City Research Online

City St George's, University of London

Citation: Chelsom, J.J.L. (1990). The interpretation of data in intensive care medicine: An application of knowledge-based techniques. (Unpublished Doctoral thesis, City, University of London)

This is the accepted version of the paper.

This version of the publication may differ from the final published version. To cite this item please consult the publisher's version.

Permanent repository link: https://openaccess.city.ac.uk/id/eprint/28493/

Copyright and Reuse: Copyright and Moral Rights remain with the author(s) and/or copyright holders. Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge, unless otherwise indicated, provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way. For full details of reuse please refer to City Research Online policy.

City Research Online: http://openaccess.city.ac.uk/ publications@citystgeorges.ac.uk

THE INTERPRETATION OF DATA IN INTENSIVE CARE MEDICINE: AN APPLICATION OF KNOWLEDGE-BASED TECHNIQUES

JOHN JAMES LEONARD CHELSOM

Thesis submitted for the degree of Doctor of Philosophy

City University

Research Centre for Measurement and Information in Medicine

January 1990

CONTENTS

List of Tables 6

List of Figures 6

Acknowledgements 9

Declaration 9

Abstract 10

PART ONE

1. Introduction 11
- 1.1 Background 11
- 1.2 Objectives 14
- 1.3 Outline of the Thesis 15

2. Knowledge-Based Systems in Medicine 16
- 2.1 Introduction 16
- 2.2 Early Systems 18
  - 2.2.2 The MYCIN System . 18
  - 2.2.3 CASNET 23
  - 2.2.4 The Present Illness Program 26
  - 2.2.5 INTERNIST-1 29
- 2.3 The Second Generation 31
  - 2.3.1 Introduction 31
  - 2.3.2 ABEL 33
  - 2.3.3 ATTENDING 35
  - 2.3.4 CADIAG-2 37
  - 2.3.5 A Set Covering Model of Diagnosis 38
- 2.4 The Third Generation 41
  - 2.4.1 Introduction 41
  - 2.4.2 MUNIN 42
  - 2.4.3 CHECK 43
- 2.5 Summary 46

3. Methods of Knowledge Representation and Control 47
- 3.1 Introduction 47
- 3.2 Rule-Based Representation 48
  - 3.2.1 Background 48
  - 3.2.2 Structure 48
  - 3.2.3 Representation 49
  - 3.2.4 Control 50
  - 3.2.5 Summary 53
- 3.3 Frame-Based Representation 54
  - 3.3.1 Background 54
  - 3.3.2 Structure 54
  - 3.3.3 Control 55
  - 3.3.4 Summary 56
- 3.4 Semantic Networks 56
  - 3.4.1 Background 56
  - 3.4.2 Reasoning in a Semantic Network 56
  - 3.4.3 Summary 59

3.5 The Blackboard Architecture 59
3.5.1 Origins 59
3.5.2 The Hearsay-II System 60
3.5.3 The HASP Project 63
3.5.4 Multiple Blackboard Panels 65
3.5.5 Generalization 66
3.5.6 Summary 68
3.6 Data Classification 69
3.6.1 Introduction 69
3.6.2 Interval Classification 69
3.6.3 Fuzzy Set Theory for Data Classification 70
3.6.4 Classification Using Probability Distributions 74
3.6.5 Summary 78
3.7 Treatment of Uncertainty 78
3.7.1 Introduction 78
3.7.2 MYCIN's Certainty Factor Model 79
3.7.3 Bayesian Methods 81
3.7.3.1 Introduction 81
3.7.3.2 Bayesian Updating in a Hierarchical Hypothesis Space 82
3.7.4 The Dempster-Shafer Theory 84
3.7.5 Possibility Theory 86
3.7.6 Discussion 87
3.8 Explanation and Query Handling 89
3.8.1 Introduction 89
3.8.2 Types of Query for Knowledge-Based Systems 90
3.8.3 Understanding Queries 92
3.8.3.1 Introduction 92
3.8.3.2 Handling Queries with PROLOG 93
3.8.4 Generating Textual Output 94
3.8.5 Summary 95
3.9 Knowledge Acquisition 95
3.9.1 Introduction 95
3.9.2 Knowledge Elicitation Techniques 97
3.9.2.1 Introduction 97
3.9.2.2 Interview Strategies 98
3.9.2.3 Observational Strategies 99
3.9.3 Computer Aids in Knowledge Acquisition 101
3.9.3.1 Introduction 101
3.9.3.2 Knowledge Base Editors and Browsers 102
3.9.3.3 Domain Dependent Knowldege Acquisition Tools 103
3.9.3.4 Domain Independent Knowledge Acquisition Tools 105
3.9.4 Summary 105
3.10 Summary 106

PART TWO

4. The Evolution of a Knowledge-Based System 107
4.1 Introduction 107
4.2 An Initial Phase of Knowledge Acquisition 108
4.2.1 Introduction 108
4.2.2 Structured Interview Sessions 110
4.2.3 Summary 113
4.3 Design of a Knowledge-Based System for Laboratory Data Interpretation 114
4.3.2 The Blackboard Diagnostic Module 115
4.3.3 A Method for Updating Belief in a Hierarchy of Hypotheses 117
4.3.3.1 Introduction 117

4.3.3.2 A Method for Evidence Handling 118
4.3.3.3 Discussion 121
4.3.4 Laboratory Data as Evidence 122
4.4 FRAMEBUILDER: A Knowledge Editing Environment 124
4.4.1 Introduction 124
4.4.2 Overview of FRAMEBUILDER 125
4.4.3 Representation of Primitive Objects 126
4.4.4 Representation of a Structured Hypothesis Space 131
4.4.5 Frame Representation 133
4.4.6 Summary 137
4.5 Knowledge Acquisition Using FRAMEBUILDER 138
4.6 Summary 141

5. Implementation of a Knowledge-Based System 142
5.1 Introduction 142
5.2 Blackboard Representation and Control 144
5.3 Patient Data 148
5.3.1 Data Structures 148
5.3.2 Data Transfer From Database to Blackboard 149
5.3.3 Truth Maintenance 150
5.3.4 Data Derivation 151
5.3.5 Data Tranfer From Blackboard to Database 152
5.3.6 Data Classification 152
5.4 Handling Evidence 153
5.4.1 Overview 153
5.4.2 Signs/Symptoms and History 155
5.4.3 Laboratory Data Variables as Evidence 156
5.4.4 Relationships as Evidence 157
5.5 Combining Hypotheses 159
5.6 Diagnosis Critique 161
5.7 Dialogue Interaction 163
5.7.1 Introduction 163
5.7.2 Textual Presentation of Diagnoses 163
5.7.3 Handling User Queries 166
5.7.3.1 Overview 166
5.7.3.2 Information Requests 167
5.7.3.3 Suppositions 167
5.7.3.4 Explanations 169
5.7.3.5 Finding The Effect of Evidence 171
5.8 Summary 172

PART THREE

6. Evaluation of a Knowledge-Based System 173
6.1 Introduction 173
6.2 Evaluation in Clinical Practice 173
6.3 Evaluation of Prototype Systems 175
6.3.1 Introduction 175
6.3.2 Selecting Test Cases 177
6.3.3 Comparison With a Gold Standard 177
6.3.4 Expert Critique 178
6.3.5 Measuring Diagnostic Accuracy 179
6.3.6 Discussion 181
6.4 Bench Tests 182
6.4.1 Introduction 182
6.4.2 Performance With Laboratory Data 183
6.4.3 Performance With Incomplete Data 184
6.4.4 An Example Dialogue 186

6.4.5 Evaluation of System Speed 188
6.5 Evaluation of a Knowledge-Based System for Blood Gas Analysis 189
6.5.1 Introduction 189
6.5.2 Analysis of Results 190
6.6 Summary 195

7. Evaluation in a Second Application Domain 196
7.1 Introduction 196
7.2 Knowledge Acquisition 196
7.2.1 Introduction 196
7.2.2 First Session 197
7.2.3 Second Session 199
7.2.4 Third Session 200
7.3 Evaluation 201
7.4 Discussion 203
7.5 Summary 204

8. Conclusions 205

References 208

APPENDICES

I. Blood Gas Analysis and Acid-Base Balance 224
A1.1 Introduction 224
A1.2 Physiology 224
A1.3 Data Measurements in Blood Gas Analysis 226
A1.4 Acid-Base Disorders 227
A1.5 Bibliography 228

II. Computer-Aided Interpretation of Blood Gas Data 229
A2.1 Introduction 229
A2.2 Early Systems 230
A2.3 Algorithm-Based Systems 231
A2.4 Nomogram-Based Systems 231
A2.5 Intelligent Systems 232
A2.6 Summary 233

III. Knowledge Base for Blood Gas Interpretation 234

IV. Evaluation of the Blood Gas System: Case Data 238

V. Lipid Physiology and Hyperlipidaemia 246
A5.1 Physiology 246
A5.2 Measurements and Observations 247
A5.3 Hyperlipidaemia 247

VI. Knowledge Base for Hyperlipidaemia 249

VII. Evaluation of the Hyperlipidaemia System: Case Data 251

VIII. An Overview of PROLOG 257
A8.1 Foundations 257
A8.2 Programming in PROLOG 258

IX. Program Listings 261

LIST OF TABLES

Table 2.1	Translation of Linguistic Terms in CADIAG-2	38
Table 2.2	Example Sets of Manifestations and Causes	39
Table 2.3	Building M+, SCOPE and FOCUS	40
Table 3.1	Conceptual Categories of Questions	90
Table 3.2	Query Types in EMYCIN	91
Table 3.3	Questions for Knowledge Acquisition	98
Table 4.1	Relationships for Data Derivation	138
Table 4.2	Compensation Limits	139
Table 4.3	Signs and Symptoms	140
Table 4.4	Probability Assignments for Signs and Symptoms	140
Table 5.1	Knowledge Sources in the Diagnostic Module	147
Table 5.2	Translation of Numerical Belief Measures	164
Table 5.3	Forms of Differential Diagnosis List	164
Table 5.4	Rules for the Output of Diagnoses	165
Table 5.5	Information Requests	167
Table 5.6	Suppositions	167
Table 5.7	Explanation Requests	169
Table 5.8	Requesting the Effect of Evidence	171
Table 6.1	Performance Indices in the Abdominal Pain System	174
Table 6.2	The Evaluation of PUFF	178
Table 6.3	Evaluation of Some Knowledge-Based Systems	182
Table 6.4	Timing of Diagnosis on Ten Test Cases	188
Table 6.5	Summary of Diagnoses Made	190
Table 6.6	Agreement on Diagnosis	191
Table 6.7	Development Expert as Gold Standard	194
Table 6.8	Senior Clinician as Gold Standard	194
Table 7.1	Disease States for Secondary Hyperlipidaemia	197
Table 7.2	Summary of Signs and Symptoms for Hyperlipidaemia	199
Table 7.3	Summary of Diagnoses for Hyperlipidaemia	202
Table 7.4	Breakdown of Results by Diagnosis	203
Table A4.1	Summary of Diagnoses for Acid-Base Balance	243
Table A7.1	Summary of Diagnoses for Hyperlipidaemia	255

LIST OF FIGURES

Figure 2.1	General Model of a Knowledge-Based System	17
Figure 2.2	Overview of MYCIN	19
Figure 2.3	The MYCIN Context Tree	21
Figure 2.4	A Three-Level Network in CASNET	24
Figure 2.5	The Structure of a Frame in PIP	27
Figure 2.6	Property Types in the INTERNIST-1 Knowledge-Base	29
Figure 2.7	The Three-Level Disease Description in ABEL	34
Figure 2.8	An Augmented Decision Network in ATTENDING	36
Figure 2.9	MUNIN's Graphical Interface	43
Figure 2.10	A Causal Network in CHECK	45
Figure 3.1	A Rule-Based System	49
Figure 3.2	The Structured Representation of a Rule	50
Figure 3.3	The Rete Match Algorithm	52
Figure 3.4	A Rete Network for Checking Premise Conditions	53
Figure 3.5	A Frame Taxonomy	54
Figure 3.6	A Simple Semantic Network	57
Figure 3.7	The Proposition Node for the Reaction Between H_2O and CO_2	57
Figure 3.8	A Partitioned Network Containing Figure 3.7 as a Space	58
Figure 3.9	Demons in Pandemonium	60
Figure 3.10	Blackboard Levels and Knowledge Sources in Hearsay II	61

Figure 3.11	The Structure of Hearsay II	62
Figure 3.12	The HASP/SIAP System	64
Figure 3.13	Three-Interval Classification Using 95% Confidence Limits	70
Figure 3.14	Seven Interval Classsification of pH	70
Figure 3.15	Membership Functions for Low, Normal and High pH	72
	a) Using Zadeh's Functions	
	b) Using a Simple Ramp Function	
Figure 3.16	Increasing the Fuzzy Sets for Classification	74
	a) Defining Further Ramp Functions	
	b) Using Linguistic Hedges	
Figure 3.17	Probability Distributions in MUNIN	75
Figure 3.18	Inter- and Intra-Individual Differences	76
Figure 3.19	Probability That Observed Value is Normal	77
Figure 3.20	A Hierarchical Hypothesis Space	82
Figure 3.21	A Hierarchy of Subsets	85
Figure 3.22	Handling a Dialogue Between Computer and Clinician	89
Figure 3.23	A Parse Tree	92
Figure 3.24	PROLOG Grammar Rules	93
Figure 3.25	PROLOG Clauses Translated From Grammar Rules	93
Figure 3.26	Definite Clasue Grammar Extension	94
Figure 3.27	Knowledge-Based System Development Cycle	96
Figure 3.28	Repertory Grids	104
	a) Simple Yes/No Rating	
	b) Rating on a Scale of 1-10	
	c) Rating Based on a Ranking Scheme	
Figure 4.1	Taxonomies of Acid-Base and Hypoxaemic State Disorders	108
Figure 4.2	Preliminary Design for a Knowledge-Based System	109
Figure 4.3	Structure of the First Interview Session	110
Figure 4.4	Structure of the Second Interview Session	111
Figure 4.5	A Simplified Classification of Disorders	112
Figure 4.6	The Third and Fourth Interview Sessions	112
Figure 4.7	The Fifth Interview Session	113
Figure 4.8	Design of a Knowledge-Based System: Overall Structure	115
Figure 4.9	Design of the Blackboard Diagnostic Module	116
Figure 4.10	A Hierarchy of Hypotheses with Assigned Evidence	119
Figure 4.11	The Probability of a High, Normal or Low Value	123
Figure 4.12	Comparison with Tango's Method of Data Classification	123
Figure 4.13	The Structure of FRAMEBUILDER	125
Figure 4.14	FRAMEBUILDER as a Frame-Based Organization of Knowledge	126
Figure 4.15	The Main Command Line in FRAMEBUILDER	126
Figure 4.16	Primitive Data Objects: Laboratory Data	127
	a) FRAMEBUILDER Display	
	b) PROLOG Representation	
Figure 4.17	Primitive Data Objects: Data Derivation Relationships	128
	a) FRAMEBUILDER Display	
	b) PROLOG Representation	
Figure 4.18	Primitive Data Objects: Signs and Symptoms	129
	a) FRAMEBUILDER Display	
	b) PROLOG Representation	
Figure 4.19	Primitive Data Objects: Patient History	130
	a) FRAMEBUILDER Display	
	b) PROLOG Representation	
Figure 4.20	Primitive Data Objects: Disorder Classes	131
	a) FRAMEBUILDER Display	
	b) PROLOG Representation	
Figure 4.21	The Frame Hierarchy	132
	a) FRAMEBUILDER Display	
	b) PROLOG Representation	

Figure 4.22	Frame Instances: Laboratory Data a) FRAMEBUILDER Display b) PROLOG Representation	133
Figure 4.23	Frame Instances: Data Relationships a) FRAMEBUILDER Display b) PROLOG Representation	134
Figure 4.24	Frame Instances: Signs and Symptoms a) FRAMEBUILDER Display b) PROLOG Representation	135
Figure 4.25	Frame Instances: Patient History a) FRAMEBUILDER Display b) PROLOG Representation	136
Figure 4.26	The Revised Hypothesis Hierarchy for Acid-Base Disorders	139
Figure 5.1	The Implemented Design of the Blackboard Diagnostic Module	142
Figure 5.2	PROLOG Representation of Blackboard Data Entries	143
Figure 5.3	PROLOG Representation of Knowledge Sources	144
Figure 5.4	The Control Cycle	145
Figure 5.5	Control Cycle with Improved Efficiency	146
Figure 5.6	PROLOG Representation of Patient Data	148
Figure 5.7	Transfer of Data Between Files, Database and Blackboard	149
Figure 5.8	Mapping of Data From Database to Blackboard	150
Figure 5.9	Pre-processing for Data Derivation	151
Figure 5.10	Data Mapping Achieved by derive-data Knowledge Source	151
Figure 5.11	Mapping of Data From Blackboard to Database	152
Figure 5.12	Data Mapping Achieved by classify_data Knowledge Source	153
Figure 5.13	Pre-processing For Hypothesis Hierarchies	154
Figure 5.14	Algorithm for Evidence Handlers	154
Figure 5.15	Finding P(elhi) For Diseases and Disorders Added as History	155
Figure 5.16	Calculation of P(elhi) for Classified Data Variables	156
Figure 5.17	Pre-processing for Relationships	157
Figure 5.18	Algorithm for Calculating P(elhi) for Relationship Evidence	158
Figure 5.19	Algorithm for sum-hypothesis Knowledge Source	159
Figure 5.20	Combining Hypotheses at the sub-diagnosis Level	160
Figure 5.21	Action of predict_disorders Knowledge Source	161
Figure 5.22	Action the critique_diagnosis Knowledge Source	162
Figure 5.23	From Output Data Structures to Text	163
Figure 5.24	Algorithm for Processing Suppositions	168
Figure 5.25	Algorithm for Explaining a Diagnosis	170
Figure 5.26	Finding the Updating Factor for a General Hypothesis Node	171
Figure 6.1	Diagnosis Based on a Single Laboratory Data Variable	183
Figure 6.2	Form for Gathering Evaluation Results	189
Figure 7.1	Hypothesis Hierarchy for Hyperlipidaemia	197
Figure 7.2	Hierarchies for Cholesterol and Triglyceride Levels	198
Figure 7.3	Blood Pressure Level	199
Figure 7.4	Coronary Risk Factor	200
Figure 7.5	Elicitation Grid for Family History	200
Figure 7.6	Elicitation Grid for IHD/PVD	201
Figure A8.1	A Simple PROLOG Program	258

ACKNOWLEDGEMENTS

I would like to acknowledge the following people who have help with this work over the last three years.

My supervisors at City University were Dr Tim Ellis and Prof Ewart Carson who sparked my initial interest and kept me going in the right direction.

At the Royal Free Hospital, Dr Derek Cramp, Dr Bob Simons, Dominic Cox and other members of the Department of Medical Informatics, the Department of Anaesthesiology and the Intensive Care Unit.

Dr Paul Collinson from the Department of Clinical Biochemistry at the West Middlesex Hospital provided his knowledge and great enthusiasm in the latter stages of the work.

All my friends and colleagues from the Department of Systems Science at City University who made my time there so enjoyable. George Zarkadakis helped me with knowledge acquisition at the Royal Free Hospital and Chris Stevens encouraged me to use the Apple Macintosh. I owe special thanks to Ron Summers for his support and guidance throughout the three years.

Finally, I would like to thank Pam Woodcock, my parents, family and friends who supported me during long periods of absent mindedness and absent finances.

DECLARATION

I grant powers of discretion to the University Librarian to allow this thesis to be copied in whole or in part without further reference to me. This permission covers only single copies made for study purposes subject to normal conditions of acknowledgement.

ABSTRACT

Faced with a rapid increase in the amount of data available to form the basis of diagnostic and management decisions in intensive care medicine, clinicians will require the assistance of computers, and in particular knowledge-based systems, if they are to avoid being overwhelmed. The development of knowledge-based systems in medicine can be traced through three generations, starting with systems that reasoned in an *ad hoc* fashion using surface level knowledge and progressing to systems that attempted to capture deeper knowledge of their specialist domains. More recently, a third generation of systems has emerged that use more rigorous methods of reasoning with surface level knowledge as a platform for the application of deeper knowledge.

A knowledge-based system has been developed for the diagnosis of disorders of acid-base balance and hypoxaemic state using a blackboard architecture to control processes of data classification and belief updating in a hierarchy of hypotheses. To acquire the knowledge for the diagnostic system, a knowledge editing environment was developed and the two systems have been combined to form a set of domain independent tools. These tools were applied to a second domain - the diagnosis of hyperlipidaemia.

Evaluation studies performed in the domains of acid-base balance and hyperlipidaemia have shown that the system performs at a level comparable to that of an expert clinician.

Cogito, ergo sum.
- René Descartes,
Le Discours de la Méthode, 1637.

Je ne pense pas donc je suis une moustache.
- Jean-Paul Sartre,
La Nausée, 1938.

PART ONE

CHAPTER ONE

INTRODUCTION

1.1 Background

In 1950, Alan Turing posed the question *Can computers think?* - almost 40 years later the question remains unanswered. In many respects Turing's question was a hypothetical one, prompted more by intellectual curiosity than practical necessity and posed at a time when debugging a computer program involved the removal of insects from electronic valves. However, the search for a machine-based intelligence has assumed a greater significance in recent years. The computational power of computers has risen to a level at which the concept of a thinking machine has become a realistic possibility, and at the same time the practical necessity of such a machine has begun to emerge. In the field of medicine, the advent of the computer age has brought a remarkable change to clinical practice, with an explosion in the number of data available to form the basis of diagnostic and management decisions. Yet, in the face of an enormous increase in the complexity of the decisions he must make, the average clinician continues to rely on his own powers of reasoning and judgement and on the accumulation of knowledge across a wide range of disciplines. To avoid being overwhelmed, it is becoming necessary for clinicians to look to computers for assistance with medical decision making.

One of the earliest mechanical aids to differential diagnosis was a device called the Group Symbol Associator, which resembled a slide-rule and was capable of matching patterns of symptoms with diseases (Nash, 1954). Within a few years, pattern matching using digital computers was being applied to medical diagnosis. In one system, a three symbol logic (1 true, 0 false, ? unknown) was used as a basis for matching strings of patient symptoms with disease characteristics to reach a diagnosis and to make prognostic and therapeutic suggestions (Paycha, 1958).

The possibility of using computers for medical diagnosis prompted an examination of the foundations of the reasoning processes involved (Ledley & Lusted, 1959). Whilst the importance of heuristic reasoning, based on the accumulated experience of the clinician, was recognized, it was maintained that the basis of diagnosis is the logical combination of medical knowledge and patient observations. The categorical statements of formal logic can be augmented by the use of probabilistic reasoning. Once logic has reduced the number of

diagnostic hypotheses to a manageable size, probabilistic techniques can be used to find the most likely diagnosis. Ledley & Lusted advocated the use of Bayes' Theorem to transform conditional probabilities of symptoms given disease states (which they assumed to be readily available) to the conditional probability of disease hypotheses given the symptoms observed in a particular patient:

$$P(D|S) = \frac{P(D).P(S|D)}{P(S)}$$

or in other words the probability of a disease hypothesis D, given a set of observed symptoms S is equal to the *a priori* probability of D multiplied by the conditional probability of the symptoms given the disease state divided by the *a priori* probability of the symptoms. Following the lead of Ledley & Lusted, and utilizing ever more powerful computers, many researchers began to investigate the application of Bayes' Theory and logical pattern matching to problems of medical diagnosis (Croft, 1972).

By the late 1960s, the art of medicine had, in the clinical laboratory at least, been transformed into a science. Some argued that all of clinical medicine could be similarly transformed and that clinicians should operate within a framework based on mathematics and logic (Card, 1970). Obviously computers would play an important role in such a science of medicine. Others, though strongly advocating the use of scientific principles in medicine, argued that an essential element of the art of medicine could not be captured by the type of logic used in digital computing (Feinstein, 1967). Nevertheless, it was predicted that computers would become increasingly prominent in clinical practice, taking over the scientific aspects of data collection, diagnosis and therapy selection, and freeing clinicians to devote more time to the art of medicine (Schwartz, 1970).

