," R-2540-NSF, The Rand Corporation, Santa Monica, California, 1980.

Hewitt, C., "PLANNER: A Language for Proving Theorems in Robots," Proc. International Joint Conference on Artificial Intelligence, 1971.

Kinnucan, P., "Computers that think like experts," High Technology, January 1984.

Hussar, D. A., Drug Interactions, in Remington's Pharmaceutical Sciences, 16th ed., A. Osel, Ed., pp. 1741-1777. Mack Publishing Co., Easton, Pa. (1980).

Jones, P.L.K., "REVEAL: An Expert Systems Support Environment," in Expert Systems: Principles and Case Studies, R. Forsyth (Ed.), Chapman and Hall Computing, 1983.

Klahr, P., "A Simulation System for Decision Support," Proc. 14th Hawaii International Conference on System Sciences (1981), 559-565.

Klahr, P., W. S. Faught, "Knowledge-Based Simulation," Proc. National Conference on Artificial Intelligence (1980), 181-183.

Klahr, P., D. McArthur, "The ROSS Language Manual," N-1854-AF, The Rand Corporation, Santa Monica, California, 1982.

Klahr, P., D. McArthur, "SWIRL: Simulating Warfare in the ROSS Language," N-1885-AF, The Rand Corporation, Santa Monica, California, 1982.

Konolige, K., N. J. Nilsson, "Multiple-Agent Planning Systems," Proc. National Conference on Artificial Intelligence, (1980) 138-142.

Kowalski, R. A., Logic for Problem Solving, North Holland, Amsterdam, 1979.

Lenat, D. B., A. Clarkson, G. Kiremidjian, "An Expert System for Indication and Warning Analysis," International Joint Conference on Artificial Intelligence, 1983.

Lesser, V. R., L. D. Erman, "A retrospective view of the HEARSAY-II architecture," Proc. Fifth International Joint Conference on Artificial Intelligence, IJCAI, Cambridge, MA, August 1977.

Long, W. J., T. A. Russ, "A Control Structure for the Time Dependent Reasoning," International Joint Conference on Artificial Intelligence, 1983.

McCarthy, J., "Recursive Functions of Symbolic Expressions and Their Computation by Machine," Communications of the ACM, Vol. 7, pp 184-195, April 1960.

McDermott, J., "R1: A Rule-based Configurer of Computer Systems," Artificial Intelligence 19 (1982), 39-88.

Miller, R. A., H. E. Pople Jr., J. D. Myers, "Internist-1: An Experimental Computer-based Diagnostic Consultant for General Internal Medicine," N.E.J. Med. 307 (1982), 68-76.

Minsky, M., "A Framework for Representing Knowledge," in P.H. Winston (ed.) The Psychology of Computer Vision, McGraw Hill, New York, NY (1975).

Morrelli, H. F., K. L. Melmon, Drug Interactions in Clinical Pharmacology, Basic Principles in Therapeutics, 2nd ed., K. L. Melmon and H. F. Morrelli, Ed., pp. 982-1007. Macmillan Publishing Co., New York (1978).

Nau, D. S., Expert Computer Systems, IEEE Trans. Comput. 18, 63-85 (1983).

Nance, R. E., "Model Representation in Discrete Event Simulation: The Conical Methodology," Technical Report CS81003-R, Department of Computer Science, Virginia Polytechnic Institute and State University,1981.

Pauker, S. G., P. Szolovits, "Analyzing and Simulating Taking the History of the Present Illness: Context Formation," in Schneider/Segvall Hein (Eds.), Computational Linguistics in Medicine, North-Holland, Amsterdam, 1977, pp. 109-118.

Pazzani, M. J., C. Engelman, "Knowledge Based Question Answering," Proc. Conference on Applied Natural Language Processing, (1983) 73-80.

Reboh, R., "Using a Matcher to Make an Expert Consultation System Behave Intelligently," Proc. National Conference on Artificial Intelligence, (1980) 231-234.

Roach, J., G. Fowler, The HC Manual: Virginia Tech Prolog, Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, Va. (1983).

Roach, J., S. Lee, J. Wilcke, M. Ehrich, "An Expert System for Information on Pharmacology and Drug Interactions," Computers in Biology and Medicine, Vol. 15, No. 1, 1985.

Roach, J., S. Lee, J. Wilcke, M. Ehrich, "An Expert System that Criticizes Decisions in Combination Drug Therapy," Proc. First Conference on Artificial Intelligence Applications, 1984.

Roberts, R. B., I. P. Goldstein, "The FRL Manual," MIT AI Lab., Memo 409, Sept 1977.