At the start of the 1970s it had become clear that the expert level of performance displayed by a number of computer diagnosis programs would not be sufficient to bring about their use in clinical practice; clinicians were not willing to adapt their reasoning processes, in the manner envisaged by Card, so as to accommodate the computer. If computers were to be used as diagnostic assistants then the reasoning processes that *they* employed would have to be adapted to accommodate *clinicians*. Such thinking led to the development of a new breed of knowledge-based diagnosis systems that attempted to model more closely the way in which human clinicians reason, drawing on methods from the field of artificial intelligence which had begun to emerge as a scientific discipline at about the same time as Ledley and Lusted were analyzing the reasoning foundations of medicine. These systems employed both categorical and probabilistic reasoning (Szolovits & Pauker, 1978) as had been suggested by Ledley & Lusted, but were also able to explain the reasons for their

conclusions and advice in a manner that had not been possible with the systems based purely on Bayes' Theory or pattern matching techniques.

The first generation of knowledge-based systems in medicine were intended primarily as exploratory research projects and none were accepted for routine clinical use. However, two systems developed with tools that emerged from the first generation systems - PUFF (Aikins *et al*, 1983) and SPE (Weiss *et al*, 1981) - are currently in routine use and, although there are still some major problems (not least in the legal implications of computer diagnosis), such successes have led to renewed predictions that computers will play an important role in medical diagnosis provided that they are made quick and easy to use (de Dombal, 1987) and once the cost of the technology on which they are based falls to a level affordable by the majority of clinicians (Kulikowski, 1988).

Nowhere has the explosion in measured data, caused by advances in technology, been greater than in intensive care medicine. In one Intensive Care Unit (ICU), the number of data measured routinely for each patient rose from 37 to 94 in the period from 1970 to 1986, with over 100 data being recorded in the initial phases of patient management (Price & Mason, 1986). The technological solution to this problem has been the development of patient data management systems such as the hospital-wide system called HELP at the Latter-Day Saints Hospital in Utah (Warner *et al*, 1971) and the dedicated ICU system at Kuopio, Finland (Kari *et al*, 1988). An obstacle to the widespread use of such systems has been the length of time required to input data, but technological solutions are emerging in the form of light pens, touch sensitive screens, hand-held remote keypads and customized graphics tablets (Collinson *et al*, 1988).

The availablity of on-line data from a patient data management system provides the opportunity to incorporate computer-based advice without the need for further interaction between computer and clinician. Several systems have already been developed that work in this way with the HELP system (Gardner *et al*, 1975; Sittig, 1987). Although methods of multivariate cluster analysis are currently being applied to the type of quantitative data available from patient data management systems or directly from laboratory equipment (*eg* Coomans *et al*, 1984) it seems that they suffer the same weaknesses as the diagnostic systems developed in the 1960s.

The requirement, therefore, is for knowledge-based systems that can reach an initial diagnosis based only on quantitative laboratory data. Such systems are likely to be accepted into routine clinical use, as has been demonstrated by PUFF and SPE which are interfaced directly with laboratory equipment, the latter being incorporated on a microchip within the equipment itself. Two of the early knowledge-based systems have also been applied to the interpretation of laboratory data: INTERNIST-1 has demonstrated an accuracy of 50%

(16/32) in advising clinical pathologists on the results of laboratory tests (Myers, 1986) and the EXPERT model has been used to construct a system for the interpretation of biochemical and haematological tests in an out-patient clinic (van Lente *et al*, 1986). Neither INTERNIST-1 nor EXPERT were designed specifically for the interpretion of laboratory data and the difficulties encountered with the implementations mentioned above suggest that knowledge-based systems should be developed for the specific purpose of laboratory data interpretation.

1.2 Objectives

This thesis describes the design, development and evaluation of a knowledge-based system for diagnosis of physiological disorders based on the interpretation of laboratory data. The intended users would be qualified clinicians, not expert in the specialist domain of the system. Initially, disorders of acid-base balance and hypoxaemic state were chosen as the target for diagnosis, based on the results of blood gas analysis (see Appendix I). This is an area of medical diagnosis that causes considerable problems for non-expert clinicians and a number of computer-aided diagnosis systems have already been developed, using conventional programming techniques (see Appendix II). In order to make the system as portable as possible, it was developed on an IBM-compatable microcomputer which has the facility to interface with laboratory equipment through an RS232 interface. Although the system could be interfaced directly with an automatic blood gas analyser, this was seen to be a technological problem and was not addressed in the context of this project.

The overall objective of this work was to apply knowledge-based techniques to the interpretation of data in intensive care medicine through the development of a knowledge-based system for the diagnosis of disorders of acid-base balance and hypoxaemic state. Specific objectives were:

(1) to make the system easy to use and to keep interaction between computer and clinician to a minimum

(2) to combine different methods of knowledge representation and reasoning within a single system

(3) to provide detailed explanation of the system's conclusions and of the domain of acid-base balance in general

(4) to make the system's knowledge easily accessible to clinicians for review and update

As will be described in Chapter 4, the utility of a general, domain independent, system for diagnosis based on laboratory data was recognized during the early stages of the project. As a result, the emphasis was placed on the creation of a set of domain independent tools with which to achieve the initial objectives set out above.

1.3 Outline of the Thesis.

The thesis is divided into three parts: the first deals with background issues in the development of a knowledge-based system for medical diagnosis, the second with the design and implementation of the prototype of such a system and the third with the evaluation of that prototype.

In Part One, Chapter 2 reviews some of the existing knowledge-based systems for medical diagnosis, introducing the main issues involved in their design and development and identifying the shortcomings that have prevented all but a few from entering into routine clinical use. Chapter 3 discusses in more detail the relevant issues of knowledge representation, manipulation and control introduced in Chapter 2 and concludes with an analysis of the methods by which knowledge can be acquired for incorporation into a knowledge-based system.

In the second part of the thesis, Chapter 4 describes the early phases of knowledge acquisition in the domain of acid-base balance, explaining how the original objectives of the project were modified to encompass the development of a set of domain independent tools. The design is then presented for the first of these tools - a knowledge-based diagnostic system, featuring a method of belief updating in a hierarchically organized set of hypotheses. The second tool developed is a knowledge editing environment, called FRAMEBUILDER, which allows a knowledge base to be created, browsed and edited through a graphical interface. Chapter 4 concludes with a description of FRAMEBUILDER, the structure of the knowledge base it creates and its use in knowledge elicitation sessions at two hospitals. Chapter 5 focusses on the implementation of the diagnostic system using an IBM compatable microcomputer and the logic programming language PROLOG (these were also used for the implementation of FRAMEBUILDER).

Part Three deals with the evaluation of the tools described in Part Two. Chapter 6 reviews some of the issues in the evaluation of knowledge-based systems in medicine and concludes with a description of the evaluation of the diagnostic system described in Chapters 4 and 5, applied to the diagnosis of disorders of acid-base balance and hypoxaemic state. Chapter 7 describes the application of both tools to the development of a knowledge-based system for the diagnosis of hyperlipidaemia and Chapter 8 presents the conclusions drawn from the work. Background information about the medical domains of acid-base balance and hyperlipidaemia, a review of computer-aided decision support programs for the interpretation of blood-gas data, an overview of the logic programming language PROLOG, details of the results of the system evaluation and listings of the suite of programs developed are included as appendices.

CHAPTER TWO

KNOWLEDGE-BASED SYSTEMS FOR MEDICAL DIAGNOSIS

2.1 Introduction

This chapter reviews the evolution of knowledge-based systems in medicine and highlights some of the interesting features of the landmark systems. Before embarking upon such a review, it seems appropriate to define what is meant by the term *knowledge-based system.* At the most fundamental level, a knowledge-based system is *a knowledge-intensive program that solves problems that normally require human expertise* (Hayes- Roth, 1984). Leaving aside arguments about the definition of the word *knowledge*, it would seem that a great many computer programs could lay claim to the title of knowledge-based system on the basis of this broad definition. There are, however, some characteristics common to all knowledge-based systems which are not generally found in other computer programs, regardless of how complex they may be. Knowledge-based systems...

...solve problems and operate in a role normally undertaken by a human expert

...generate and reason with symbolic descriptions of data

...reason in an opportunistic fashion, persuing goals and strategies relevant to a particular situation, instead of following a predetermined reasoning path (however complex that might be)

...explicitly represent knowledge about the domain in which they operate, as distinct from the strategies for applying that knowledge.

...can function with uncertain or incomplete data

...can entertain multiple, possibly competing hypotheses and reach conclusions that may not be categorically proven

...can explain their reasoning and justify their solutions.

Structurally, most medical knowledge-based systems can be viewed at some level of their design along the lines shown in Figure 2.1.

It is clear from the characteristics listed above, that the computer programs developed in the 1960s for medical diagnosis were not knowledge-based systems in any respect other than that they were performing tasks normally performed by human specialists. Generally, these programs operated either by making statistical predictions based on large amounts of

recorded data (see Croft, 1972 for a review) or by following rigidly prescribed protocols in which knowledge of the domain and procedures for its application were thoroughly intermingled (*eg* Bleich, 1969).

The failure of early computer programs to be accepted by clinicians can be attributed to their inflexibility and tendency to produce solutions 'out of a hat', without any justification or explanation of their reasoning process. By the start of the 1970s, however, the potential for computers to play a significant role in the processes of information retrieval and decision making in medicine had been recognized (Schwartz, 1970) and a new breed of systems was about to emerge. At four institutions in the United States (Stanford University, Rutgers University, the University of Pittsburgh and the Massachussettes Institute of Technology) work began on the first generation of medical knowledge-based systems, which have been followed, roughly speaking, by two further generations (Kulikowski, 1988). The following sections trace the evolution of medical knowledge-based systems through these three generations and introduce the important issues of knowledge acquisition, representation and manipulation which have emerged. The four major first generation systems are reviewed in some detail, since they introduce many of the themes that will recur throughout this thesis; thereafter attention will be focussed on systems that introduced new and interesting ideas.

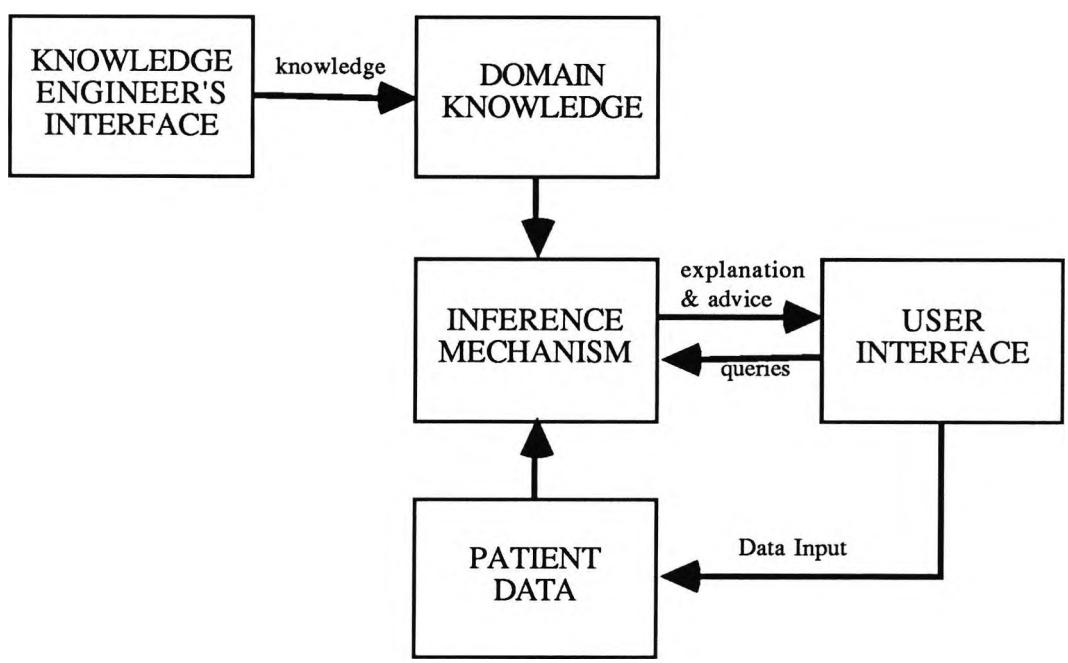


Figure 2.1 General Model of a Knowledge-Based System. Domain knowledge is input through the knowledge engineer's interface. The user inputs data and queries to receive advice and explanation.

2.2 Early Systems

2.2.1 Introduction

The first generation of knowledge-based systems in medicine, developed at the four institutions mentioned above, encompassed the three knowledge representation schemes that have formed the basis of many subsequent systems (rule-based, frame-based and causal/semantic network). The emphasis, then, was on the exploration of fundamental methods for acquiring knowledge from expert clinicians, explicitly representing this knowledge and reasoning with it in a manner that could be satisfactorily explained to the systems' users. Given the basic level of the research and the computing resources available (typically mainframe computers with teletype terminals) it does not seem surprising that none of the early systems found their way into routine clinical use. The success of these systems lies in the way in which they set the guidelines for the development of the subsequent generations.

2.2.2 The MYCIN System

The original MYCIN system (Shortliffe *et al*, 1973) was implemented with the aim of advising clinicians about antimicrobial therapy and highlighting research issues in applied artificial intelligence; specifically in the areas of knowledge acquisition, representation and inference. The structure of the system is shown in Figure 2.2, and it can be seen that it comprised three main sub-programs: the consultation system, whose operation is outlined below, the explanation system and a rule-acquisition system. These latter two became research projects in their own right (Scott *et al*, 1977; Davis, 1979); indeed MYCIN provided the basis for over a decade of research at Stanford Univerity (Buchanan & Shortliffe, 1984).

MYCIN reasoned about the patient, cultures grown, organisms isolated, operative procedures undergone and drugs and therapies administered. These were organized into a hierarchy of CONTEXTS, and during a consultation, specific nodes were created as instances of each type of CONTEXT. Each node was charcterized by the values of clinical parameters, of which there were 65 in the original system. Parameters were of three types: *single-valued* parameters could take one of several mutually exclusive values (*eg* organism identity); *multi-valued* parameters could have more than one value at a given time (*eg* drug allergies); *yes-no* parameters were either true or false (*eg* adequate drug dose). As part of its static knowledge base, MYCIN held information about the properties of each clinical parameter: its expected value (*ie* numeric, yes/no, etc), prompts (sentences used to ask the user about the parameter), a condition to be evaluated before asking the user to input its

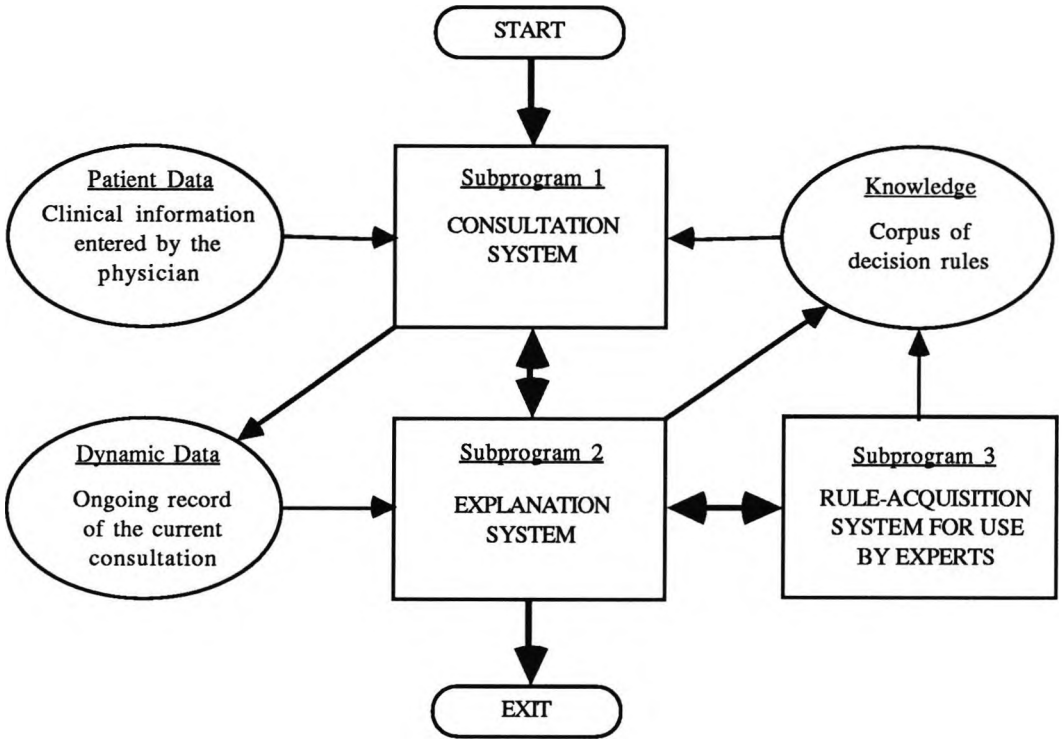


Figure 2.2 Overview of MYCIN. The flow of control is shown by heavy arrows, the flow of information by light arrows. (From Shortliffe *et al*, 1975).

value, whether it was laboratory data, a list of rules that inferred its value, a list of rules that referenced it without updating its value, its units of measurement and an English language description. A list of properties was also stored for each CONTEXT type.

The main body of the static knowledge base was made up of four types of representation structure: a *dictionary* was used for language understanding; *lists* were used to group knowledge in easily handled units (*eg* ORGANISMS was a list of all known organisms); *tables* contained information about clinical parameters (*eg* one table held the gram stain, morphology and aerobicity of each organism); *rules* contained the knowledge used to make inferences.

The rules were grouped into categories according to the CONTEXT about which they made inferences. There were about 200 rules in the original system and about 500 by 1978. Each rule consisted of a premise and an action. The *premise* was a conjunction of conditions which were themselves functions that could incorporate logical *or* or *not*. Disjunction of conditions was not allowed; this was handled by having two or more rules with the same action. The *action* typically concluded the value of a clinical parameter and had associated with it a *certainty factor* (CF) which indicated the strength of the conclusion given that the premise of the rule was true. An example of one of MYCIN's rules is:

RULE 037

IF The stain of the organism is gramneg

AND The morphology of the organism is rod

AND The aerobicity of the organism is anaerobic

THEN There is suggestive evidence (.6) that the class of the organism is bacteroides

(adapted from Shortliffe *et al*, 1975)

During a consultation with MYCIN, a dynamic knowledge base was created, consisting of a context tree, hypotheses about the values of the clinical parameters characterizing nodes in the tree and a record of the consultation (*ie* the questions asked and the rules invoked). At the start of a consultation the node for PATIENT was created as the root of the context tree and the following *goal rule* was considered:

RULE 092

IF There is an organism which requires therapy

AND Consideration has been given to the possible existence of additional organisms requiring therapy, even though they have not actually been recovered from any current cultures

THEN Compile the list of possible therapies which, based on sensitivity data, may be effective against the organisms requiring treatment

AND Determine the best therapy recommendations from the list

(adapted from Shortliffe *et al*, 1975)

MYCIN built up its dynamic knowledge base by backward chaining through its rule-base: in order to reach the goal in Rule 092, the premise conditions had to be evaluated. Evaluating the conditions usually required the values of certain clinical parameters; if some of these were unknown, then rules were invoked that concluded the values required; evaluating the conditions of these rules involved the invocation of further rules, and so on. This behaviour was achieved by using two simple procedures which called each other recursively. The MONITOR procedure took each condition in the premise of a rule, gathered the information needed to evaluate it, performed the evaluation and then either rejected the rule or moved on to the next condition, depending on whether the evaluation yielded false or true. To gather information that was not available on the database, the FINDOUT procedure was called. For laboratory data, the user was first asked to supply a value; if this could not be done, the list of rules which could be used to infer the required

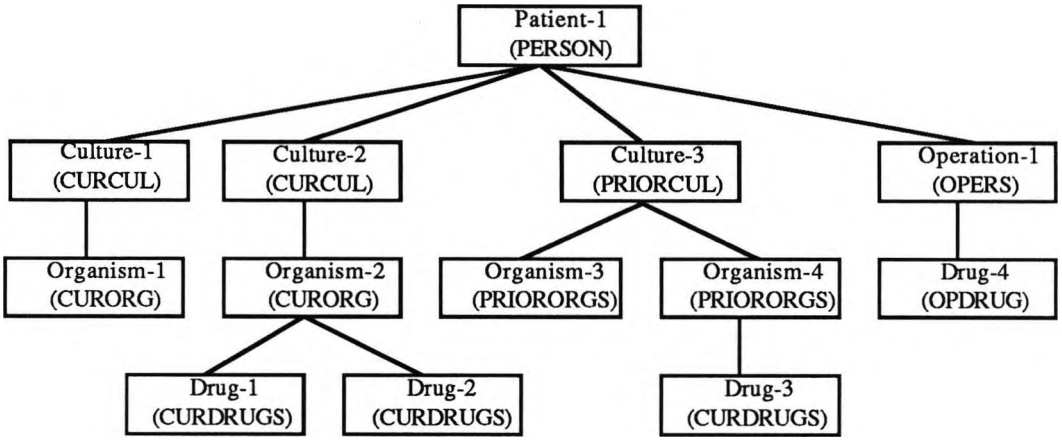


Figure 2.3 The MYCIN Context Tree. A patient with two recent cultures, one old culture and one recent operation. Context types are shown in brackets. (Adapted from Buchanan & Shortliffe, 1984; 84).

value was retrieved. The MONITOR procedure was then applied to each rule in the list until the value was returned. For other types of data, MONITOR was called immediately and the user was only questioned if the value could not be inferred from the rules invoked.

There were several ways in which nodes could be added to the context tree as it expanded from the PATIENT root node during a consultation. When a condition in the premise of a rule referenced a context type for which no nodes had been created, the context type was passed to the FINDOUT procedure which then asked the user to input nodes of that type. Nodes could also be created implicitly when an associated node was created (*eg* each organism had an associated culture) or when the action part of a rule was executed. A typical context tree for a consultation is shown in Figure 2.3.

Associated with each node of the context tree were hypotheses of the values of characteristic clinical parameters. Each hypothesis had a certainty factor which ranged from -1 (definitely untrue) to +1 (definitely true). The certainty factor could be input by the user when asked for the value of the parameter (CF=1 was assumed as the default) or it could be calculated when the value was inferred by a rule. In this case, the calculated certainty factor depended on the combined uncertainty of the conditions in the premise (called the TALLY) and the certainty factor associated with the action. Consider, for example, Rule 037 given above, with the data:

Val[ORGANISM-1,GRAM] = ((GRAMNEG 1.0))
Val[ORGANISM-1,MORPH] = ((ROD .8)(COCCUS .2))
Val[ORGANISM-1,AIR] = ((ANAEROBIC .6)(FACUL .4))

The TALLY of the premise would be .6 (the minimum of the certainty factors of the conditions) which would be combined with the certainty factor .7, to calculate the certainty factor of the hypothesis that the class of ORGANISM-1 was bacteroides. The method used to calculate the new certainty factor is discussed in Section 3.7.2. Since the premises of rules were programmed with linguistic terms (*eg* KNOWN, SAME, NOTSAME, MIGHTBE, etc) special functions were used to combine them with the numerical certainty factors in the knowledge base. For instance, the function for KNOWN returned *true* if the CF of the parameter on the database was more then 0.2, otherwise it returned *false*.

When MYCIN questioned the user during the execution of the FINDOUT procedure, one of the prompt properties of the parameter required was used to pose the question in the manner most appropriate to the prevailing situation. A set of questions was also asked whenever a new node was added to the context tree (the set was stored as a property of the CONTEXT type). Several levels of assistance were offered to aid users in answering questions: recognized responses could be listed, the rule generating the question could be displayed, detailed explanation of the question could be obtained by asking WHY, or access to the more detailed explanation facilities could be gained.

After the premise of the top goal had been evaluated, the dynamic database would contain various hypotheses as to the identity of the organisms present. Associated with each organism was a rule which listed possible drug therapies with a measure of the sensitivity of the organism to each drug. These rules were used to compile a list of potential therapies for the patient. The best therapy was selected by using a complex algorithm which took into account the sensitivity information, details of any existing therapy and a desire to prescribe the minimum number of drugs necessary (this part of the original MYCIN system did not use rule-based reasoning). Rule-based reasoning was used to prevent the use of drugs that were contra-indicated for a particular patient.