Robinson, A. E., D. E. Wilkins, "Representing Knowledge in an Interactive Planner," Proc. National Conference on Artificial Intelligence, (1980) 148-150.

Schank, R. C., Scripts, Plans, Goals and Understanding, Wiley, 1977.

Schank, R. C., Inside Computer Understanding: Five Programs Plus Miniatures, R. C. Schank & C. K. Riesbeck (Eds.), Hillsdale, N.J., 1981.

Shannon, R. E., "Artificial Intelligence and Simulation," Proc. 1984 Winter Simulation Conference, (1984) 3-9.

Shortliffe, E. H., Computer-based Medical Consultations: MYCIN, Elsevier, New York, 1976.

Shortliffe, E. H., B. G. Buchanan, "A Model of Inexact Reasoning in Medicine," Mathematical Biosciences, Vol. 23, 1975.

Siklossy, L., "Modeled Exploration By Robot," Technical Report TR-1, Department of Computer Science, University of Texas at Austin, 1972.

Slagle, J. R., E. J. Halpern, H. Hamburger, R. R. Cantone, "A Decision Support System for Fire Support Command and Control," Proceedings - Trends and Applications 1983: Automating Intelligent Behavior Applications and Frontiers, (1983) 68-75.

Smith, R. G., P. Friedland, "A Users Guide to the Units Systems," Technical Report, Heuristic Programming Project, Stanford University, 1980.

Stefik, M., "Planning with Constraints (MOLGEN: Part 1)," Artificial Intelligence 16 (1981), 111-140.

Stefik, M., "Planning and Meta-Planning (MOLGEN: Part 2)," Artificial Intelligence 16 (1981), 141-170.

Stefik, M., J. Aikins, R. Balzer, J. Benoit, L. Birnbaum, F. Hayes-Roth, E. Sacerdoti, "The Organization of Expert Systems, A Tutorial," Artificial Intelligence 18 (1982) 135-173.

Swinehart, D., R. Sproull, "SAIL," Technical Report 57.2, Stanford AI Project, 1971.

Szolovits, P., S. G. Pauker, "Categorical and Probabilistic Reasoning in Medical Diagnosis," Artificial Intelligence 11 (1978), 115-144.

U.S. Department of Health and Human Services, Expert Computer Program Overhauled, Res. Resources Reporter VII, No. 5, 7-8 (1983).

Teknowledge, Inc., "S.1 User's Guide," Teknowledge, Inc., Palo Alto, CA, June, 1984.

Teknowledge, Inc., "S.1 Sample Knowledge Systems," Teknowledge, Inc., Palo Alto, CA, 1984.

Van Melle, W., "EMYCIN: A Domain-independent Production-rule System for Consultation Programs," Heuristic Programming Project, Stanford University, 1980.

Van Melle, W., A. C. Scott, J. S. Bennett, M. A. Peairs, "The EMYCIN Manual," Technical Report, Heuristic Programming Project, Stanford University, 1981.

Warren, D., "Implementing Prolog: Compiling Predicate Logic Programs," Reports 39 and 40, Department of Artificial Intelligence, University of Edinburgh, U.K. (1977).

Warren, D. H., L. M. Pereira, "Prolog - The Language and Its Implementation Compared to Lisp," Proc. Symposium on AI and Programming Language, SIGPLAN Notices 12 (8) and SIGART Newsletter 64, 1977.

Waterman, D. A., R. H. Anderson, Frederick Hayes-Roth, Philip Klahr, Gary Martins, Stanley J. Rosenschein, "Design of a Rule-Oriented System for Implementing Expertise," N-1158-1-ARPA, The Rand Corporation, Santa Monica, California, 1979.

Waterman, D. A., F. Hayes-Roth, "An Investigation of Tools for Building Expert Systems," R-2818-NSF, The Rand Corporation, Santa Monica, California, 1982.

Weiss, S. M., C. A. Kulikowski, "EXPERT: A System for Developing Consultation Models," Proc. 6th International Joint Conference on Artificial Intelligence, (1979) 942-947.

Weiss, S. M., C. A. Kulikowski, S. Amarel, "A Model-Based Method for Computer-Aided Medical Decision-Making," Artificial Intelligence 11, (1978) 145-172.

Zadeh, L. A., "Making computers think like people," IEEE Spectrum, August 1984, pp 26-32.

Zadeh, L. A., Fuzzy Sets and Systems, North Holland, Amsterdam, 1983. 68-76 (1982).