The success of the MYCIN project was measured more in terms of its contribution to applied artificial intelligence than to medicine. Nevertheless, its generalization as the knowledge-based system shell, EMYCIN (Van Melle, 1979), was used for the development of a system for the interpretation of pulmonary function test data (Aikins *et al*, 1983) which became one of the first (and still one of the only) knowledge-based systems used routinely in clinical practice.

2.2.3 CASNET

The CASNET model (Weiss *et al,* 1978) was proposed as a general model of diagnosis and was applied in practice to the diagnosis and treatment of the eye disease glaucoma. Knowledge about a patient was represented at three levels: *observations, physiological states* and *categorizations* for diagnosis, prognosis and treatment. The physiological states represented summaries of abnormal physiological events and were connected by causal links of varying strength to form a directed, acyclic graph. Diseases were represented by particular pathways in the causal network; the progression of a disease was modelled by adding further nodes to its pathway. Observations were connected with one or more physiological states by associative links of varying strength; any one state could have any number of associated observations. Figure 2.4 shows the three-level NETwork with its CAusal and ASsociative links (hence the name CASNET).

At the start of a consultation, each node, n_j, in the network of physiological states had an assigned measure of confidence, $Cf(n_j) = 0$. As observations were made, the associative weights between observations and states updated the Cf values in the following way:

If $|Cf(n_j)| < |a_{ij}|$ then $Cf(n_j)$ was set to a_{ij}

If $Cf(n_j) = -a_{ij}$ then $Cf(n_j)$ was set to zero until an observation with association $|a_{kj}| > |a_{ij}|$ was made

In other instances, $Cf(n_j)$ remained unaltered.

(where a_{ij} was the associative weight between the ith observation and the state n_j and $-1 \leq a_{ij} \leq 1$).

According to its Cf value, a node could be either *confirmed, denied* or *undetermined*:

If $Cf(n_j) > \Theta$ then n_j was confirmed

If $Cf(n_j) < -\Theta$ then n_j was denied

otherwise n_j was undetermined

(where Θ was a positive threshold value).

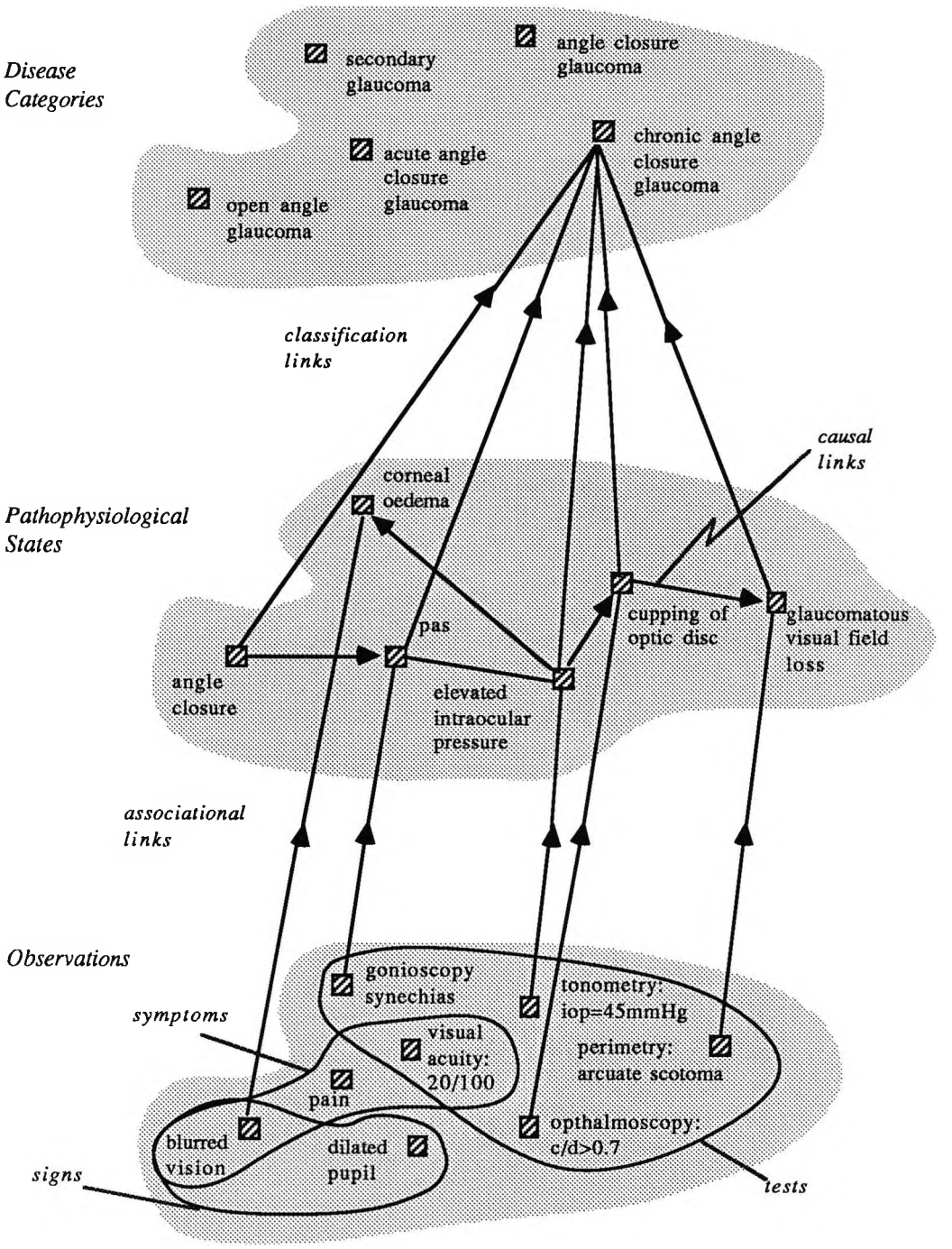


Figure 2.4 A Three-Level Network in CASNET (Adapted from Weiss *et al*, 1978)

Nodes in the network which had no causal antecendents were designated as *starting nodes*, and played an important role in identifying a patient's diseases. These starting nodes represented the basic causes or mechanisms of diseases - the set of most likely starting nodes competed to describe the disease(s) present for a particular patient. The progression of a disease was modelled by tracing an *admissible pathway* of nodes, in a causal chain that originated from one of the starting nodes. An admissible pathway was one that passed through confirmed or unconfirmed nodes only, without including any denied nodes. The classification of pathways of nodes into disease states was achieved by a set of tables which also indicated appropriate treatment plans for each disease. Consider for example, the following table:

(n25,D1,T1) (n26,D2,T2) (n30,D3,T3) (n31,D4,T4)

If an admissible pathway existed from n25 to n31 then disease D4 and treatment T4 were indicated; if the pathway only extended as far as n26 then D2 and T2 would be indicated. The tables could also contain rules which indicated that certain states should be denied for a particular disease to be present. Since treatment should depend not only on the identified disease, but also on the current therapy and observed contra-indications, the treatment plans contained ranked lists of therapies. Each therapy was linked to observations by *preference measures* which were used to score the possible therapy options in exactly the same way that the associative links were used to confirm physiological states.

Three possible strategies were developed to focus questioning of the user:

a set of questions could be asked when certain logical conditions were met

questions could focus on pathways classified as the most likely disease states as described above

the pathophysiological states could be weighted as described below, and the highest weighted state could then used to focus questioning

Two weights could be calculated for a node; its *forward weight* and its *inverse weight*. The forward weight measured the evidence accumulated by the causes of a node and was calculated by identifying all admissible pathways to the node, from starting or confirmed nodes, which did not pass through any other confirmed nodes. The weight of each pathway was the product of the causal link strengths along it; the weights for all possible paths were then added to give the forward weight. The inverse weight of a node was a measure of the belief in those nodes of which it was the cause The inverse weight of a node n_i, due to n_j (a confirmed node lying on a causal pathway from n_i) was proportional

to the forward weight of pathways passing through both n_i and n_j divided by the forward weight of all possible pathways through n_j. The overall inverse weight of n_i was then fixed as the maximum inverse weight from all possible confirmed effects. Finally, the node with the greatest weight, forward or inverse, was chosen to be the focus for questioning.

The developers of CASNET recognized the similarity between their model and a Markov model of diagnosis (*eg* Gheorghe *et al*, 1976) since the weight assignments of nodes were derived from their immediate causal antecedents. The relaxation of the conditions of exhaustivity and mutual exclusivity on the causal successors of a node, led to an enormous reduction in the size of model required (the vast number of probability assignments required by more formal statistical models remains one of their major limitations). CASNET introduced into the domain of computer-aided medical decision making, the notion of updating beliefs in networks of hypotheses, albeit in an *ad hoc* manner. The search for more formal methods of evidential updating in belief networks has since become an important area of artificial intelligence research (see Section 3.7). CASNET was also unique amongst the first generation systems in that it explicity represented knowledge about diseases at different levels of abstraction; an idea which was developed further in the system ABEL (Patil, 1981) which is discussed in Section 2.3.2.

2.2.4 The Present Illness Program

The first description of an artificial intelligence program to appear in a purely clinical journal was that of the Present Illness Program (Pauker *et al*, 1976). The emphasis of the project, which was developed at MIT and Tufts University School of Medicine, was to model the cognitive behaviour of a clinician as he took the history of a patient with oedema (an excessive retention of fluid by body tissues). Knowledge of diseases and clinical or physiological states was held in *frames* (see Section 3.3) which were considered as hypotheses for explaining findings about the patient. The structure of a frame, with its associated slots of information, is shown in Figure 2.5. Two types of links were used to connect frames together; links between competing hypotheses that could form a list of differential diagnoses and links between complementary hypotheses that could be present together to explain a patient's findings. The complementary links could describe *causality* (CAUSE-OF, CAUSED-BY), *complications* (COMPLICATION-OF, COMPLICATED-BY) or *simple associations* between hypotheses in cases where the phsyiological mechanism of the connection was unclear.

```
relation to findings
            TRIGGERS                <findings>
            FINDINGS                <findings>

logical decision criteria
            IS-SUFFICIENT           <findings>
            MUST-HAVE               <findings>
            MUST-NOT-HAVE           <findings>

complementary relation to other hypotheses
            CAUSED-BY               <hypotheses>
            CAUSE-OF                <hypotheses>
            COMPLICATED-BY          <hypotheses>
            COMPLICATION-OF         <hypotheses>
            ASSOCIATED-WITH         <hypotheses>

competing relation to other hypotheses
            DIFFERENTIAL-DIAGNOSIS
            (<condition 1><hypotheses>....(<condition k><hypotheses>)

numerical likelihood estimator
            SCORE
            ((<condition 1,1><score 1,1>)...(<condition 1,n1><score 1,n1>))
            ...
            ((<condition m,1><score m,1>)...(<condition m,nm><score m,nm>))
```

Figure 2.5 The Structure of a Frame in PIP (From Szolovits & Pauker, 1978)

A method which combined both categorical and uncertain reasoning was used to update the belief in hypothesis frames as findings about the patient were gathered under the guidance of the program (Szolovits & Pauker, 1978). Each hypothesis frame could be in an *active*, *semi-active* or *inactive state* depending upon the way in which input findings matched the prototypical findings held in the frames. When an input finding matched a FINDING of an active frame, the belief in that frame was updated using the method described below. When an input finding matched a TRIGGER for a frame then that frame was put into its active state (if it wasn't already) and its belief was updated. At the same time any complementary frames were semi-activated and the condition clauses associated with competing frames were evaluated; frames whose conditions evaluated to *true* were also semi-activated. Once a frame had been made semi-active, all its prototypical findings acted as if they were triggers.

Uncertain reasoning was accomplished by a scoring mechanism which involved calculating a likelihood estimate for each active frame; when the likelihood estimate exceeded an upper limit, the frame was confirmed and if it fell below a lower limit the frame was returned to a

semi-active state. The likelihood estimate was the average of a *binding score* and a *matching score*. The binding score of a frame measured the extent to which a hypothesis accounted for the observed findings - it was the ratio of the number of observed findings appearing in the frame or any of its complementary frames to the total number of observed findings. The matching score of a frame measured the extent to which the observed findings fitted the hypothesis. The local matching score was calculated by evaluating the scoring clauses for the frame and dividing by the maximum possible score (its value thus ranged from one to arbitrarily large negative numbers). The total matching score for a frame was the sum of the local matching score for that frame and all its complementary frames.

As well as the scoring mechanism just descibed, a degree of categorical reasoning was employed. No hypothesis frame could be confirmed on the basis of its score, unless the conditions specified in its MUST-HAVE and MUST-NOT-HAVE slots held. Also a frame could be confirmed, without recourse to the scoring mechanism, if any of the findings in its IS-SUFFICIENT slot were observed.

Questioning of the user was controlled by asking for information about unknown findings in the highest scoring hypothesis frame. Once a finding had been input, frames were confirmed, activated, semi-activated or de-activated as necessary, belief in active frames was updated and the next question was selected. This strategy could lead to rather eratic behaviour when two or more hypotheses were scoring at about the same level - the focus of questioning would shift rapidly from one hypothesis to another.

The developers of PIP took great care to point out the analogies between their program and the thought processes of clinicians. Hypothesis frames were equated with a human's long-term memory, findings and active frames with short-term memory. The semi-activated frames were said to model hypotheses that were 'in the back of the physician's mind' (Szolovits & Pauker, 1978). The way in which the system considered a small number of active hypotheses, persued the most promising with an appropriate questioning strategy and then updated the active list, was was also said to reflect the behaviour of an expert clinician. What was not explored, however, was the question of whether this was the *best* way to reach a diagnosis or was the method employed by human problem solvers because they were unable to consider more than a handful of hypotheses simultaneously. Should the developers of computer systems try to model the cognitive processes of humans, or should they try to exploit to the full the computer's ability to store and manipulate large quantities of data?

2.2.5 INTERNIST-1

Perhaps the most ambitious of the early medical knowledge-based systems was developed at the University of Pittsburgh and originally called DIALOG (Pople *et al*, 1975), later to be known as INTERNIST-1. The aim of the project was to diagnose diseases across the whole spectrum of internal medicine and to achieve this, a massive knowledge base was constructed under the guidance of an eminent internist, Jack D. Myers. When the first documented evaluation of the system was performed, the knowledge base comprised some 500 disease profiles, with 3550 different manifestations, and was estimated to cover 70-75% of internal medicine (Miller *et al*, 1982).

Each disease profile listed the possible manifestations of the disease, which could be patient history, signs, symptoms or laboratory test results. Associated with each listed manifestation (a typical disease profile had 85 of them) were two positive, integer parameters; the *evoking strength* and the *frequency* of the manifestation. The evoking strength was a measure of the degree to which the disease should be considered as an explanation for the manifestation. It ranged from 0, for a manifestation so common that it could not be considered specific to that disease, to 5 for pathogenic manifestations. The frequency of a manifestation corresponded to the probability of the manifestation given the disease expressed on a scale of 1, for rarely occurring manifestations, to 5 for those occurring in every case.

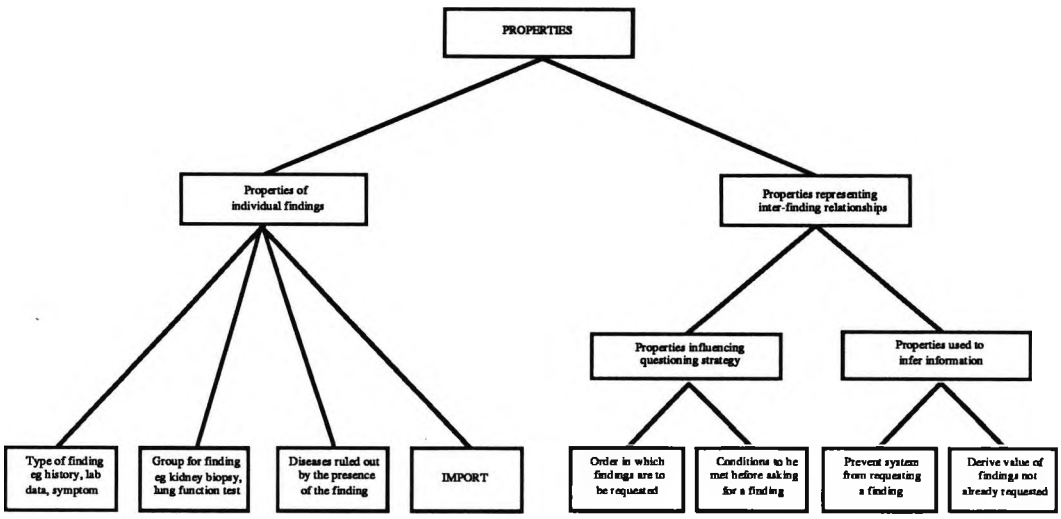


Figure 2.6 Property Types in the INTERNIST-1 Knowledge Base. (Adapted from Masarie *et al*, 1985).

As well as disease profiles, the knowledge base contained details about the *properties* of the manifestations (Masarie *et al*, 1985). The properties represented information such as the relationships between findings, the clinical significance of a finding and the cost of obtaining it. Much of the common sense displayed by INTERNIST-1 in its later versions was accredited to this part of the knowledge base.

By 1985, fourteen basic properties had been defined, which were of two general types - those which were properties of a single finding and those which represented a relationship between two or more findings. Further classification of the properties took place along the lines shown in Figure 2.6. The only property directly used for creating and ranking diagnostic hypotheses was the *import* property. It was an integer in the range 1 to 5, which expressed the extent to which a finding had to be explained by a patient's diagnosis; a finding with an import of 5 had to be explained by the diagnosis, an import of 1 indicated that the finding often occurred in healthy patients and need not necessarily be explained.

Before beginning a consultation, the disease profiles in the knowledge base were reorganized so that each finding was stored with a list of all diseases that could explain it. At the start of a consultation, the user input an initial set of findings from which an initial list of hypotheses was formed. The members of this list were diseases that could individually, or in combination, explain all the input findings. The hypotheses were ranked according their scores, which were calculated in the manner described below; hypotheses which scored less than a certain threshold were temporarily inactivated but could be reconsidered when further evidence became available.

INTERNIST-1 ranked hypotheses by assigning positive and negative points in an *ad hoc* fashion. Each observed finding that appeared in the profile of a disease contributed a positive score, whose magnitude depended upon the evoking strength of the finding in the profile. Bonus points were awarded to hypotheses that were linked to diseases already confirmed by the system. Negative scores for findings which were observed to be absent were assigned in the same way. Additional negative points were awarded for each observed finding left unexplained by a hypothesis; the number of points depending upon the import property of the finding.

Once the hypotheses had been scored, the system identified sets of competing hypotheses, called *problem areas*, which explained the same set of findings. A diagnosis was concluded if it had no competitors or if the difference between its score and that of its nearest competitor was above a certain threshold. In each problem area for which no diagnostic conclusion had been reached, one of three questioning strategies was persued. If the leading hypothesis was scoring far higher than its nearest competitor then the system tried

to confirm this hypothesis by asking about findings which had a high evoking strength in its disease profile. If there were five or more closely competing hypotheses then findings with high frequency scores in their profiles were asked, in the hope that some hypotheses would be eliminated if the finding proved to be absent. When there was a lesser number of competing hypotheses the system tried to maximize the spread in their scores.

The major shortcomings of INTERNIST-1, as identified by its developers (Miller, 1984), were its failure to take account of the severity of findings and diseases (they were either present or absent), its lack of anatomical or temporal reasoning and its insufficient knowledge of causality. These problems were addressed by the specification of the CADUCEUS system (Pople, 1977), whose implementation was expected to continue into the 1990s (Miller, 1984).

In the meantime, INTERNIST-1's vast knowledge base has continued to expand and has been adapted to operate as an electronic textbook of medicine, both on mainframe and micro computers (First *et al*, 1985; Masarie & Miller, 1987).

2.3 The Second Generation

2.3.1 Introduction

A second generation of knowledge-based systems emerged from two sources. As the four major projects described above drew to a conclusion, their developers started work on new systems, building on their earlier experience. Although evaluations of the early systems revealed that they approached the accuracy of expert clinicians in their diagnostic conclusions (see Section 6.2), none were accepted into routine clinical use. Partly this can be attributed to their poor user interfaces and the fact that they had been intended originally as research projects, not practical clinical tools. One general criticism could also explain, to a degree, their non-acceptance by clinicians; they failed to structure their task of diagnosis in the same way as their human counterparts (Feinstein, 1977). The successors to INTERNIST-1 and MYCIN, CADUCEUS (Pople, 1977; 1982) and NEOMYCIN (Clancey & Letsinger, 1981) both incorporated the concept of taxonomies of disease hypotheses as a fundamental organization of medical knowledge. These taxonomies could be used to model the way in which a clinician can form a general hypothesis based on a few observations, refining it to more specific hypotheses as more data are gathered.

Problems with MYCIN's reasoning strategy emerged when it was used as the basis for the teaching system GUIDON (Clancey, 1979). The backward chaining rules did not result in the same patterns of reasoning as human clinicians and couldn't be used to teach diagnostic skills to students. MYCIN's rules were reconfigured, separating out the factual knowledge

from the strategic knowledge. Not only were disease hypotheses organized hierarchically, but so was the strategic knowledge. In the resulting hierarchy of tasks, each strategic procedure was represented as a set of metarules; it was these rules that controlled the problem solving strategy and the way in which data were requested of the user. An interesting description of the evolution from MYCIN to NEOMYCIN and the impact of increasingly powerful technology is given by Clancey (1986).

Out of the first generation of knowledge-based systems came the first domain-independent knowledge-based system *shells*, EMYCIN (van Melle, 1979) and EXPERT which was developed by the designers of CASNET to be *a relatively simple language and notation to represent expert knowledge* (Weiss & Kulikowski, 1979). Like CADUCEUS and NEOMYCIN, EXPERT allowed for the taxonomical classification of diseases and also allowed the same classification for therapies. It provided different mechanisms to question the user for findings which could be used to infer intermediate and causal hypotheses related to the disease hierarchies. Reasoning was accomplished by decision rules providing for finding-finding, finding-hypothesis and hypothesis-hypothesis relations. Weiss & Kulikowski expressed the desire to place *an emphasis on categorical reasoning instead of suboptimal scoring functions*, but included just such a function nevertheless. EXPERT has been used to develop a number of systems including AI/RHEUM (Lindberg *et al*, 1980) for diagnosis in rheumatology and ANEMIA (Quaglini *et al*, 1986) for diagnosis and management of anaemic patients.

It was during the development of the second generation systems that the distinction between different knowledge levels was formalized. In particular, there was recognition of the distinction between heuristic knowledge, which captures the skill of problem solving, and causal knowledge of the domain (Michie, 1981) as evidenced by the separation of these types of knowledge in NEOMYCIN. Similarly, the ABEL project (Patil, 1981) was an attempt to model the deeper, physiological level of knowledge that had been absent in the first generation systems.

As well as the second generation of projects developed by the original workers at Stanford, MIT, Rutgers and Pittsburgh, new projects emerged from research groups attracted by the promise shown in the application of artificial intelligence techniques to medical problem solving. Many of these groups had been working for some time in the field of computer-aided diagnosis and were able to offer interesting new perspectives on the knowledge-based system approach. Three such perspectives are described below along with the multiple-level causal model of ABEL.

2.3.2 ABEL

At MIT, one of the major drawbacks of the first generation systems was seen to be their inability to deal adequately with co-occurring diseases and the way in which their manifestations affected each other. In an attempt to remedy these shortcomings, the ABEL system was developed (Patil, 1981; Patil *et al*, 1981), with causal knowledge about diseases, observations and pathophysiology represented at multiple levels of abstraction.