A TOPICAL INDEX TO GUESS/1 LIBRARY ROUTINES

Blackboard Manipulation Routines ... 100
- BLACKBOARD_ERASE ... 100
- BLACKBOARD_LOOKUP ... 100-101
- BLACKBOARD_UPDATE ... 102-103

Explanation Subsystem Routines ... 104
- SYSMSG ... 104
- USAGE ... 104

GUESS/1 Initialization and Execution Routines ... 105
- GETCOMMAND ... 105
- GETTERMTYPE ... 105
- GUESS/1 ... 105
- SHOWCOMMAND ... 106
- START ... 106

Hierarchical Tree Manipulation Routines ... 107
- TREE_ADD ... 107
- TREE_CREATE ... 107-108
- TREE_DESTROY ... 108
- TREE_ERASE ... 109
- TREE_EXPAND ... 109
- TREE_LOOKUP ... 110
- TREE_UPDATE ... 111

Human-System Interface Routines ... 112
- ASK_USER ... 112, 114
- g$spell ... 115
- MENU ... 116

Miscellaneous Utility Routines ... 117
APPENDA ... 117
CHECK_WORD ... 117
DECREMENT ... 117
DEFAULT_OF ... 118
DELETE_MEMBER ... 118
DIFFERENT_SET ... 119
DISPLAY_HR_MIN_SEC ... 119
g$stratm ... 120
GET PROMPT ... 120
GETELEMENT ... 120
INCREMENT COUNTER ... 121
INCREMENT QUERY# ... 121
INCREMENT SYSCMD# ... 122
LISTLEN ... 122
LPRINT ... 123
MEMBER ... 123
NOT_MEMBER ... 123
NUMERIC ... 124
PRECISION ... 124
PRINT_LIST ... 124
RESET CPU ... 124
RESET QUERY# ... 125
RESET SYSCMD# ... 125
RESET TIME ... 125
RET_QUESTION ... 126
RN ... 127
RN_SEED ... 126
RN_SEED RANDOM ... 126
SAME_SET ... 127
SAVE KNOWLEDGE_BASE ... 127
SAVE_QUESTION ... 128
SET COUNTER ... 128
SET PROMPT ... 128
SET SYSMSG OFF ... 129
SET SYSMSG ON ... 129
SET TERMINAL ECHO ... 129
SET TERMINAL NOECHO ... 130
SET TRACE OFF ... 130
SET TRACE ON ... 130
SHOW APPLICATION ... 131
SHOW COUNTER ... 131
SHOW CPU ... 131
SHOW DATETIME ... 132
SHOW FLAGS ... 132
SHOW TIME ... 132
SUBSET_OF ... 133
SUCCEED ... 133
SUPERSET_OF ... 133
SYSCOMMAND ... 134

SYSTEM ... 134

Object-List Manipulation Routines ... 135
ADD ... 135
GETALL ... 135
GETCURRENT ... 136
GETFIRST ... 136
GETNEXT ... 137
OBJECTLIST_ERASE ... 137

Relational Table Manipulation Routines ... 138
TABLE_ADD ... 138
TABLE_CREATE ... 138-139
TABLE_DESTROY ... 139
TABLE_ERASE ... 140
TABLE_LOOKUP ... 141

Semantic Net Manipulation Routines ... 142
DNET_ADD ... 142
DNET_CREATE ... 142-143
DNET_DESTROY ... 143
DNET_ERASE ... 144
DNET_RETRIEVE ... 145
UNET_ADD ... 146
UNET_CREATE ... 146
UNET_DESTROY ... 147
UNET_ERASE ... 147
UNET_RETRIEVE ... 148
Stack Manipulation Routines ... 149
INIT_STACK ... 149
POP ... 149
PUSH ... 149
TOP_OF_STACK ... 150

Standard SSA_MAP Routines ... 151
CLEAR ... 151
CPU ... 151
ECT ... 151
HELP ... 151
MAIL ... 151
MENU ... 151
MSG ... 151
NOMSG ... 151
NOTRACE ... 151
REPEAT ... 151
RN ... 151
SAY ... 151
SPELL ... 151
STATUS ... 151
STOP ... 151
TIME ... 151
TRACE ... 151
USER ... 151
Syntax-Semantic Action Mapping Routines ... 152
GET_QUESTION ... 152
REMOVE_NONKEYWORDS ... 152
REMOVE_PUN ... 153
SUB_CONTROL_Z ... 153
SUBSTITUTE_SYNONYMS ... 153
SYSHELP Routines ... 154
SYSHELP ... 154
System Security Routines ... 155
CLASSIFICATION ... 155
LOGIN ... 156
LOGOUT ... 156
NEED_TO_KNOW ... 156