Each level of description consisted of a semantic network of *nodes* and (predominantly) *causal* links. Normal or abnormal states of physiological parameters were represented as nodes, characterized by a set of *attributes* (*eg* severity, value, temporal characteristics, etc). The causal links were modelled as multi-variable relationships which mapped the attributes of one node to the attributes of another, under particular prevailing conditions. A node at one level of description was described as a *composite node* if it could be represented by a network of nodes (called the *elaboration structure*) at the next, more detailed level of description. One of the nodes in the elaboration structure was designated the *focus node* and was linked to the composite node by a focus link. Two other types of link specified an undefined association between nodes or a grouping of nodes that were of semantic interest (*eg* symptoms suggesting a common disorder) but which were not causally related.

Given a set of observations about a patient, ABEL constructed a set of *patient specific models* which were portions of the causal network, spanning all possible levels, instantiated by the data. There would usually be a number of patient specific models competing to explain the data and for each one ABEL generated a set of *diagnostic closures*. A typical patient specific model would contain an incomplete causal pathway; the set of diagnostic closures for the model was formed by finding the possible extensions that could be traced. The diagnostic closures provided predictions of the patient observations; if a reported finding contradicted all the predictions then a new set of patient specific models had to be generated.

The user input a set of patient data to start the consultation; thereafter data were gathered by questioning of the user, the order of questions being decided by constructing a hierarchy of the goals to be satisfied in order to discriminate between the competing patient specific models and diagnostic closures (Patil *et al*, 1982). The initial set of patient specific models was generated by identifying possible areas on an acid-base nomogram (see Appendix II) based only on blood gas and electrolyte data. Each area of the nomogram corresponded to a node at the highest level in ABEL's knowledge structure. The nodes instantiated in this way were expanded so that the lower, physiological level nodes which they subsumed also became instantiated.

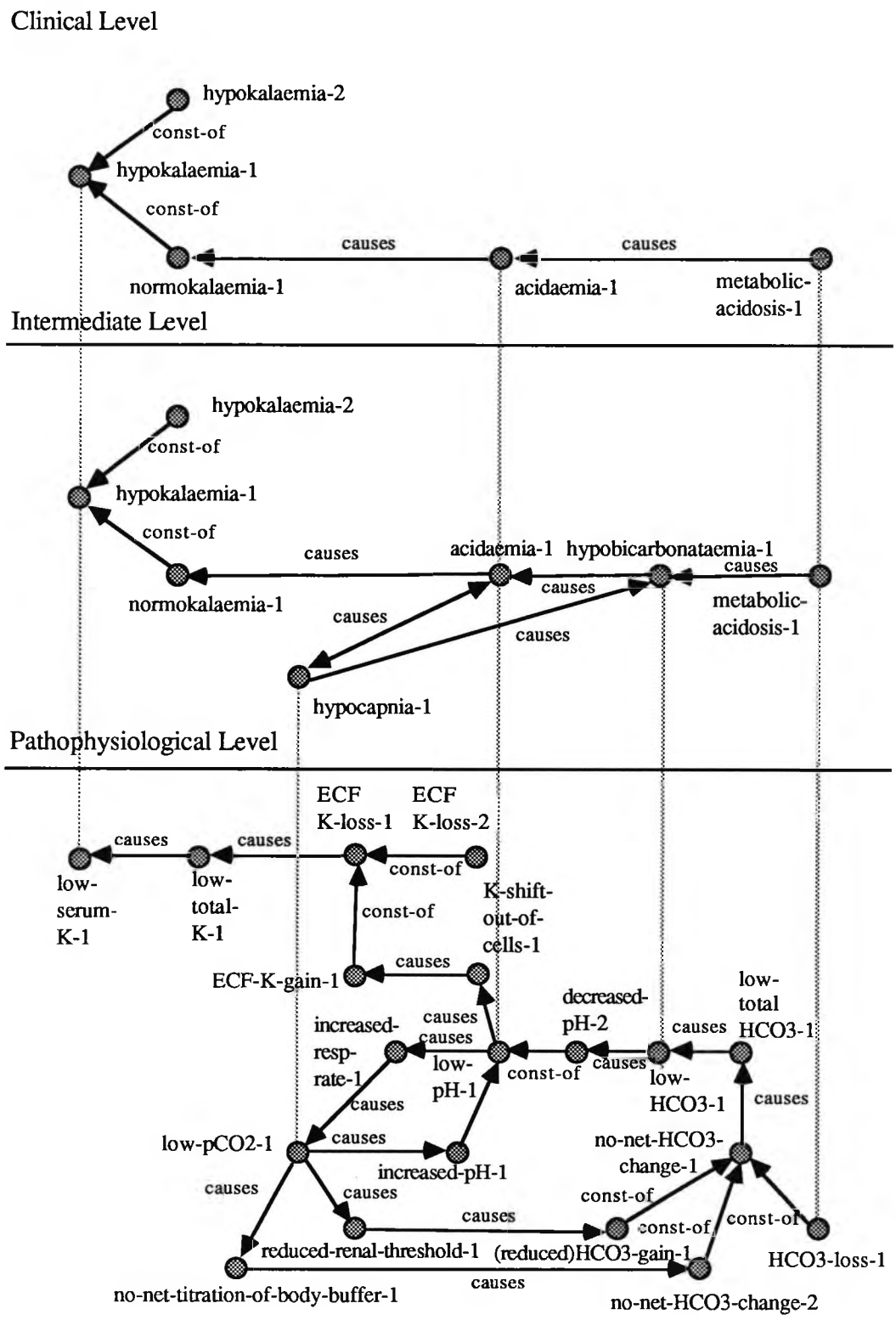


Figure 2.7 The Three-Level Disease Description in ABEL (Adapted from Patil, 1981).

At the physiological level of representation the process of component summation could take place; quantititive causal relations could be combined so that two or more causes could contribute to the instantiation of a node. Networks instantiated at the lower levels could then be aggregated to instantiate higher level nodes through the focus links between the representation levels. This provided an important mechanism, whereby the interaction between the manifestations of co-occurring disorders could be determined at the physiological level and then propagated to the level of observations.

2.3.3 ATTENDING: A Critiquing Approach

The ATTENDING system (Miller, 1983) proposed a new role for knowledge-based systems in medicine. It's task was to plan the anaesthetic management of patients undergoing surgery, but in addition to the planned surgical procedure and the patient's relevant medical problems, the user was asked to input his own anaesthetic management plan. This plan was then *critiqued* by the system and any other feasible alternatives were suggested.

The generation of an anaesthetic management plan involves a series of decisions in which the benefits and risks for the patient must be weighed. The first decision is whether to give general or local anaesthesia. For general anaesthesia, decisions must be made about how to induce and maintain anaesthesia and how to intubate the patient. Each decision involves the selection of different drugs or techniques in order to achieve a goal; this knowledge was represented in ATTENDING by adapting the augmented transition network formalism from linguistic analysis (where it is used as a method of top-down parsing for natural language understanding). The resulting *Augmented Decision Network* (ADN) consisted of a series of nodes connected by arcs, where each arc represented the outcome of a decision. Part of the ADN used by ATTENDING is shown in Figure 2.8. In order to generate an anaesthetic management plan, the topmost network in Figure 2.8 was traversed from a start node (one with no incoming arcs) to a terminal node (one with no outgoing arcs). Each arc traversed represented a decision made about the plan; when two or more arcs connected two nodes, there were a number of alternative ways in which the same goal could be achieved. Some arcs represented sub-networks in the ADN. In order to traverse such an arc, the sub-network had to be traversed successfully from its start node to a terminal node. In this way, the hierarchical nature of the decision making process in anaesthesia management was captured, although the full potential of the ADN representation was not exploited because the networks were restricted to linear progressions of nodes, with one start and one terminal node.

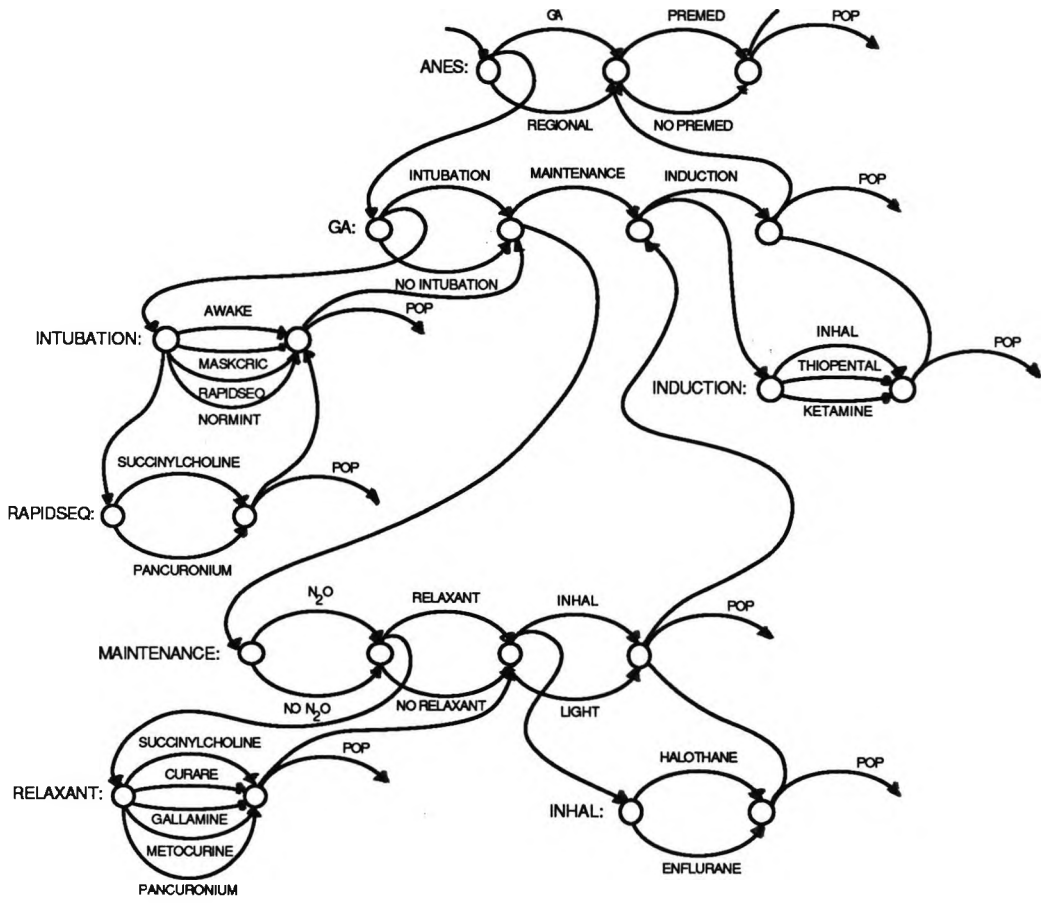


Figure 2.8 An Augmented Decision Network in ATTENDING (Adapted from Miller, 1983).

The ADN can be described as *augmented* because each arc had routines attached which activated or deactivated it in the context of a particular case (*eg* the arc for spinal anaesthesia was deactivated for a case of eye surgery). Knowledge about how the clinical condition of the patient affects the anaesthetic techniques used, was held in frames which consisted of *management principles* and *risk factors* associated with the condition. These were linked to arcs in the ADN, so that each arc was augmented by a list of risks and benefits (defined as LOW, MODERATE, HIGH or EXTREME) which were used in deciding whether to traverse that arc or not. The risks (benefits were defined as negative risks) were combined heuristically when several alternative actions were under consideration. First, the alternatives were broadly considered by assigning to each an overall risk equal to the greatest risk associated with its constituent elements. The alternative with the lowest risk was then chosen; if two or more alternatives shared the same, lowest risk a set of domain specific rules was used to discriminate between them.

When an anaesthetist input his management plan, ATTENDING converted it into a *patient action tree,* which was a hierarchical structure representing the major management actions. These actions were then used to guide the system through a traversal of its ADN. Where alternative arcs existed one of them would, necessarily, correspond to an action in the patient action tree. Any alternative whose risk was less than or equal to the risk of the anaesthetist's own input action was included as a feasible alternative in the final output critique.

The generation of the prose form of ATTENDING's critique was achieved by a second adaptation of the augmented transition network formalism. Traditionally, ATNs have been used to parse sentences as part of the process of language understanding; ATTENDING stored fragments of prose along the arcs of the ATN so that sentences were generated as the arcs were traversed.

The critiquing approach has been generalized in a shell program called E-ATTENDING (Miller, 1986) which consists of a production rule interpreter, a system of *expressive frames* that can be manipulated to transfer data input by the user or inferred by the production rules into critiquing comments, and a prose generation module. Although the ATTENDING systems introduced some powerful ideas about the role to be played by knowledge-based systems in medicine, the novel knowledge manipulation techniques proposed (ADNs, prose generation, expressive frames) have not been exploited to their full potential. How well these techniques would scale up to function at their full theoretical power remains unclear.

2.3.4 CADIAG-2

The system CADIAG-2 (Adlassnig, 1980; Adlassnig *et al*, 1985) uses *fuzzy set theory* (Zadeh, 1965) and *compositional rules of inference* (Zadeh, 1973) to produce sets of confirmed, excluded and hypothesized diseases. The knowledge base contains factors expressing the frequency of occurrence of symptom S with disease D (actually P(S|D)) and the strength of confirmation of S for D (P(D|S)). Similar relationships are specified for combinations of symptoms with diseases, symptoms with other symptoms and diseases with other diseases. The relationships are represented as rules of the form:

IF <antecedent>
THEN <consequence>
WITH frequency of occurrence **O** and strength of confirmation **C**

Frequency of Occurrence	Interval	Strength of Confirmation
always	[1.00,1.00]	always
almost always	[0.99,0.98]	almost always
very often	[0.97,0.83]	very strong
often	[0.82,0.68]	strong
sometimes	[0.67,0.33]	medium
seldom	[0.32,0.18]	weak
very seldom	[0.17,0.03]	very weak
almost never	[0.02,0.01]	almost never
never	[0.00,0.00]	never

Table 2.1 Translation of Linguistic Terms in CADIAG-2 (From Adlassnig, 1980)

The knowledge base can be built up either by statistical studies on a patient database or by having experts express linguistic estimates of O and C. These linguistic terms are then translated using the scheme in Table 2.1.

At the start of a consultation, data can be input using linguistic terms (*eg* high fever) which are translated in a similar manner to above. Alternatively, numerical data can be transferred from the hospital database and classified by a *fuzzy interpreter* as described in Section 3.6.3. After gathering patient symptoms, the symptom-symptom rules are used to infer unknown data, fuzzy values are assigned for symptom combinations and the diagnostic process begins. Three rules of compositional inference are used - *confirmation composition, positive exclusion composition, negative exclusion composition* - to produce a fuzzy description of a patient's disease based on the fuzzy symptom data. These rules will be described in more detail in Section 3.7.5.

2.3.5 A Set Covering Model of Diagnosis

An interesting perspective on the problem of medical diagnosis was provided by the incorporation of a set covering model into a functional knowledge-based system for the diagnosis of liver disorders (Reggia *et al,* 1983).

Consider a set D={d1,d2,d3...dn} of diseases and let M= {m1,m2,m3...mp} be the set of all manifestations that could be observed in D. Further sets can be defined as follows:

man(di)={ma,mb,...} the set of manifestations of disease di

causes(mj)={da,db,...} the set of diseases that cause mj

di	man(di)
d1	m1 m4
d2	m1 m3 m4
d3	m1 m3
d4	m1 m6
d5	m2 m3 m4
d6	m2 m3
d7	m2 m5
d8	m4 m5 m6
d9	m2 m5

mi	causes(mi)
m1	d1 d2 d3 d4
m2	d5 d6 d7 d9
m3	d2 d3 d5 d6
m4	d1 d2 d5 d8
m5	d7 d8 d9
m6	d4 d8

Table 2.2 Example Sets of Manifestations and Causes (Adapted from Reggia *et al*, 1983)

An example of the sets defined above is shown in Table 2.2. If M+ is the set of manifestations observed in a particular patient, then sets of diseases E can be generated such that E *explains* M+ *ie* each manifestation in M+ is caused by one or more of the members of E. The best explanation of M+ is given by the set E with the least members (the set with the lowest *cardinality*).

The set covering model can be implemented by updating three data structures as manifestations are observed:

M+ the set of manifestations so far observed
SCOPE the set of diseases that cause at least one member of M+
FOCUS the set of sets of lowest cardinality that explain M+

The sequence of updating can be summarized:

1) Get the next manifestation mj
2) Retrieve the causes of mj *ie* the set causes(mj)
3) Generate the new scope which is SCOPE $\cup$ causes(mj)
   ie add any causes of mj that are not already in the SCOPE
4) Form the new FOCUS

Table 2.3 shows how M+, SCOPE and FOCUS are built up using the above algorithm. When the first manifestation m1 is observed, the scope contains the four diseases that cause m1 and the focus is the same four diseases (*ie* four sets of cardinality 1). When the next observation, m4, is made, d5 and d8 are added to the scope and the new focus {d1,d2} is formed by intersecting the causes of m4 with the old focus (*ie* either d1 or d2 could explain both the manifestations m1 and m4). When m5 is observed, d9 is added to the scope, but when the causes of m5 are intersected with the focus, the empty set is obtained. This

indicates that no single disease can explain the three manifestations and that sets of cardinality 2 must be created from the diseases in the scope to form the new focus.

A knowledge-based system which used the set covering model as the basis of its reasoning strategy, held knowledge about liver disorders in frame-like *descriptions*; findings were associated with each disorder using a symbolic probability estimate ranging in five steps from *never* to *always*. A distinction was made between those findings *caused by* a disorder and those which *suggested* its presence but were not caused by it; these were called *setting factors* and included findings such as age and sex.

The disorder descriptions were used to construct the sets causes(mj) and also played a role in the control of the system. The pattern of questioning was determined by finding the most commonly occurring manifestation in the descriptions of the disorders in the current focus - these disorders were considered as being the active hypotheses. The descriptions could also categorically reject a disorder during the process of gathering manifestations. If a manifestation which always occurred with a disorder was observed to be absent, or if one that never occurred with that disorder was present, then the disorder was effectively removed from the set of possible disorders, D.

Once all possible questions had been asked, the focus would typically contain more than one possible explanation for the observed manifestations. At this stage, a scoring scheme was used to rank the alternative hypotheses. The score for each hypothesis was based on the combination of a *setting score* and a *matching score*. The setting score represented the general likelihood of a disorder in a particular context and was the sum of the initial probability of the disorder and the probability estimates associated with its setting factors (symbolic probabilities were converted to integer scores in the range 0 to 4). The matching score was a measure of how well a disorder fitted the manifestations.

The set covering model provided an important theoretical basis for the notion of coverage and parsimony in a medical diagnosis, that was interesting not only in its application as a knowledge-based system but also as a cognitive model in its own right.

Observation made	M+	SCOPE	FOCUS
none	∅	∅	∅
m1	{m1}	{d1 d2 d3 d4}	{d1 d2 d3 d4}
m4	{m1 m4}	{d1 d2 d3 d4 d5 d8}	{d1 d2}
m5	{m1 m4 m5}	{d1 d2 d3 d4 d5 d6 d7 d8 d9}	{d1 d2}x{d7 d8 d9} and {d8 d3} {d8 d4}

Table 2.3 Building M+, SCOPE and FOCUS (Adapted from Reggia *et al*, 1983).

2.4 The Third Generation

2.4.1 Introduction

By the mid 1980s, rapid technological developments had made powerful microcomputers and AI workstations widely available, and knowledge-based system *shells*, similar to EMYCIN and EXPERT, were beginning to emerge as commercial products. As a result, a large number of small, specialized knowledge-based systems began to be developed. These projects have introduced an ever widening audience of clinicians to the concept of knowledge-based consultation systems in medicine, and this may prove to be their main achievement; in terms of their method of implementation they show little advance on the first generation systems and whether they will be accepted for routine use remains to be seen.

At the other end of the scale, powerful technology has led to the development of some extremely large systems. The INTERNIST-1 knowledge base has continued to expand and has been transported to microcomputers, on which it operates as an electronic textbook of medicine (Masarie & Miller, 1987). The DXplain project has a knowledge base covering about 2000 diseases, 4700 manifestations and 65,000 inter-relationships (Barnett *et al*, 1987). It uses a simple algorithm for generating lists of diagnostic hypotheses - its aim being to suggest all possibilities, not to identify a specific diagnosis. DXplain is supported by the American Medical Association and has been made available at over 40 locations on their nationwide computer network. In England, the Oxford System of Medicine (Fox *et al*, 1987) is aimed at providing support for general practitioners, which requires a massive knowledge base, accessible in a variety of different ways through a user-friendly interface.

The distinction between surface and deep levels of knowledge representation has continued to occupy researchers in artificial intelligence. Whilst some have explored the use of deep level, qualitative, causal models (Forbus, 1984; Kuipers, 1986), or combined qualitative/quantitative models (Kunz, 1984) others have searched for more rigorous methods of reasoning at the surface level (see Section 3.7). Reasoning with deep level knowledge is computationlly expensive; reasoning with heuristic, surface level knowledge is more efficient but at the cost of losing the richness of explanation offered by deeper representations. A number of systems have recently been proposed which combine deep and surface level knowledge. The developers of ABEL have described the redesign of their system along such lines (Patil & Senyk, 1987) and two systems that combine efficient surface level reasoning and deep physiological knowledge are described below.

2.4.2 MUNIN

MUNIN is a knowledge-based system which is designed to diagnose neuro-muscular diseases based on the results of EMG tests and to advise the electromyographer on the best tests to perform (Andreassen *et al*, 1987). Two knowledge bases exist in the system, one holding details of neuroanatomy, the other representing a causal network of diseases, pathophysiological features and EMG findings. A prototype network is based on a *nanohuman* with only one muscle and consists of 4 disease states (including normal), 8 pathophysiological features and 15 EMG findings. The causal links between nodes are quantified by conditional probabilities and the *a priori* probabilities of the diseases are updated using a Bayesian scheme that propagates the effect of evidence through the network (see Section 3.7.3.2). The direction of propagation can also be reversed, allowing the expected frequency of EMG findings to be calculated for a given distribution of disease probabilities. This feature is useful for verifying the subjective conditional probabilities associated with the network links; by assuming a 100% probability for a disease, the expected frequency of various EMG findings can be compared to real case data and the network adjusted to achieve a match.

A graphical interface allows the user to see the probabilities of the various hypotheses change as evidence is propagated and is shown in Figure 2.9. It can be seen from Figure 2.9 that a diagnosis of *Others* is defined. All possible evidence is assigned with equal weighting for this node, thus ensuring that Others will be diagnosed if all other hypotheses have been ruled out by the evidence. Another interesting feature of MUNIN is the method of classification of numerical data using probability distributions - this is discussed in more detail in Section 3.6.4.

Therapeutic decisions are made using a *utility matrix* which specifies a subjective utility for the treatment of each disease when applied to each of the other possible diseases. The benefit of selecting the treatment T_i (corresponding to disease D_i) is given by:

$$BEN(T_i) = \Sigma_j UTIL(T_i|D_j).P(D_j)$$

where $UTIL(T_i|D_j)$ is an element from the utility matrix expressing the utility of treatment T_i with disease D_j and $P(D_j)$ is the probability of D_j based on MUNIN's assessment of the patient observations. The treatment affording maximum benefit is selected. The system's developers have noted the ethical difficulties in determining the elements of the utility matrix since many varied and conflicting considerations must be reduced to a single number.

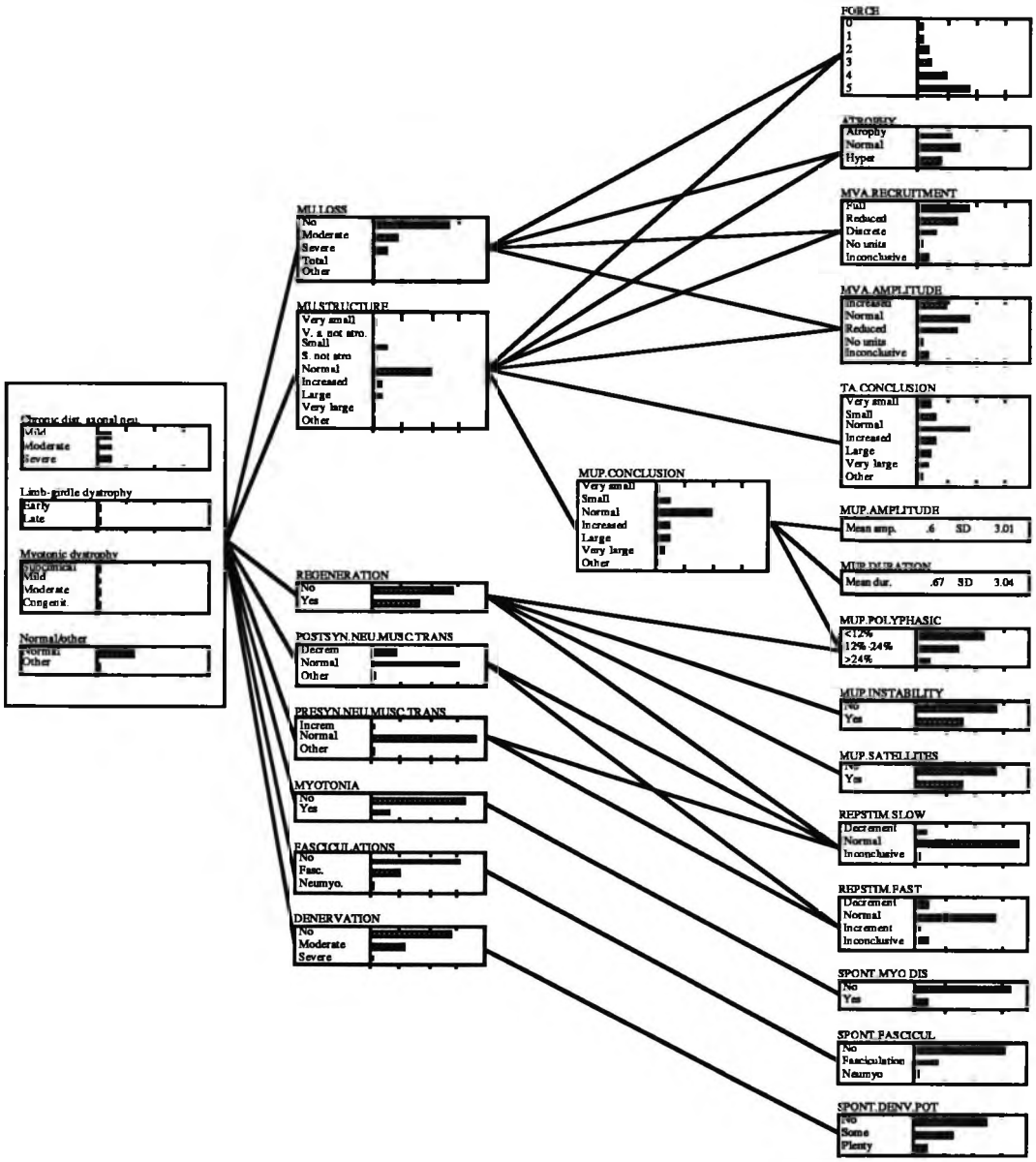


Figure 2.9 MUNIN's Graphical Interface (Adapted from Andraessen *et al*, 1987)

The current implementation of MUNIN uses a very small causal network but already the computational expense of evidence propagation is creating problems (it takes about 20 seconds to propagate evidence through the network). It is hoped that the use of better algorithms, computer languages and hardware will make it possible to expand the prototype into a fully functional system.

2.4.3 CHECK

A knowledge-based system for the diagnosis of liver disease, called CHECK and developed at the University if Turin, combines heuristic and deep causal knowledge within a single system (Molino *et al,* 1986). At the surface, heuristic level, knowledge held in

frames is used to produce ranked lists of hypotheses in a fashion similar to the operation of PIP. The knowledge held as a causal network can be used to confirm hypotheses from the surface level, to provide explanations for a specific case or to give general explanations of the domain.

The knowledge frames at the surface level are organized hierarchically; there are 10 frames representing broad classifications of liver diseases and each of these has a number (1-13) of sub-classifications. Slots in each frame represent *major* and *supplementary* findings, *triggers* for the frame, *associated* and *alternative* hypotheses and *confirmation rules* which can exclude a hypothesis or alter its degree of belief under certain conditions. After some initial data have been entered, the broad classification frames are scored and refinement focusses on the sub-classifications of the highest-ranking hypothesis (and any associated classes specified in its frame). If no hypothesis is triggered and supported by some evidence, default hypotheses are used for focussing further investigation.

The deeper level knowledge is a causal network with four types of node; *hypotheses* correspond to diagnostic hypotheses at the surface level; *initial states* represent pathophysiological conditions that can be considered as the causes of disease - other intermediate states or final states represent pathophysiological conditions in the evolution of a disease; *findings* are manifestations from the surface level; *actions* represent types of cause/effect relationships. It follows that states are only connected together through action nodes; findings are connected to states; hypotheses are connected to findings. The connecting arcs can also be of several types, defining:causal relationships, links between states and manifestations, hypotheses that can be considered as sets of states, subclassifications, or network loops. Arcs can be labelled as MUST (be traversed) or MAY (be traversed) and can have attached conditions to be evaluated before traversal. Arcs can be connected together at entry or exit to nodes with logical constraints (AND,OR) or quantifications (*eg* at least one of, at most one of,...). Part of CHECK's network for alcoholic cirrhosis is shown in Figure 2.10.

Reasoning in the causal network is initiated by instantiating hypothesis nodes from the conclusions reached at the surface level. The states representing underlying physiological causes of the hypotheses are then identified and paths are traced from these causes to the hypotheses, with arc conditions and action nodes being evaluated so that a portion of the network becomes instantiated. If no suitable instantiation of the network proves possible, the initial hypotheses from the surface level may be rejected. The network can also be used to search forward from a hypothesis to identify its consequences and manifestations.

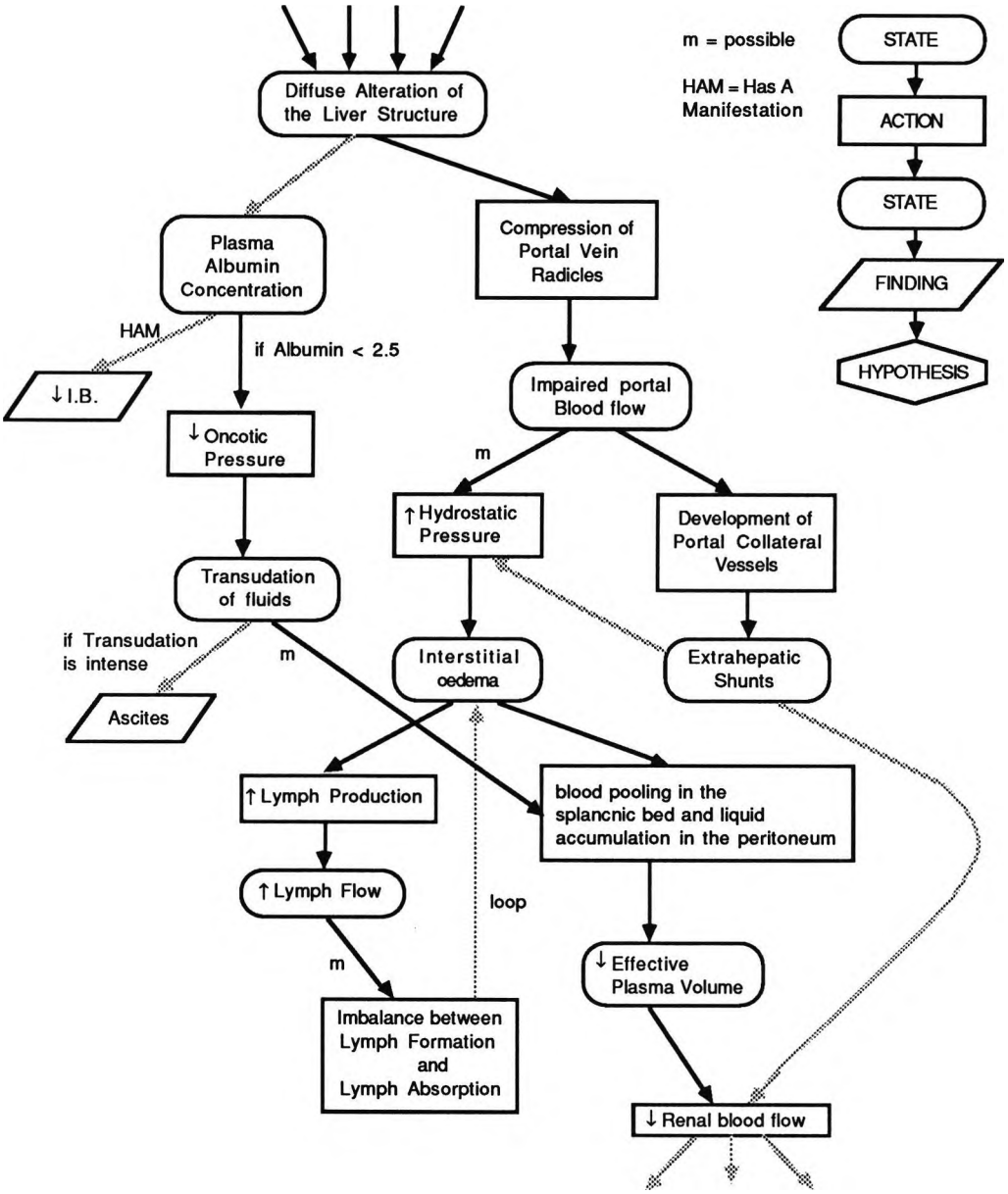


Figure 2.10 A Causal Network in CHECK (Adapted from Molino *et al,* 1986).

The network is implemented through object-oriented procedures (see Section 3.3) programmed in PROLOG. Nodes are represented as objects with the inherited properties of their class (classes being hypotheses, states, findings and actions); the arcs are implemented by message passing between the objects. The surface level of CHECK is based on an earlier system, LITO2 (Cravetto *et al,* 1985), and comprises 400k of PROLOG code. The first version of the causal network in CHECK covered only one disease class (cirrhosis) but comprised over 100 nodes and another 400k of PROLOG - an indication of the enormous effort and computing resources needed to represent and reason with deep level knowledge.

2.5 Summary

This chapter has outlined the development of the field of knowledge-based systems, applied to the problem of medical diagnosis, since its inception in the early 1970s. A description has been given of four early systems (MYCIN, CASNET, PIP and INTERNIST-1) which encompassed the three main methods of knowledge representation - rules, frames and semantic/causal nets. The major shortcomings of these systems have been identified as poor user interfaces, the failure to represent knowledge at a physiological level, poor modelling of human clinicians in terms of the structure of the diagnostic process and a reliance upon *ad hoc* scoring mechanisms for the handling of uncertainty.

A second generation of systems has been described which explored the use of deeper knowledge representations and introduced the concepts of the set covering model and the computer-generated critique. Finally, the emergence of a new generation of knowledge-based systems has been identified. These systems exploit the increasing power of technology to provide improved user interfaces and employ more rigorous methods of reasoning in the presence of uncertainty. Generally speaking, deep level causal knowledge has been found to be unnecessary and inefficient for routine diagnosis and is now being incorporated in systems to augment an initial diagnosis based on more efficient surface level reasoning.

Some of the themes of knowledge representation, manipulation and acquisition that have been introduced above will be developed further in the next chapter.

CHAPTER THREE

METHODS OF KNOWLEDGE REPRESENTATION AND CONTROL

3.1 Introduction

The last chapter introduced some of the issues of knowledge representation and control that have been explored in the context of knowledge-based systems in medicine. This chapter presents in more detail a selection of those issues which are relevant to the knowledge-based system described in Chapters 4 and 5.

There are many different types of knowledge that could be represented in a computer system; the best representation for a body of knowledge depends on its type and the way in which it is to be manipulated. One way in which knowledge can be categorized is to describe it as *declarative* or *procedural.* Declarative knowledge is descriptive, factual knowledge; procedural knowledge is the knowledge of action. Knowledge can also be represented at different levels (Michie, 1981): *surface-level* or *heuristic* knowledge expresses associations without regard for the underlying cause; *deeper-level, causal* knowledge explores the reasons for these associations at different levels of complexity.

Three main methods of knowledge representation have been discussed in connection with knowledge-based systems in medicine: rule-based, frame-based and semantic network representations. These are considered in more detail in Sections 3.2, 3.3 and 3.4, which discuss the types of knowledge accommodated by each representation and the methods of reasoning that can be employed.

Section 3.5 describes the blackboard architecture which can be used for the overall control of a knowledge-based system. The next two sections are concerned with the manipulation of data and knowledge; Section 3.6 discusses the problem of data classification, which is of special interest in medical systems; several methods of handling uncertainty in knowledge-based systems are presented in Section 3.7 and their relative merits are discussed.

The interface between a knowledge-based system and its users is an important factor in its acceptability. Two aspects of the use of a natural language interface are presented in Section 3.8 - the understanding of user queries and the generation of textual output.

Finally, Section 3.9 addresses the problem of how knowledge can be acquired by interaction between domain experts and knowledge engineers.

3.2 Rule-Based Representation

3.2.1 Background

The origin of rule-based systems is to be found in the production system method of inference proposed by Post (1943). Discrete chunks of knowledge are encoded as conditional statements of the type:

If <condition 1>, <condition 2>, . . <condition n>	Left Hand Side (LHS) Premise
	---
Then execute <action>	Right Hand Side (RHS) Action

and the system operates by successively executing such rules to produce a chain of reasoning.

It has been suggested (Newell & Simon, 1972) that this representation of knowledge reflects the way in which humans organize their own knowledge for problem solving. Production rules appear a direct and natural way to encode the heuristic rules often expressed by expert human problem solvers when asked to explain their reasoning process; the realization that these rules could be captured and used by a computer has led to renewed interest in the study of heuristics in their own right (Lenat, 1982).

Rule-based systems operate best in domains characterized by a large number of distinct states, where knowledge about how to move from state to state can be separated from knowledge about the states themselves. They are not well suited to domains involving well defined governing principles, complex control processes or strong prescriptions for the use of specific items of knowledge (Davis & King, 1977).

3.2.2. Structure

There are three basic elements to a rule-based system; a *dynamic database* of facts about the current problem, a *knowledge base* of rules and an *inference engine* which matches rules with the database and executes their actions so as to infer new data. An elaboration of this basic structure is shown in Figure 3.1.

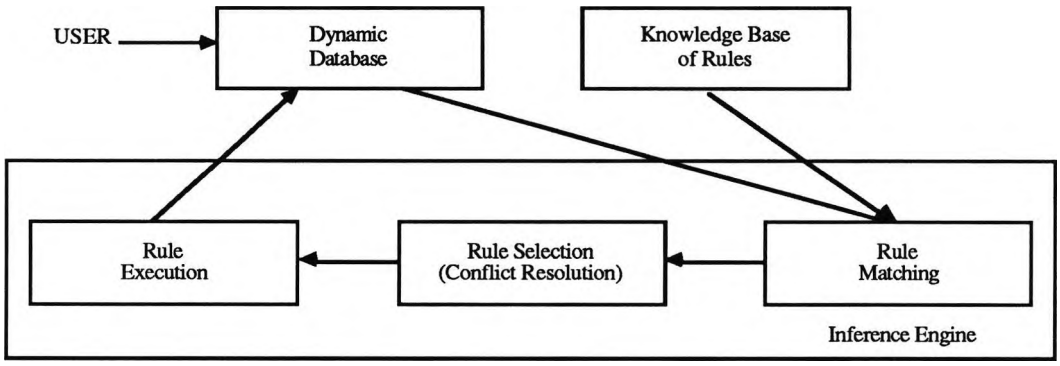


Figure 3.1 A Rule-Based System. (Arrows show the flow of information)

The inference engine can be viewed as a three stage process of matching rules in the knowledge base with data in the database (the match can be with the LHS or RHS of the rules, depending on the control strategy used), selecting one of the matched rules (conflict resolution) and then modifying the database by execution of the action in the RHS of the selected rule. Discussion of these three elements is left until Section 3.2.4, but it is worth noting here that the inference mechanism can be made completely independent of the domain represented in the knowledge base.

3.2.3 Representation

When used in a practical system implementation, it is useful to represent rules as more than simple premise-action pairs. At the lowest level, rules are represented in a machine-executable format - perhaps LISP, PROLOG or a specially designed rule-based language with its own interpreter and compiler. When the chaining of rules is traced to generate explanations of the system's conclusions, it can be advantageous to present rules to the user in an English language format; it may also be useful to specify rules in an intermediate, restricted vocabulary format to aid in editing and debugging.

Rules can be organized into hierarchical sets, each set being used in a particular problem-solving context to perform a specific function (MYCIN's rules were organized according to the CONTEXT tree - see Section 2.2.2). The speed of system execution can be improved by storing lists of variables used in the premise of a rule and concluded by its action. The properties described above can be organized into an information structure for each rule, such as the one shown in Figure 3.2.

Rule set: Lab data **Rule no:** 045

Premise: current_data(variable,pH,V), V>7.45.

Action: assert(current_data(disorder,alkalaemia)).

Edited by: John Chelsom **Date:** 25/6/89

English Translation: If arterial blood pH is greater than 7.45 then the patient has alkalaemia

Intermediate Translation: If pH>7.45 then conclude alkalaemia

Premise Dependents: pH

Concludes Data: disorder

Figure 3.2 The Structured Representation of a Rule.

3.2.4 Control

There are two basic strategies that can be used for the control of a rule-based system; *forward-chaining* or *backward-chaining* (both strategies can be used at different times in the same system). Backward-chaining, or goal-directed, systems seek first to satisfy the LHS of a top-goal rule, so that its RHS can be evaluated to achieve the overall goal of the system. In order to satisfy the LHS conditions, information that is not available in the database must be inferred by the execution of further rules; the system therefore searches for rules whose RHS can be used to infer the information required. Invocation of these rules will require additional data, leading to the invocation of further rules and so on. The MYCIN system operates in this fashion and has been described in Section 2.2.2.

The forward-chaining, or data-driven, strategy involves scanning the LHS of all rules to find those whose conditions are satisfied by the current state of the database. One of the matching rules is then chosen for execution, the database is modified by the RHS of the chosen rule and the process of scanning recommences.

Since a piece of data is normally only inferred by a few rules, the process of matching the RHSs of rules in a backward-chaining system is quite straightforward and this can make such systems very efficient. The matching of LHSs is potentially very inefficient, since it may involve the evaluation of many conditions; forward-chaining systems can spend a large proportion of their execution time performing LHS matches. For a system of any

great size (more than a hundred rules, say) a full scan of each LHS would become extremely expensive, computationally, and alternative means of triggering rules must be found. One solution to this problem is to execute the first rule for which a LHS match is found in the database; this also eliminates the need for the step of conflict resolution on a set of matched rules.

Another way to improve the efficiency of rule matching (in either forward- or backward-chaining systems) is the use of indexing. Each rule has an associated set of descriptors which are matched against features in the database to produce a list of rules whose LHSs might be matched successfully. The exhaustive evaluation of LHS conditions is then performed on just this subset of the total rule set.

Efficiency can be further improved by the use of meta-rules (Davis, 1980). Meta-rules are rules which reason about other rules in the knowledge base, either to control the process of LHS pattern matching or to assist in the selection of one matched rule for execution. An example of a meta-rule that could be used in this selection procedure might be:

> **IF** the age of the patient is greater than 60
> **AND** there are rules that mention high risk
> **AND** there are rules that mention low risk
> **THEN** it is very likely (0.8) that the former should be used after the latter.

(adapted from Davis, 1980).

Meta-rules are used in the following way. After indexing has produced the set of potentially useful rules, S, any meta-rules are evaluated and executed so that S may be reduced in size before pattern matching takes place. Meta-level rules thus represent a higher level of rules in the system. In theory, there could exist meta-meta-rules, executed before and reasoning about the meta-rules; indeed the use of meta-rules could be extended to an arbitrary number of levels.

The generation of a set of matched rules can also be made faster by improving the efficiency of the pattern matching algorithm itself. The most rudimentary pattern matcher would examine every element in the database on each control cycle of the system and match these with every LHS in the rule set. Usually, only a few database elements will be changed on each cycle, and so many unchanged expressions will be re-evaluated unnecessarily. By introducing a memory of all matched LHS conditions and updating it as changes are made to the database, unnecessary re-evaluations can be eliminated. One pattern matching algorithm that works in this manner is the Rete Match Algorithm (Forgy, 1982).

Figure 3.3 The Rete Match Algorithm (Adapted from Forgy, 1982)

The Rete algorithm can be viewed as a black box which ouputs changes to the conflict set (*ie* the set of rules with LHS matches in the database) on receipt of an input of changes made to the database (Forgy, 1982). This concept is illustrated in Figure 3.3. Changes to the database are recorded as tokens consisting of a tag indicating the type of change (*eg* '+' indicates addition to the database, '-' indicates retraction) and a set of arguments corresponding to database elements.

The process of matching a LHS expression can be thought of as a series of tests that must be performed on that expression. The Rete algorithm stores each LHS expression as a network of nodes in which each node represents one such test. The root node in the network is the input to the central box in Figure 3.3, the terminal node is its output. The algorithm works by passing tokens to the root nodes of networks, performing the tests at each node and propagating the token down the network if the test is successful. A LHS match is indicated when a token reaches the terminal node of a network.

There are two basic types of test that can be performed on a LHS expression. Intra-element tests involve only one database element (*eg* pH>7.45) and are represented by nodes with one input and one or more outputs; inter-element tests involve more than one database element (*eg* $\Delta pH \leq 0.8\Delta pCO_2$) and are represented by nodes with two inputs and one output. In the most basic Rete network, information is stored on each system cycle by saving the inputs to every two input node. A simple Rete network is shown in Figure 3.4.

The TREAT algorithm (Miranker, 1987) improves on the performance of Rete by explicitly storing the conflict set on each cycle so that re-evaluation of conditions is limited to those that involve altered database elements. Both TREAT and Rete rely on careful organization of memory storage locations and representation at machine level to achieve efficiency, but even without using these, pattern matching in rule-based systems can be made more efficient by adopting some of the concepts of the algorithms.

Once the conflict set has been generated, several strategies have been suggested (Davis & King, 1977) for selecting one rule for execution (conflict resolution). Apart from the use of meta-rules, mentioned above, these strategies could be:

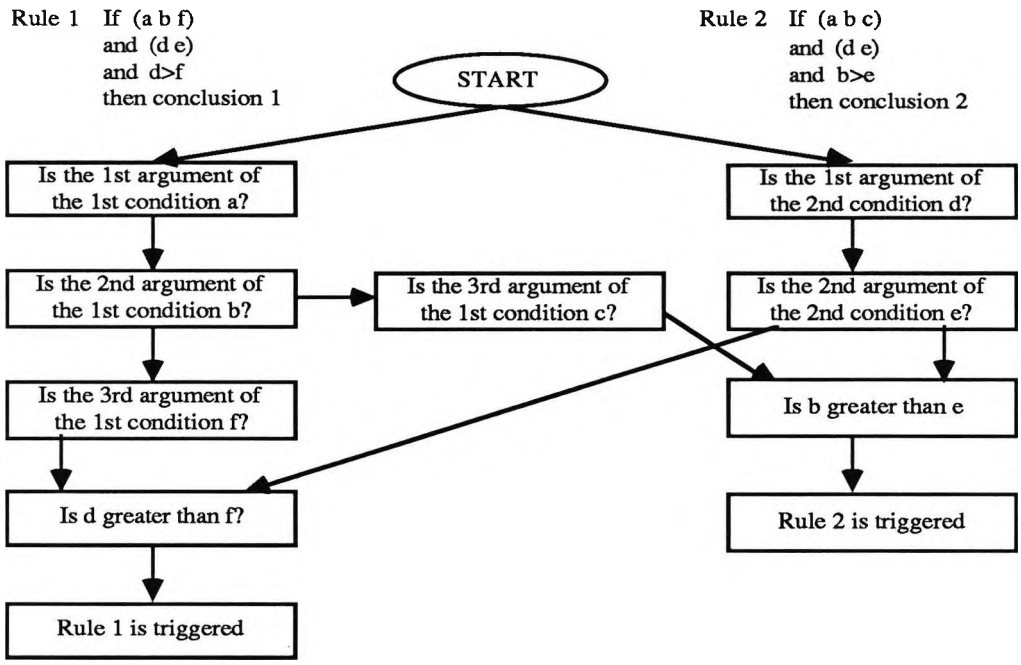


Figure 3.4 A Rete Network for Checking Premise Conditions

arrange all the rules in the system according to their priority for execution: execute the highest priority rule in the conflict set

arrange elements in the database according to their priority for discovery: execute the rule that infers the highest priority data

execute the most specific rules first

execute the most recently triggered rule in the conflict set

In fact it could be argued that all these strategies can be implemented as meta-rules of various degrees of generality.