Terminal Input/Output Routines ... 157
CLEAR_SCREEN ... 157
HOLD_MORE ... 157
SET CURSOR ... 158
SET TERM$TYPE ... 158

AN ALPHABETICAL INDEX TO GUESS/1 LIBRARY ROUTINES

ADD ... 135
APPENDA ... 117
ASK_USER ... 112, 114

BLACKBOARD_ERASE ... 100
BLACKBOARD_LOOKUP ... 100-101
BLACKBOARD_UPDATE ... 102-103

CHECK_WORD ... 117
CLASSIFICATION ... 155
CLEAR ... 151
CLEAR_SCREEN ... 157
CPU ... 151

DECREMENT ... 117
DEFAULT_OF ... 118
DELETE_MEMBER ... 118
DIFFERENT_SET ... 119
DISPLAY_HR_MIN_SEC ... 119
DNET_ADD ... 142
DNET_CREATE ... 142-143
DNET_DESTROY ... 143
DNET_ERASE ... 144
DNET_RETRIEVE ... 145

ECT ... 151

g$spell ... 115
g$stratm ... 120
GET PROMPT ... 120
GET_QUESTION ... 152
GETALL ... 135
GETCOMMAND ... 105
GETCURRENT ... 136
GETELEMENT ... 120
GETFIRST ... 136
GETNEXT ... 137
GETTERMTYPE ... 105
GUESS/1 ... 105

HELP ... 151
HOLD_MORE ... 157

INCREMENT COUNTER ... 121
INCREMENT QUERY# ... 121
INCREMENT SYSCMD# ... 122

INIT_STACK ... 149

LISTLEN ... 122
LOGIN ... 156
LOGOUT ... 156
LPRINT ... 123

MAIL ... 151
MEMBER ... 123
MENU ... 116, 151
MSG ... 151

NEED_TO_KNOW ... 156
NOMSG ... 151
NOT_MEMBER ... 123
NOTRACE ... 151
NUMERIC ... 124

OBJECTLIST_ERASE ... 137

POP ... 149
PRECISION ... 124
PRINT_LIST ... 124
PUSH ... 149

REMOVE_NONKEYWORDS ... 152
REMOVE_PUN ... 153
REPEAT ... 151
RESET CPU ... 124
RESET QUERY# ... 125
RESET SYSCMD# ... 125
RESET TIME ... 125
RET_QUESTION ... 126
RN ... 127, 151
RN_SEED ... 126
RN_SEED RANDOM ... 126

SAME_SET ... 127
SAVE KNOWLEDGE_BASE ... 127
SAVE_QUESTION ... 128
SAY ... 151
SET COUNTER ... 128
SET CURSOR ... 158
SET PROMPT ... 128
SET SYSMSG OFF ... 129
SET SYSMSG ON ... 129

SET TERM$TYPE ... 158
SET TERMINAL ECHO ... 129
SET TERMINAL NOECHO ... 130
SET TRACE OFF ... 130
SET TRACE ON ... 130
SHOW APPLICATION ... 131
SHOW COUNTER ... 131
SHOW CPU ... 131
SHOW DATETIME ... 132
SHOW FLAGS ... 132
SHOW TIME ... 132
SHOWCOMMAND ... 106
SPELL ... 151
START ... 106
STATUS ... 151
STOP ... 151
SUB_CONTROL_Z ... 153
SUBSET_OF ... 133
SUBSTITUTE_SYNONYMS ... 153
SUCCEED ... 133
SUPERSET_OF ... 133
SYSCOMMAND ... 134
SYSHELP ... 154
SYSMSG ... 104
SYSTEM ... 134
TABLE_ADD ... 138
TABLE_CREATE ... 138-139
TABLE_DESTROY ... 139
TABLE_ERASE ... 140
TABLE_LOOKUP ... 141
TIME ... 151
TOP_OF_STACK ... 150
TRACE ... 151
TREE_ADD ... 107
TREE_CREATE ... 107-108
TREE_DESTROY ... 108
TREE_ERASE ... 109
TREE_EXPAND ... 109
TREE_LOOKUP ... 110
TREE_UPDATE ... 111

UNET_ADD ... 146
UNET_CREATE ... 146
UNET_DESTROY ... 147
UNET_ERASE ... 147
UNET_RETRIEVE ... 148
USAGE ... 104
USER ... 151

The vita has been removed from the scanned document