3.2.5 Summary

This section has described how discrete chunks of problem-solving knowledge can be captured in the form of production rules which make alterations to a dynamic database if certain conditions are satisfied. It has been shown that these rules can be chained together to reason towards problem solutions and that the efficiency of this process depends critically on the control mechanisms used. Representation of rules in a practical system implementation can be achieved by storing the facets associated with each rule in knowledge structures that closely resemble the frame-based representation of knowledge described in the next section.

3.3 Frame-Based Representation

3.3.1 Background

The general notion of organizing knowledge into structured descriptions called *frames*, was first explicitly formulated in the field of computer vision (Minsky, 1975). The theory of frames represented a middle ground between the proponents of declarative and procedural knowledge representations, and just as these representations formed the basis for two classes of programming languages, so frames provided the basis for a new class of object-orientated languages such as KRL (Bobrow & Winograd, 1977), RLL (Greiner & Lenat, 1980) and KL-ONE (Brachman & Schmolze, 1985). Other programming environments have been developed, for example KEE (Kehler & Clemenson, 1984) and LOOPS (Stefik *et al*, 1983), which combine production rules and frames, thus exploiting the strength of rules in representing knowledge of actions and the strength of frames in representing large bodies of static knowledge.

Frame-based systems are best suited to solving classification problems (including diagnosis) since they provide an explicit and efficient method of representing the attributes of prototypical objects, against which particular instances of objects can be matched.

3.3.2 Structure

A knowledge frame contains a structured description of an object or class of objects. Frames can be organized into taxonomies in which a link between two frames indicates that one is either a member or a subclass of the other. Features of an object are represented by slots in the frame; there are two main classes of slots. Where a frame represents a class of objects, slots can refer to features of the class itself (these will be referred to as *generic slots*), or to properties of the class members (*member slots*).

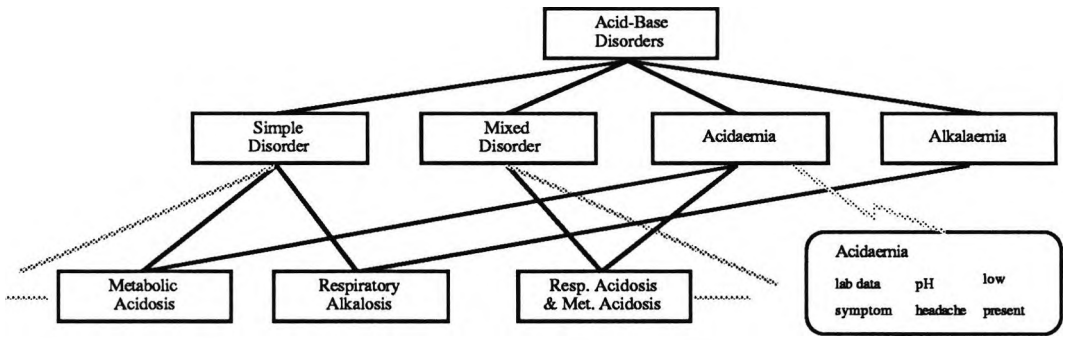


Figure 3.5 A Frame Taxonomy. (Note that a frame can be a member of more than one class)

Frames which represent individual objects have only generic slots, describing features of the object. Each slot has a set of facets, which are properties of the slot, and each facet has a value. A frame taxonomy and an example of a frame with two slots is shown in Figure 3.5. Notice that a frame can be a member of more than one class simultaneously.

An important feature of frame-based representation is the ability for frames to inherit features through the taxonomy. This means that features common to an entire class of objects need only be stored once - as a member slot in the frame for that class. Since frames may be objects in more than one class, there could be a conflict if a frame inherits the same slot from two different sources; in this case the precedence for inheritence must be specified.

3.3.3 Control

The purpose of the frame taxonomies in many systems is to exploit the properties of inheritence to achieve efficient knowledge representation; the reasoning process is achieved by an external inference engine operating on that knowledge base (Fikes & Kehler, 1985). Some reasoning mechanisms can be incorporated into the frame structure itself by the use of *message passing, methods* and *demons*. Messages can be passed between frames to activate methods attached to the receiving frame. A method is a procedure that performs a sequence of operations which may depend on the content of the activating message. Demons are methods that are attached to particular frame slots and are activated when the slot is accessed.

The structured representation of a rule in a rule-based system, shown in Figure 3.2, can be viewed as a frame for the rule and the CONTEXT tree in a system such as MYCIN corresponds to the frame hierarchy. The control mechanism of a rule-based system could also be replicated using messages, methods and demons. The premise conditions for the rule can be placed as a demon in the premise slot of the frame. When the slot is accessed, the demon is evaluated, sending a message to its own frame to execute the rule action, stored as a method for the frame. The action method then sends messages to other frames in order to update the database and trigger further rules.

When a frame-based system is used for classification problems, the observed data define a frame for the object to be classified. This frame is then matched with the prototypical frames in the knowledge base to find the most likely classification. One control strategy that manipulates frame-based knowledge in this manner was used in the Present Illness Program, described in Section 2.2.4.

3.3.4 Summary

Knowledge bases organized as a hierarchy of frames provide an efficient way of representing both declarative and procedural knowledge. A frame is a structured description of an object or class of objects, comprising slots, facets and values. Declarative knowledge of the attributes of an object is represented as <slot, facet, value> triples in its frame; procedural knowledge is represented by attaching methods and demons to frames. Frame-structured knowledge bases can be used with an external inference engine to solve classification problems - they can also replicate the function of a rule-based system by using messages, methods and demons.

3.4 Semantic Networks

3.4.1 Background

Semantic networks were developed as a model of the long-term human memory structures that represent the meaning of English words (Quillian, 1967). This model was built up using *nodes,* connected by *associative links*, to form a network structure. At a basic level of representation, semantic networks can be coded as <node-association-node> triples, but it is more usual to think of them in terms of *graphical analogues of data structures representing "facts"* (Schubert, 1976). A simple network of nodes and links is shown in Figure 3.6.

In the network proposed by Quillian (1967) each node was identified by an English word and could be one of two varieties. *Type* nodes were linked to a sub-network, or plane, of *token* nodes representing the meaning of the word. Each token node had a link to its own type node - tokens for the same word could appear repeatedly throughout the network as members of different planes. Thus the whole structure could be thought of as a three-dimensional space of nodes and links in which a series of intersecting planes drawn through the space represented the meanings of words.

3.4.2 Reasoning in a Semantic Network

A semantic network is a static body of declarative knowledge which can be manipulated by a separate inference engine to perform particular functions. Indeed, the same network could be manipulated in several different ways to achieve different goals, although in general the stucture of the network will be influenced by the intended use and method of inference. The early semantic network described above was used to compare and contrast the meanings of words by searching for the possible pathways between a pair of type nodes that existed in the network.

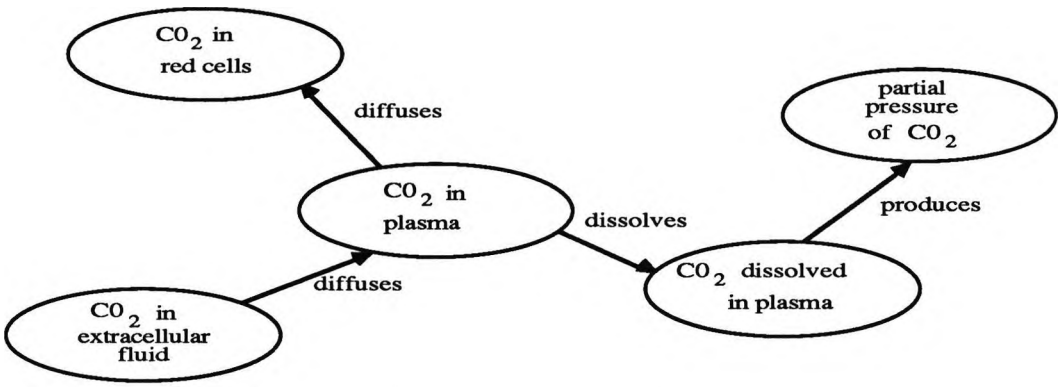


Figure 3.6 A Simple Semantic Network.

Starting from each node simultaneously, the search explored links, tagging each new node encountered with the name of its predecessor in the path and the name of the start node. When a node was encountered on the path from the first node, that had already been tagged with the name of the second, a connection had been found and the entire pathway could be generated by advancing to the tagged predecessors on the path from the second node. This process can be described as a *breadth first intersection* search on the network. Such a search could be used in the network shown in Figure 3.6 to answer the question *How does CO_2 in extracellular fluid affect pCO_2?* with the pathway *CO_2 in extracellular fluid diffuses to CO_2 in plasma which dissolves and gives rise to pCO_2*

The nodes in Figure 3.6 can be thought of as simple *concept nodes* representing basic concepts in the domain of the semantic net. Another useful type of node is the proposition node which represents relationships between concept nodes. The proposition node can be viewed as a predicate with links to concept nodes that form its arguments; these links have been referred to as case roles (Fillmore, 1968). An example for the chemical reaction of water and carbon dioxide in red blood cells is shown in Figure 3.7.

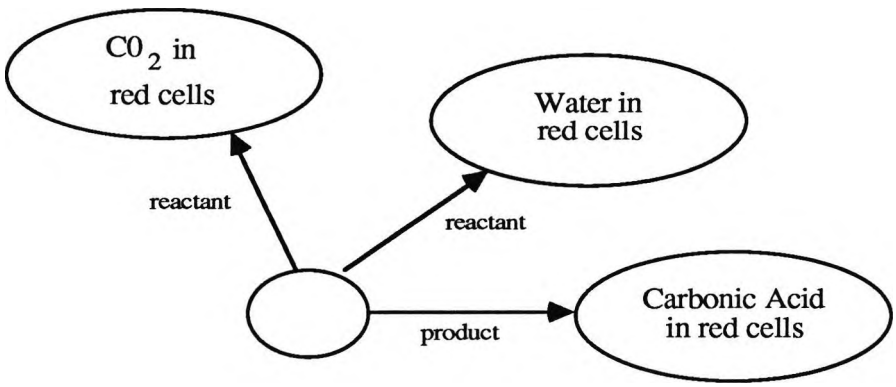


Figure 3.7 The Proposition Node for the Reaction Between H_2O and CO_2

It may be useful to refer to the structure in Figure 3.7 as *the reaction between carbon dioxide and water in red cells* and to use this aggregation as a concept node. This introduces the notion of partitioning the network into spaces (Hendrix, 1975) , where a space consists of a sub-network of nodes and any one node can be a member of several spaces. A partitioned network is shown in Figure 3.8, containing the space in Figure 3.7. The problem of expressing such concepts as quantification and logical connection in semantic networks has been addressed by extending the ideas of proposition nodes (Schubert, 1976) and partitioning (Hendrix, 1979).

The networks presented so far could be used to generate detailed, deep level explanations of a domain by fairly straightforward tracing of pathways; the arcs in these networks act as simple descriptive links between nodes and no restrictions are placed on the arcs that can be used. Other semantic networks have made use of a limited set of arc types which convey specific information to the network interpreter as it explores paths through the nodes and arcs; the networks in CASNET, ABEL and CHECK that were described in Chapter 2 would fall into this category.

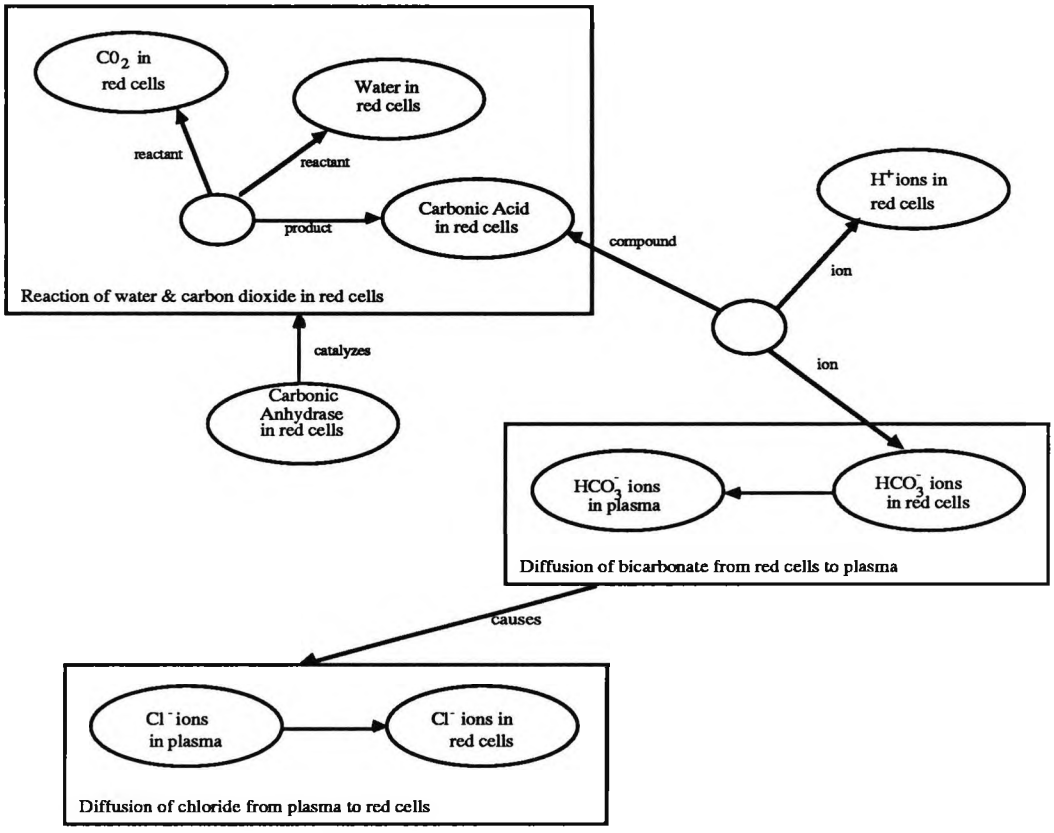


Figure 3.8 A Partitioned Network Containing Figure 3.7 as a Space

One of the most commonly used (and misused) types of arc is the IS-A link. It can form the connection between two concept nodes where one is a sub-classification of the other and allows the inheritance of properties. The two basic types of IS-A link connect classes to sub-classes and classes to individual elements; there are also other, more subtle differences in the possible meanings of IS-A links that should be considered when constructing networks (Brachman, 1983). A semantic network containing IS-A links bears more than a passing resemblance to the frame-based representation described in Section 3.3.

3.4.3 Summary

Semantic networks of nodes linked by associative arcs were developed in the late 1960s as a method for representing the meanings of English language words. Knowledge can be retrieved from semantic networks by simple tracing of pathways between nodes. More sophisticated analysis can be performed by the use of propositional nodes, partitioning or definition of special arc types that affect the way in which the network is manipulated.

3.5 The Blackboard Architecture

3.5.1 Origins

In the late 1950s, Oliver G. Selfridge proposed a model in which a collection of independent processing modules could interact in parallel to solve complex pattern recognition problems, such as the production of type-written text from spoken input (Selfridge, 1959). In its most elementary form, the model comprised a collection of *demons* each of which was designed to recognize a particular pattern and to indicate the extent to which this pattern matched the input data. A control demon had the task of deciding which of the 'shrieking demons' had the loudest output; the whole ensemble being called Pandemonium. A slightly refined model was used in practice, since it was found that many of the patterns to be recognized had similar features.

In the arrangement of Pandemonium shown in Figure 3.9, the pattern recognition process was split into four levels. At the lowest level were demons which held the input data; at the second level were *computational demons* which extracted features from the data; at the third level were *cognitive demons* and at the highest level was the *decision demon* which selected one of the cognitive demons as output. The cognitive demons were weighted, linear combinations of the output from the computational demons and were in effect hypotheses of the problem solution.

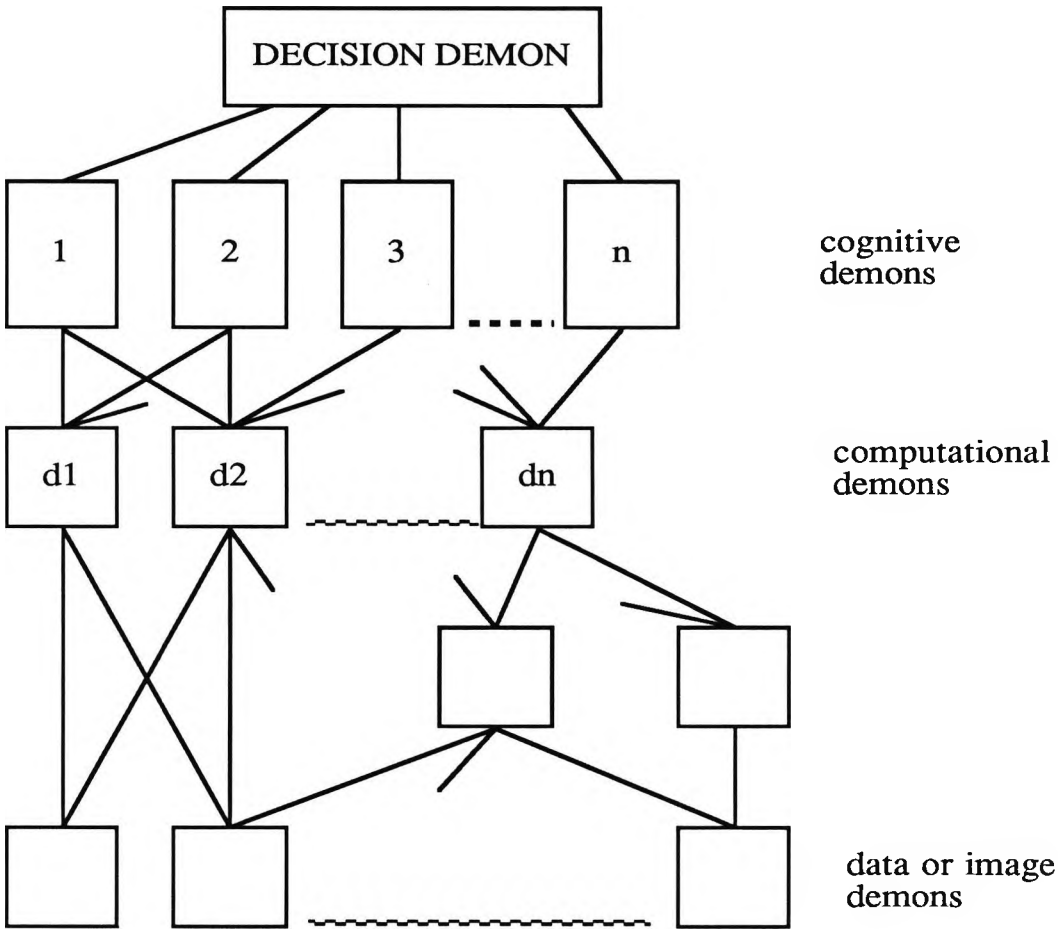


Figure 3.9 Demons in Pandemonium. Each cognitive demon is a weighted, linear combination of the set of computational demons. (Adapted from Selfridge, 1959)

In 1962, Allen Newell described a problem solving strategy in terms of a group of workers viewing a blackboard on which is written the emerging solution to the problem; each worker can see from the blackboard when he has something useful to contribute (Newell, 1962). Newell went on to point out that this is precisely the idea behind the Pandemonium model. Newell's colleague Herbert Simon is reported (Nii, 1986) to have suggested the use of this blackboard model to the designers of the Hearsay II speech understanding system (Reddy *et al*, 1973), which became the inspiration behind a multitude of subsequent problem-solving systems.

3.5.2 The Hearsay-II System

The Hearsay systems were developed at Carnegie-Mellon University as part of the Speech Understanding Systems Program organized by the Advanced Research Projects Agency (ARPA) of the US Department of Defence between 1971 and 1976. The first Hearsay model was demonstrated in June 1972 and claimed to be *the first system to demonstrate live, connected speech recognition using nontrivial syntax and semantics* (Reddy *et al*,

1973). Although many of the features of a *blackboard system* are evident in this early version of Hearsay, it was not until the development of Hearsay-II (Erman & Lesser, 1975) that the blackboard model was explicitly defined.

Hearsay-II was designed to handle two types of uncertainty inherent in the process of speech understanding; uncertainty introduced by imperfect speech production and detection, and uncertainty introduced during the process of understanding the detected signal. In order to achieve this, Hearsay-II had to be capable of entertaining a series of hypotheses at every level, from the processing of raw signal data, through the formation of words and phrases, to a complete interpretation of the spoken utterance. Creating and refining these hypotheses would require the interaction of many diverse sources of knowledge.

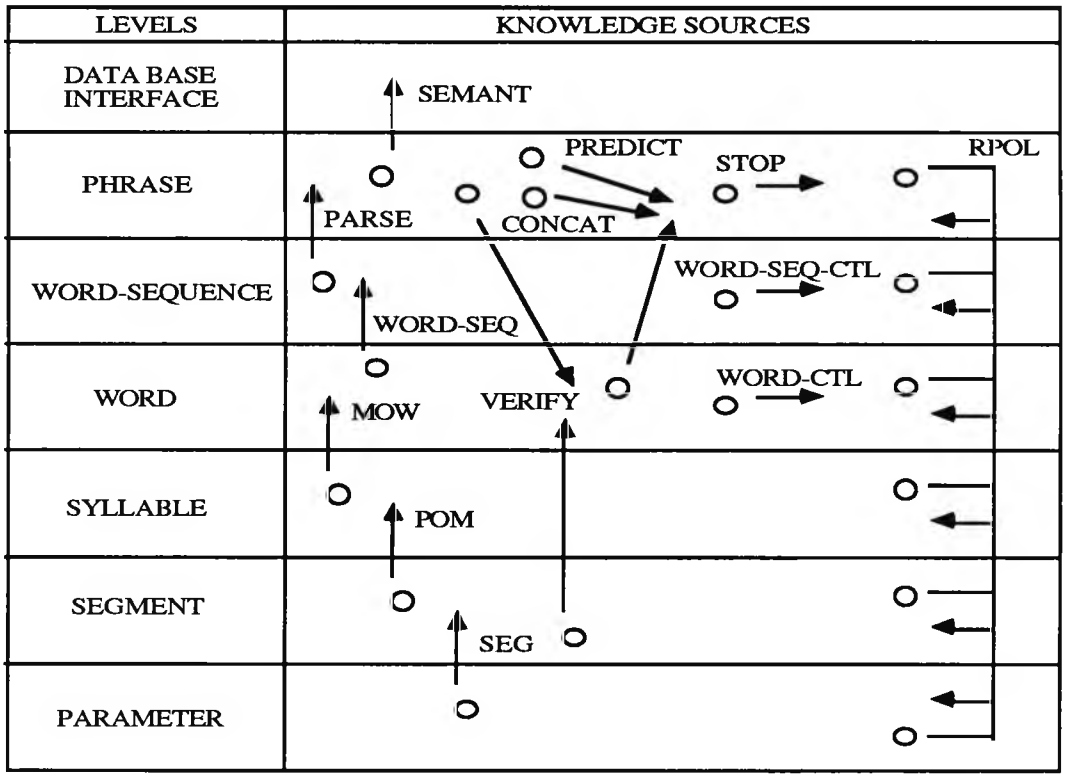


Figure 3.10 Blackboard Levels and Knowledge Sources in Hearsay-II. POM creates syllables from segments, MOW creates words from syllables, WORD-CTL controls the number of words created, WORD-SEQ creates word sequences, WORD-SEQ-CTL controls the number of word sequences created, PARSE parses word sequences to form phrases, PREDICT predicts words that follow phrases, VERIFY rates segments against word-phrase hypotheses, CONCAT joins word-phrase pairs, RPOL rates hypotheses, STOP decides when to finish, SEMANT creates the final, unambiguous interpretation. (Adapted from Erman *et al*, 1980)

The *knowledge sources* were defined as condition-action pairs; the condition module specified the circumstances in which the action could be executed, so as to contribute to the problem solving process. The knowledge sources were independent modules (typically 30 pages of Algol-like code) whose only inter-communication was through the blackboard - a global database of hypotheses split into a hierarchy of discrete levels, ranging from raw signal data at the lowest level to a definite interpretation of the spoken utterance at the highest. The knowledge sources and blackboard levels, as they existed in September 1976, are shown in Figure 3.10.

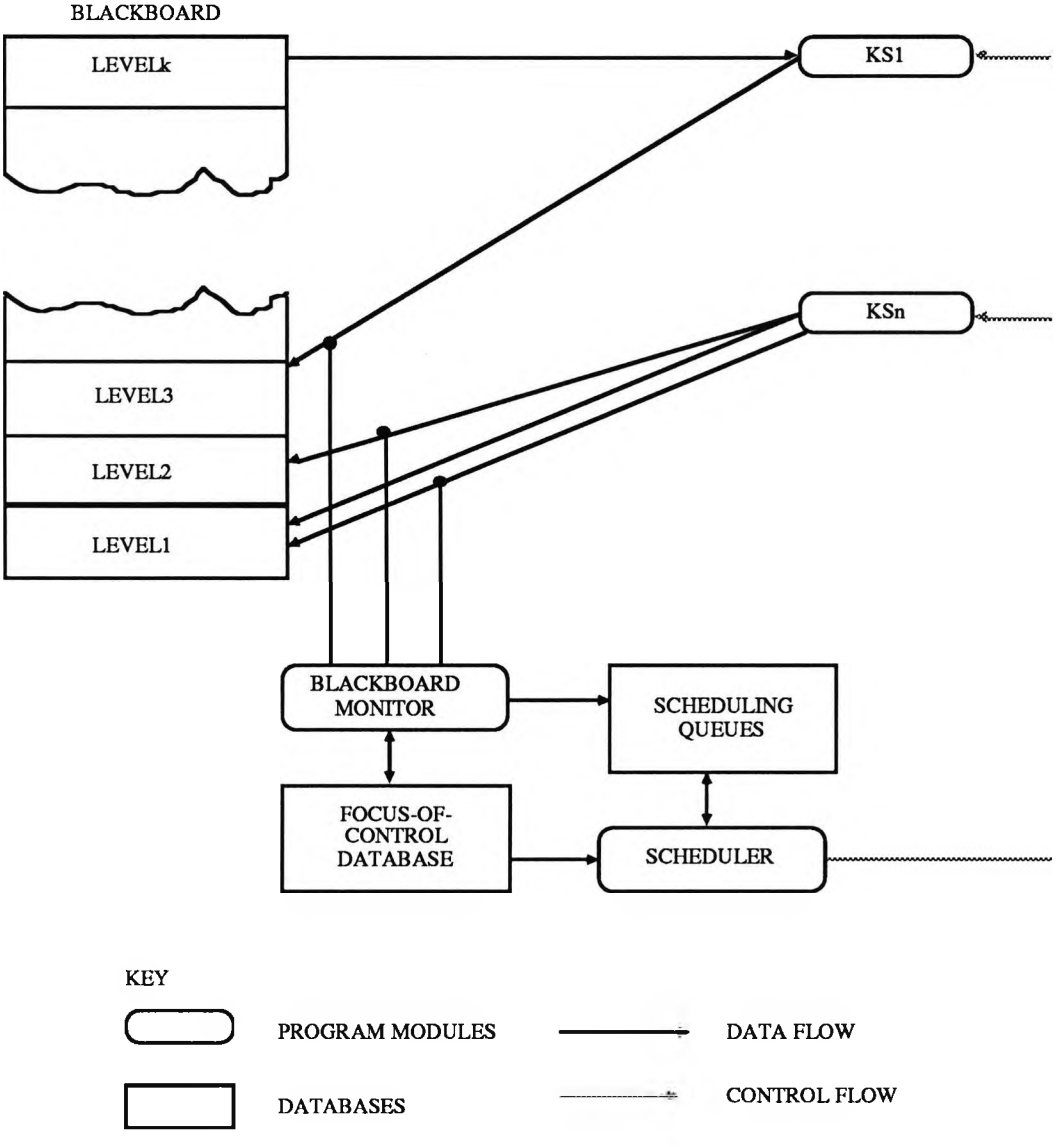


Figure 3.11 The Structure of Hearsay-II (Adapted from Erman *et al*, 1980)

Hearsay-II operated in a cyclic fashion. A *scheduling queue* of activities was maintained - an activity being either the execution of a condition module or of an action whose condition had already been successfully evaluated. On each cycle the *scheduler* selected an activity to perform by finding the highest priority amongst the members of the scheduling queue. When a knowledge source condition module was selected, the current data on the blackboard were used to find all possible instantiations of the conditions - the set of which was called the *stimulus frame*. The corresponding action module of the knowledge source was then added to the scheduling queue. When an action module was selected as the next activity, the action itself was performed and any resultant changes made to the blackboard were noted by a blackboard monitor. Any knowledge source condition modules affected by these blackboard changes were added to the scheduling queue. When rating the activities in the scheduling queue, Hearsay-II used the stimulus frame mentioned above, the response frame (a description of the likely effect of executing a knowledge source action), and general information on the state of the problem solution, such as the strength of the hypotheses at each level of the blackboard and the amount of processing time being used. Figure 3.11 shows the overall architecture of the HEARSAY-II system.

3.5.3 The HASP Project

Work on the HASP system (Nii *et al*, 1982), developed jointly by the Stanford Heuristic Programming Project and Systems Control Inc, was begun in 1972 with the aim of identifying ocean-going vessels primarily using information gathered by arrays of underwater sonar detectors, but also utilizing information such as the location of shipping lanes and intelligence reports on ship movements. The sonar detectors produced a series of frequency spectra of the received signal over time. A persistent source of constant frequency would produce a solid line trace on a frequency vs time display. Rotating shafts, propellors and on-board machinery create a distinctive *sonar footprint* for a particular class of vessel, but the problem of identifying the class, speed and course of several vessels from a noisy picture of their overlapping footprints is by no means easy.

The blackboard in the HASP system was called the *current best hypothesis* of the situation. It was split into four levels - at the lowest level were *sonogram lines*, at the next level were *harmonics* (groupings of the lines), then *signal sources* (*eg* propellors, shafts, etc) with their estimated position, and at the highest level were *hypotheses* of the vessels present, with their class, velocity and position. Nodes at different levels were linked so that the entire structure formed a hierarchy of hypotheses. There were two types of link: expectation links connected high level nodes to nodes suggested by them at lower levels and reduction links connected low level nodes to aggregations at higher levels.

The knowledge sources were rule-based modules which were organized into a three level hierarchy. *Specialist* knowledge sources made modifications to the blackboard, knowledge source *activators* were responsible for selecting which specialists to execute and at the head of the hierarchy was the *strategy* knowledge source responsible for overall control of the system. There were four types of knowledge source activators; the *clock event manager*, *expectation event manager*, the *problem event manager* and the *blackboard event manager*. Each type was associated with a list of events. The blackboard event list recorded all changes made to the blackboard; the expectation event list contained events that were expected at some future, unspecified time (*eg* intelligence reports that a particular vessel might be present); the problem event list held information which was not available at the time a knowledge source was executed but which would have been useful to know (the knowledge source could then be re-executed when the information became available); the clock event list held the type and time of predicted events (*eg* the temporal characteristics of the sonograms of some vessels were known). A diagram of the structure of the HASP system is shown in Figure 3.12.

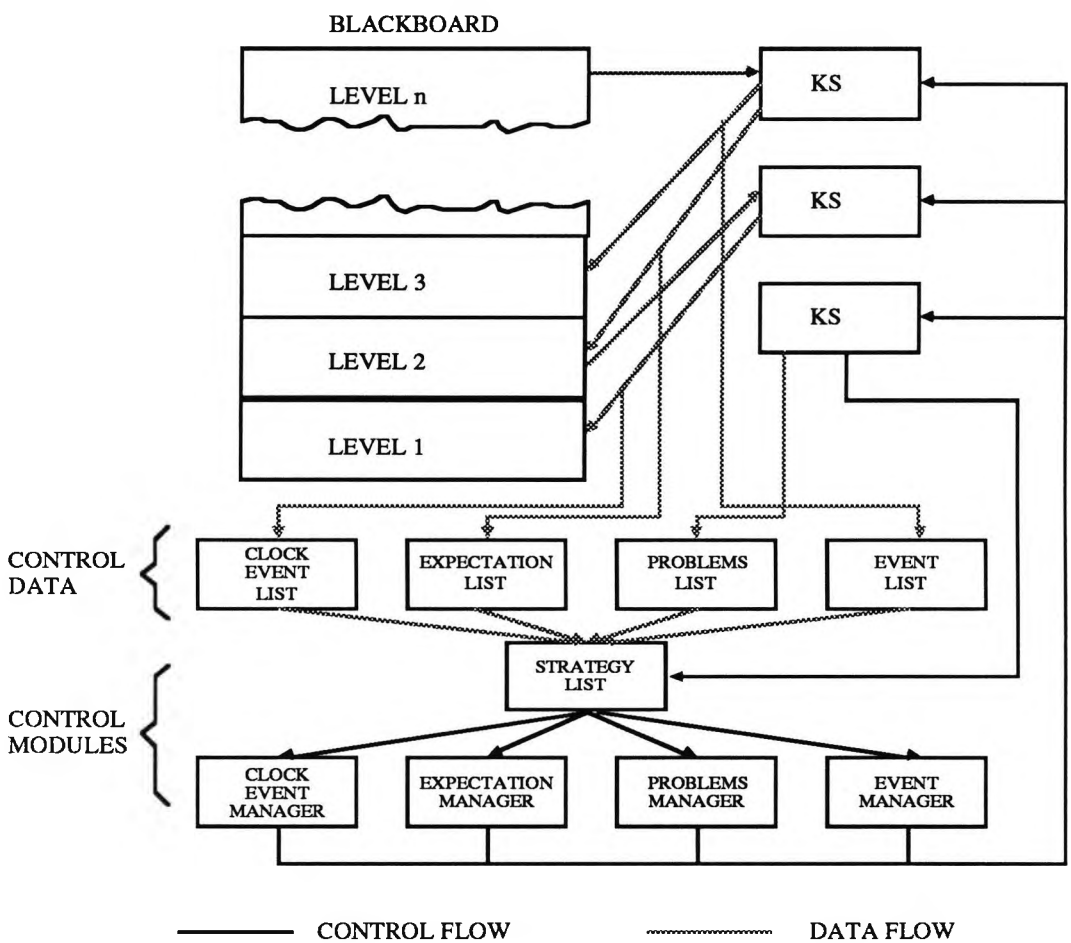


Figure 3.12 The HASP/SIAP System

On each cycle, the strategy knowledge source selected a knowledge source activator, which in turn selected a node from the appropriate event list to be the next *focus of attention*. The focus of attention was used to select and execute all the relevent knowledge source specialists which made changes to the blackboard and event lists so that the cycle could begin again.

One of the most interesting aspects of HASP was its use of top-down, model-driven reasoning, wherby important features of a model (in this case represented as a knowledge frame) were searched for in the lower level data. This feature of control was built into many of the later blackboard systems and indeed the Hearsay-II model itself was adapted to incorporate both data-directed and goal-directed scheduling (Corkill *et al*, 1982). The HASP project was concluded in 1975, but work was continued by the commercial partner as the SIAP project until 1980. It is reported (Nii *et al*, 1982) that an evaluation of the HASP/SIAP system produced results comparable with, and in some cases better than, the human experts.

3.5.4 Multiple Blackboard Panels

The CRYSALIS system (Englemore & Terry, 1979) extended the blackboard model by introducing a dual panelled blackboard. The system's task was to construct a three dimensional model of proteins, based on information derived from x-ray diffraction called electron density maps (EDMs). The knowledge sources and control modules were all rule-based and the control structure consisted of strategy, activator and specialist knowledge sources operating in the same manner as HASP. However, there were no condition modules in the knowledge sources to check whether they were suitable for execution - in this sense the process was not truly opportunistic, leading to doubts about classifying CRYSALIS as a blackboard system (Nii, 1986b).

The dual panelled blackboard comprised a density plane and a model plane. The density plane held information concerning the electron density map and was split into four levels - the lowest, parametric level held the raw EDM data and the higher levels contained abstractions derived from the parametric level. The model plane contained a three level hypothesis of the protein structure; the atomic level indicated the spatial location of atoms, the superatomic level described groups of atoms and the stereotypic level held details of large parts of the protein structure which formed recognized structures. Nodes on the levels of each blackboard panel were linked to form a hierarchy of hypotheses within each panel; there were also links between data on the density panel and the hypotheses which these supported on the model panel.

3.5.5 Generalization

Within the last decade, the designs of many AI systems have been based to varying degrees on the blackboard model. Primarily these systems have been concerned with the transformation of raw signal data into a symbolic description, although some have had other functions - the planning system OPM (Hayes-Roth *et al*, 1979) being a good example. As well as these specialist blackboard systems there have also been attempts to produce general expert system shells which implement a domain independent blackboard model. One of the designers of the Hearsay systems went on to collaborate in the production of Hearsay-III which *can be viewed as an extension along some directions of the Hearsay-II architectural style, and as a generalization of it along others* (Erman *et al*, 1981).

At Stanford University, the home of HASP and CRYSALIS, the AGE system (Attempt to Generalize) was intended to ease the process of knowledge engineering by creating a suite of useful software tools (Nii & Aiello, 1979). Though originally conceived as more than a general blackboard shell, the first task undertaken was to implement a set of tools for construction of blackboard systems. Facilities were provided to accommodate hierarchical blackboard structures with expectation and reduction links as in HASP, which could be arranged in multiple planes as in CRYSALIS. The knowledge sources were sets of rules with associated triggering conditions, lists of hypothesis levels spanned and links created by the knowledge source, and means of binding variables in the knowledge source when triggered. A mechanism for handling uncertainty in the rules was included; it was also possible to incorporate user-defined uncertainty handlers. Control modules were provided to provide both model-driven and data-driven strategies.

A particularly interesting feature of AGE was the use of an intelligent front-end to the system which not only made general help facilities available but also guided the user through the process of construction of an expert system. AGE was used to develop a number of systems from scratch and also to reimplement CRYSALIS and the rule-based system PUFF (Aikins *et al*, 1983), originally developed using EMYCIN (see Section 2.3). Work on AGE ceased in 1983 but the experience gained contributed in part to the BB1 system described below.

The BB1 system, developed by the Heuristic Programming Project at Stanford University, is a practical implementation of a *blackboard architecture for control* (Hayes-Roth, 1985). It provides the basic mechanisms of a blackboard system and tools for creating and editing blackboard structures and knowledge sources. There is a low-level blackboard-manipulation language which incorporates the ability to execute LISP functions.

Knowledge source actions and control mechanisms can be programmed using this language or in a language defined by the user, suitable for a particular domain application (this is the recommended way of programming in BB1). The blackboard structures supported by BB1 can be split into an arbitrary number of levels which contain objects composed of attribute-value pairs and links which can be used to connect the objects into hierarchical structures. Facts about objects are inherited through the hierarchy, making a blackboard structure similar to a frame-based knowledge base (see Section 3.3). A graphical editor, BBEDIT, is used to create and modify the blackboard structures.

There are two types of knowledge source in BB1 - domain or control - but they have a uniform structure consisting of 16 fields. Trigger conditions are evaluated and cause the knowledge source to become triggered if they are all true; at this stage context variables derived from the prevelant blackboard data are bound to form *knowledge source activation records* (KSARs); more than one KSAR can be created for each triggered knowledge source. When a KSAR is formed, a list of pre-conditions is evaluated; if they are all true then the KSAR is designated as *executable*, if not the KSAR is designated *triggered* and the unmet preconditions are rechecked on each appropriate cycle. Obviation conditions are evaluated in the same way as trigger conditions - when all the obviation conditions are true, the KSAR is removed from the system.

The knowledge source variables are a list of variable-expression pairs which are evaluated just before the knowledge source is executed. The knowledge source actions are the rule set which is evaluated when the KSAR is executed. Other knowledge source fields identify the blackboard from which triggering occurs and the blackboard(s) affected when the actions are evaluated, the levels worked from and acted upon, the cost and reliability of the knowledge source (used for control purposes as defined by the user) and several administrative fields such as the name, author and description of the knowledge source.

The operation of BB1 involves the characteristic cycle of blackboard systems: execute a knowledge source activation record, update the agendas of triggered knowledge sources and schedule the next KSAR for execution. The cycles on which the obviation and preconditions described above are checked can be defined by the user (the default is every cycle). The cycle proceeds as follows. The interpreter acts on the KSAR that has been chosen for execution, executing the actions and recording the blackboard events generated. The agenda maintainer then makes changes to the agendas of triggered, executable and obviated KSARs. If the cycle has been designated for checking preconditions, then this is done for all executable and triggered KSARs. If any of the preconditions of an executable KSAR are not true, the KSAR is returned to its triggered state; if all the preconditions of a triggered KSAR are true, it becomes executable. The events generated by the most recently

executed KSAR are used to generate new KSARs in either the triggered or executable states, depending on the evaluation of the preconditions. If the cycle has been designated for checking of obviation conditions, any KSARs whose obviation conditions are satisfied are removed to the obviation agenda, effectively eliminating them from the system. The interpreter selects the next KSAR for execution by rating each executable KSAR using the control mechanisms described below.

The control-plan blackboard is defined as part of the BB1 system and has four levels; *strategy* (highest), *focus*, *heuristic* and *schedule* (lowest). Objects at the strategy level describe the overall problem solving behaviour desired; objects at the focus level can be part of one or more strategies and define the attributes of KSARs that should be executed to achieve a specific local goal; at the heuristic level are defined specific LISP functions that can be used to rate KSARs. Finally an object is created at the schedule level on each cycle to describe the KSAR that was executed. Often a single strategy will be sufficient to guide the entire problem solving process. The strategy is used to make decisions on which focus (or foci) to persue; it is at the focus and heuristic levels that executable KSARs are rated. It has been shown (Hayes-Roth, 1985) that the blackboard architecture for control can replicate the control of other blackboard systems such as Hearsay-II and HASP, and also of systems which use meta-level control rules (see Section 3.2).

3.5.6 Summary

The major drawback of using a blackboard architecture for problem solving is that it introduces overheads that are expensive both in terms of computing resources and speed at run time. To offset this there are many advantages both as a means of analyzing a problem and as a method for developing a solution. The use of independent knowledge sources means that fields of knowledge, diverse in origin, content and representation, can be brought to bear on a problem and that these can be developed and adapted without affecting the rest of the system. This has the added advantage that experiments with different configurations of knowledge sources can easily be performed within the overall system framework. The blackboard model allows reasoning with uncertain and unpredictably incomplete data to produce competing solution hypotheses of varying degrees of certainty which can share the same data and sub-hypotheses. The blackboard database means that the problem solving process can be interrupted and resumed at will and allows for truely opportunistic problem solving behaviour, whereby the most appropriate data, knowledge and control strategies can be used to advance the most promising hypotheses towards the desired solution.

3.6 Data Classification

3.6.1 Introduction

The numerical value of a laboratory data measurement conveys no information in itself; in order to provide information it must be compared with some reference. The concept of reference values was introduced to describe the measurements taken from a reference population, where the precise nature of the population and the conditions under which the measurements were taken are specified (Grasbeck & Soris, 1969).

There are four ways in which variation in the value of a laboratory data variable can be introduced (Kringle & Johnson, 1986). *Inter-individual* variation stems from the differences (*eg* age, sex, general health, etc) that exist between the members of a population; *intra-individual* variation is due to changes in a particular individual over time (*eg* diet, emotional state, etc); *pre-analytical* variation is introduced by the manner in which samples are handled in the laboratory (*eg* storage conditions, processing in a centrifuge); *analytic* variation is introduced by the actual method of measurement. If care is taken to eliminate systematic errors (biases), the four sources of variation each introduce random errors and contribute to the formation of a Gaussian distribution of values in the reference population. The distribution may become skewed if some factor introduces a systematic bias in all measurements. If the factor responsible for the bias can be identified then the skewness can be corrected by partitioning the reference population. For instance the bias introduced by a subject's age could be eliminated by dividing the reference population into different age groups and calculating a separate reference range for each one. An alternative way to correct a skewed distribution in the reference population is to apply a transformation to the measured data values in order to present them as a standard Gaussian distribution.

Many different methods have been proposed for relating observed values to reference values in clinical chemistry (Dybkaer, 1981); the following sections describe three methods that have found application in knowledge-based systems.

3.6.2 Interval Classification

One of the simplest and most widely used methods of classification for laboratory data is to divide the range of possible values into intervals, using reference limits, and to classify data according to the interval in which they fall. The classification into three intervals can be made by setting two reference limits at the 95% confidence limits on a Gaussian distribution. This is shown in Figure 3.13.

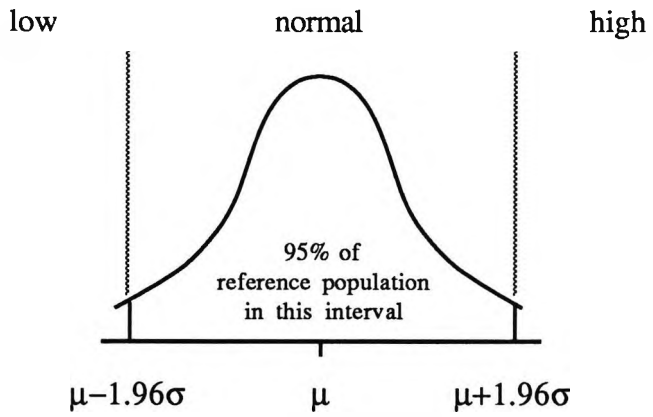


Figure 3.13 Three Interval Classification Using 95% Confidence Limits (Gaussian distribution with mean μ and standard deviation σ)

The disadvantage of this method is the enormous loss of information that it entails; for instance, values of μ-1.95σ and μ+1.95σ would both be classified as normal. The information loss can be reduced by increasing the number of classification intervals. Figure 3.14 shows the classification of pH into seven intervals - very low, low, fairly low, normal, fairly high, high and very high.

The reliance on the Gaussian distribution for interval classification has been criticized by many authors (*eg* Mainland, 1969) who advocate the use of non-parametric methods of determining reference limits (a non-parametric method makes no mathematical assumptions about the distribution of values in the reference population). One such method, recommended by the International Federation of Clinical Chemistry (1982) is to calculate the 0.95 interfractile interval, by removing the highest 2.5% and lowest 2.5% of values in the reference population and setting the reference limits at the highest and lowest remaining values.

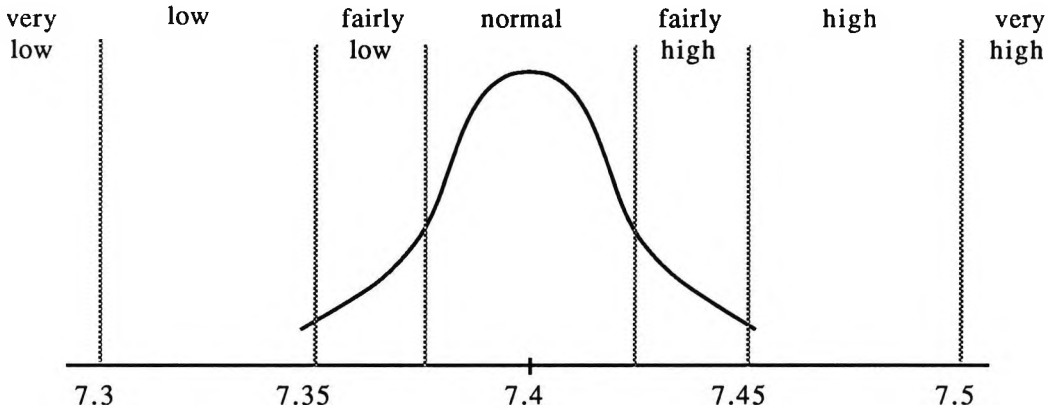


Figure 3.14 Seven Interval Classification of pH

3.6.3 Fuzzy Set Theory for Data Classification

Consider a space of elements, $U=\{u_1,u_2,u_3,....\}$ and a subset of elements $A=\{a_1,a_2,...a_n\}$ in that space. According to traditional set theory, each element a_i is a member of A and not a member of its complement A'. The theory of fuzzy sets (Zadeh, 1965) introduces the notion of elements that are partly members of A and partly members of A'. The degree to which an element u of the space U is a member of the fuzzy set A, is represented by a membership function $f(u_i)$ which assigns a number in the range [0,1] to each element u_i expressing its grade of membership of A. If $f(u_i)=1$ then u_i is a member of A; if $f(u_i)=0$ then u_i is not a member of A; if $0<f(u_i)<1$ then the degree to which u_i is a member of A increases with $f(u_i)$. The membership function of the complement of A, A', is defined as $1-f(u_i)$. It can be seen that a set in traditional set theory corresponds to a fuzzy set for which $f(u_i)=0$ or 1.

Fuzzy set theory provides a useful method for data classification. Returning to the problem of classifying pH values, three fuzzy sets can be defined; low pH (acidaemia), high pH (alkalaemia) and normal pH. Three membership functions can now be defined so that for any observed pH value, three measures are obtained, expressing the degree to which it can be considered as high, low or normal. The membership functions can take any form; two types that have been used in practice are described below.

Zadeh (1972) has defined some useful functions:

$$
\begin{aligned}
S1(u_i;\alpha,\beta,\chi) &= 0 && \text{for } u_i\leq\alpha \\
&= 2((u_i-\alpha)/(\chi-\alpha))^2 && \text{for } \alpha\leq u_i\leq\beta \\
&= 1 - ((u_i-\chi)/(\chi-\alpha))^2 && \text{for } \beta\leq u_i\leq\chi \\
&= 1 && \text{for } u_i\geq\chi \\
& && \text{where } \beta=(\alpha+\chi)/2
\end{aligned}
$$

$$
\begin{aligned}
S2(u_i;\alpha,\beta) &= S1(u_i;\beta-\alpha,\beta-\alpha/2,\beta) && \text{for } u_i\leq\beta \\
&= 1-S1(u_i;\beta,\beta+\alpha/2,\beta+\alpha) && \text{for } u_i>\beta
\end{aligned}
$$

These can be used to define three membership functions for high, low and normal pH, shown graphically in Figure 3.15a:

low pH	$1-S1(u_i;7.25,7.35,7.45)$
normal pH	$S2(u_i;0.1,7.4)$
high pH	$S1(u_i;7.35,7.45,7.55)$

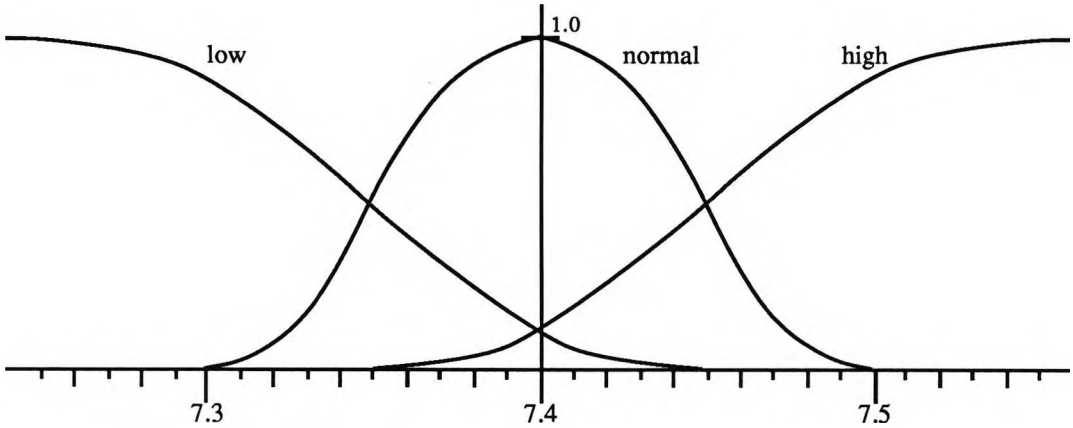


Figure 3.15 (a) Membership Functions for Low, Normal and High pH, Using Zadeh's Functions

Alternatively, a simple ramp function could also be used:

$$
\begin{aligned}
S3(u_i;\alpha,\beta,\chi,\delta) &= 0 && \text{for } u_i \le \alpha \\
&= (u_i-\alpha)/(\beta-\chi) && \text{for } \alpha < u_i \le \beta \\
&= 1 && \text{for } \beta < u_i \le \chi \\
&= (\delta-u_i)/(d-c) && \text{for } \chi < u_i \le \delta \\
&= 0 && \text{for } u_i > \delta
\end{aligned}
$$

Figure 3.15b shows the membership functions defined by:

low pH	$S3(u_i;0,0,7.3,7.4)$
normal pH	$S3(u_i;7.3,7.4,7.4,7.5)$
high pH	$S3(u_i;7.4,7.5,10,10)$

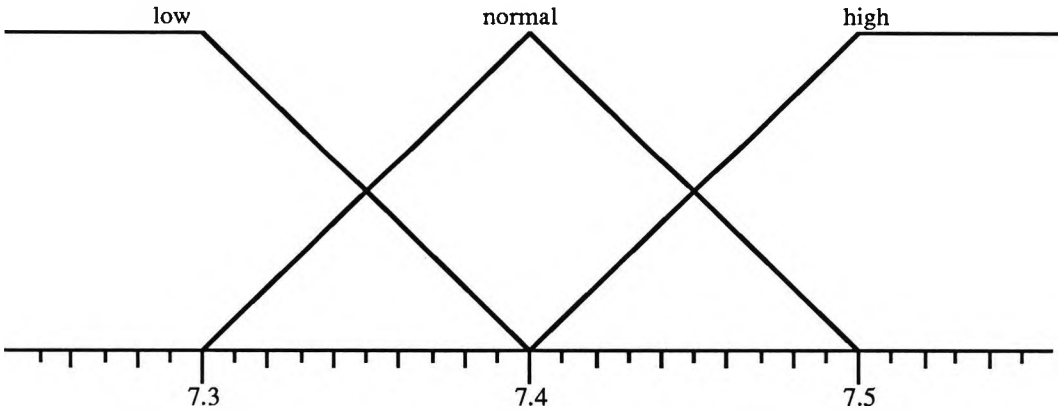


Figure 3.15 (b) Membership Functions for Low, Normal and High pH. Using a Simple Ramp Function

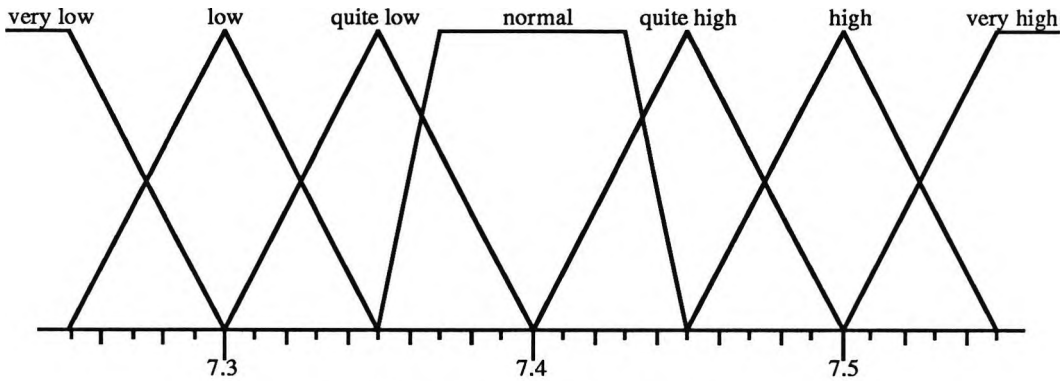
Fuzzy set theory offers two methods to increase the number of groups (fuzzy sets) into which data are classified. The first method is simply to define additional fuzzy sets and associated membership functions. This approach has been used in a knowledge-based system for the assessment of liver function (Lesmo *et al*, 1984) which uses the classifications normal, slightly altered, altered and very altered. The classification of pH according to this scheme is shown in Figure 3.16a using the membership functions:

very low	S3(0,0,7.25,7.3)
low	S3(7.25,7.3,7.3,7.35)
quite low	S3(7.3,7.35,7.35,7.4)
normal	S3(7.35,7.37,7.43,7.45)
quite high	S3(7.4,7.45,7.45,7.5)
high	S3(7.45,7.5,7.5,7.55)
very high	S3(7.5,7.55,10,10)

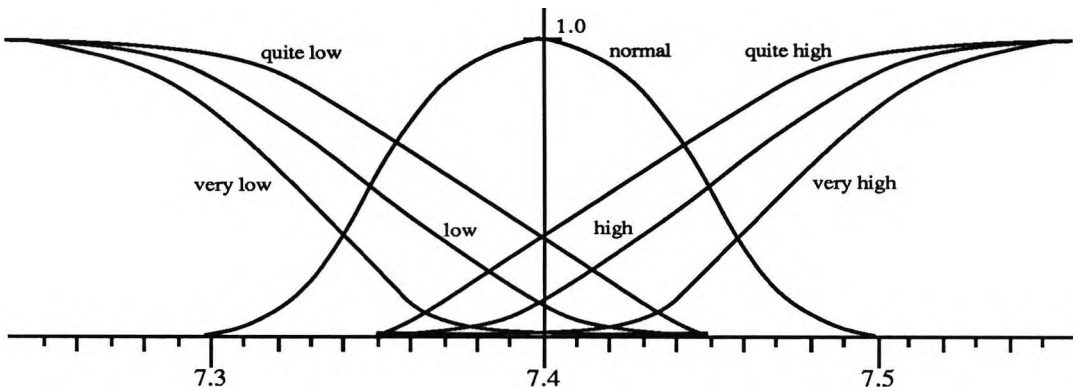
A second method for classification into the same seven groups defined above, is to consider the terms *very* and *slightly* as *modifiers* or *linguistic hedges* (Zadeh, 1972). Such modifiers define a transformation for the membership function of a fuzzy set. For instance, if u_i is a member of A according to membership function $f(u_i)$, then it is a member of very.A according to $[f(u_i)]^2$ and it is a member of quite.A according to $[f(u_i)]^{0.5}$

Figure 3.16b shows the membership functions defined by:

very low	$[1\text{-}S1(u_i;7.25,7.35,7.45]^2$
low	$1\text{-}S1(u_i;7.25,7.35,7.45)$
slightly low	$[1\text{-}S1(u_i;7.25,7.35,7.45)]^{0.5}$
normal	$S2(u_i;0.1,7.4)$
slightly high	$[S1(u_i;7.35,7.45,7.55)]^{0.5}$
high	$S1(u_i;7.35,7.45,7.55)$
very high	$[S1(u_i;7.35,7.45,7.55)]^2$



(a) Defining Further Ramp Functions



(b) Using Linguistic Hedges

Figure 3.16 Increasing the Fuzzy Sets for Classification

It is tempting to think of fuzzy classifications in terms of the probability that u_i is low, very low etc. It can be seen, however, that $\Sigma f(u_i)$ is not equal to unity in the above definitions, as it would have to be for a probabilistic interpretation. Indeed, *the notion of a fuzzy set is completely non-statistical in nature* (Zadeh, 1965) and special rules of fuzzy inference must be applied when fuzzy classification is employed as part of a knowledge-based system (see Section 3.7.5). The next section describes a method in which probability theory is used to derive classifications similar to those of fuzzy set theory.

3.6.4 Classification Using Probability Distributions

It has been proposed (Cheeseman, 1985) that a probabilistic method of classification exists, similar to the method using fuzzy set theory described above. The probability that a data value will be classified in a particular interval can be expressed as a series of probability distributions. Such a method has been used in the knowledge-based system MUNIN (see Section 2.4.2) and an example from this system for the classification of the duration of motor unit potential into six intervals is shown in Figure 3.17.

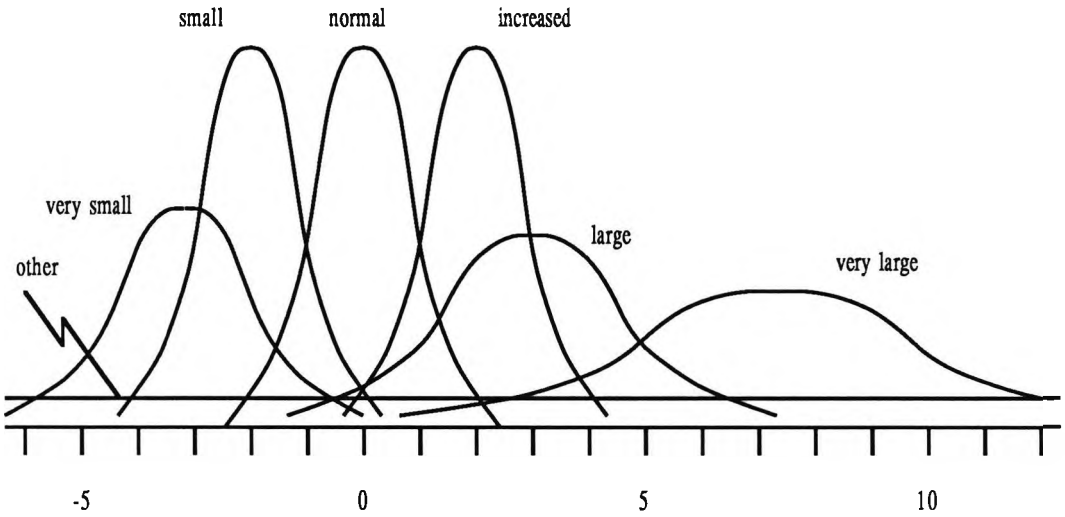


Figure 3.17 Probability Distributions in MUNIN

One major problem associated with the use of probability distributions for data classification lies in the definition of the distributions themselves. A method of generating the distributions for a classification into three intervals has been proposed, using the concept of an *individual difference quotient* (Tango, 1981). The derivation of the individual difference quotient for a laboratory test is based on the assumption that the Gaussian distribution of values observed in a reference population, $N(\mu,\sigma^2)$, is due to inter-individual and intra-individual differences in the population (analytical variation is assumed to be negligible). The intra-individual variation of observed values for a subject i, is represented by the distribution $N(\mu_i,\sigma_i^2)$ and the assumption is made that $\sigma_i=\sigma_e$ for all subjects *ie* inter-individual differences are restricted to variation in the mean value for each subject.

The value v_i obtained in a laboratory test on subject i can be thought of as the sum of the reference population mean, the difference between the subject's mean and the reference population mean (β_i) and the difference between the value itself and the subject's mean (e_i). This is illustrated in Figure 3.18.

If it is assumed that the inter-individual differences are distributed as $N(0,\sigma_\beta^2)$ then the variation in the reference population can be written as:

$$\sigma^2 = \sigma_\beta^2+\sigma_e^2 \qquad (3.6.1)$$

and the individual difference quotient θ can be defined for the laboratory test as:

$$\theta = \sigma_\beta/\sigma_e \qquad (3.6.2)$$

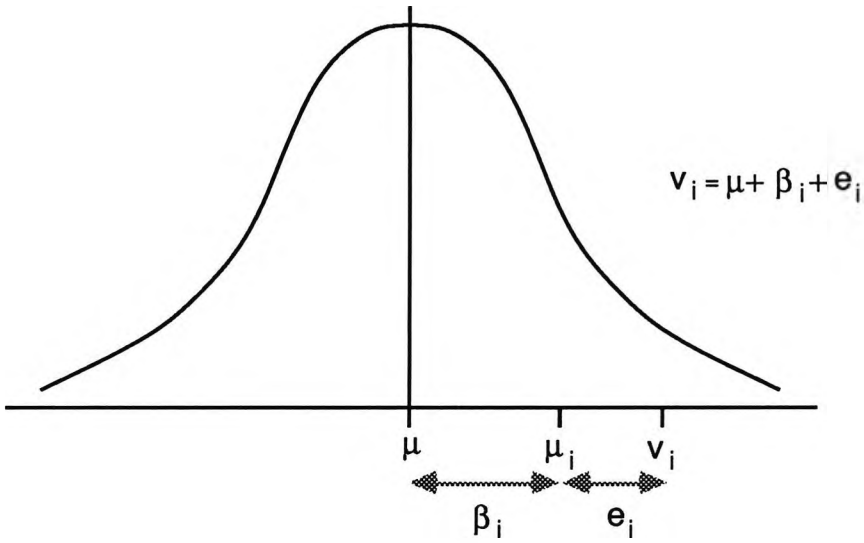


Figure 3.18 Inter- and Intra-Individual Differences

The value of θ has been measured for 23 laboratory tests in a population of 24 carefully selected, healthy patients and was found to lie in the range 0.5 to 2.5 (Tango, 1981). In 17 of the 23 tests $\theta>1$ indicating that inter-individual differences outweighed intra-individual differences.

For the classification of test data, consider the value v_i of the laboratory data variable V, that has been measured in a subject and that lies t standard deviations above the population mean *ie*

$$v_i = \mu + t\sigma$$

If the reference range for V is defined in the intra-individual distribution as $\mu_i - 2\sigma_e \leq V \leq \mu_i + 2\sigma_e$, then the probability that v_i is normal is given by the probability that μ_i lies within +/- $2\sigma_e$ of v_i. The distribution of μ_i is known to be $N(\mu, \sigma_\beta^2)$ and so

$$P(v_i \text{ is normal}) = \int_{v_i - 2\sigma_e}^{v_i + 2\sigma_e} \phi(u | \mu, \sigma_\beta^2)\, du$$

where ϕ is the Gaussian distribution function

substituting $u' = u - \mu_i$

$$P(v_i \text{ is normal}) = \int_{v_i - \mu_i - 2\sigma_e}^{v_i - \mu_i + 2\sigma_e} \phi(u' | 0, \sigma_\beta^2)\, du'$$

$$= \int_{t\sigma-2\sigma_e}^{t\sigma+2\sigma_e} \phi(u'|0,\sigma_\beta^2)\,du'$$

$$= \Phi\left(\frac{t\sigma+2\sigma_e}{\sigma_\beta}\right) - \Phi\left(\frac{t\sigma-2\sigma_e}{\sigma_\beta}\right)$$

where $\Phi=N(u|0,1)$ the standard Gaussian distribution function

but $\sigma^2=\sigma_\beta^2+\sigma_e^2$ and $\theta=\sigma_\beta/\sigma_e$ from (3.6.1) and (3.6.2) and hence

$$P(v_i \text{ is normal}) = \Phi\left(\frac{t\sqrt{1+\theta^2}+2}{\theta}\right) - \Phi\left(\frac{t\sqrt{1+\theta^2}-2}{\theta}\right)$$

Similar expressions can be derived for $P(v_i$ is low) and $P(v_i$ is high) enabling the classification of V into three intervals when θ is known for the laboratory test. The probability that the observed value is normal as t varies from 0 to 2 is shown in Figure 3.19 for several values of θ.

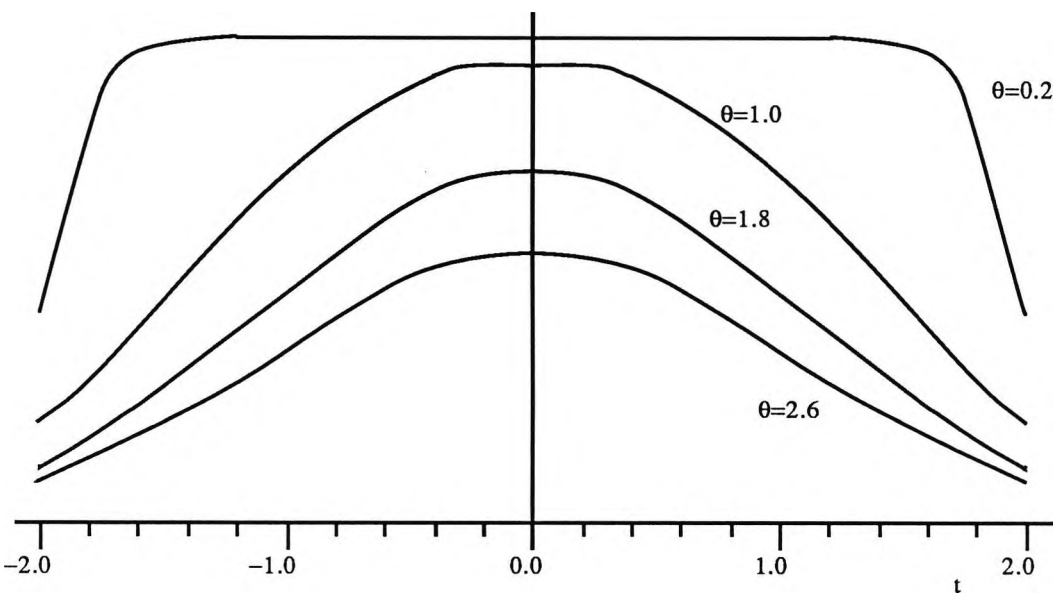


Figure 3.19 Probability That an Observed Value is Normal (Using data from Tango, 1981)

3.6.5 Summary

Laboratory data variables can be classified by comparison with typical values from a reference population. The standard method of classification, both in the clinical laboratory and in knowledge-based systems, has been to divide the range of reference values into three intervals and to classify data according to the interval in which they fall. This method involves a substantial loss of information and several alternative methods are available which preserve a greater proportion of the information content of the original data. These methods include the division of the reference values into more than three intervals, the application of fuzzy set theory and the definition of probability distributions for classification. In the latter case, a method has been presented for direct calculation of probability distributions, which exploits the difference between inter-individual and intra-individual variations of laboratory data variables.

3.7 Treatment of Uncertainty

3.7.1 Introduction

The ability to function under conditions of uncertainty is an important feature of knowledge-based systems (Hayes-Roth, 1984). Uncertainty can exist both in the representation of the knowledge contained in the system and in the data input to it, and these can introduce uncertainty into the system's output. Most of the first generation knowledge-based systems used *ad hoc* scoring schemes to handle uncertainty under the assumption that these were the best way to explicitly represent uncertainty in a knowledge base.

A general dissatisfaction with the performance of the early *ad hoc* methods led to a search for more formal methods of handling uncertainty. There was renewed interest in the use of Bayes' Theory, extended from its simplest application to operate in structured hypothesis spaces; the Dempster-Shafer Theory of evidence (Shafer, 1976) was recognized as having potentially useful applications in knowledge-based systems; the Theory of Possibility (Zadeh, 1978) which emerged from Fuzzy Set Theory, was proposed as an alternative to probability theory.

The following sections present MYCIN's certainty factor model as an example of an early *ad hoc* scoring scheme, then examine the simple application of Bayes' Theorem and its use in a hierarchically organized set of hypotheses. Finally the Dempster-Shafer theory and the Theory of Possibility are presented and the relative merits of the Probability, Possibility and Dempster-Shafer theories are discussed in the context of knowledge-based systems.

3.7.2 MYCIN's Certainty Factor Model

The development of MYCIN's certainty factor model was prompted by its designers' dissatisfaction with existing techniques for handling uncertainty (Shortliffe & Buchanan, 1975). It was intended to be similar to the existing theory of confirmation (Carnap, 1950), adapted for the practical purposes of a rule-based reasoning strategy. It is interesting to note that one of the main doubts expressed about subjectivist Bayesian updating was its reliance on clinicians' estimates of uncertainty measures - a feature that is preserved in the certainty factor model. Two basic measures are defined in the model:

MB(h|e) is a measure of the increase of belief in the hypothesis h when evidence e is observed

MD(h|e) is a measure of the increase of disbelief in the hypothesis h when evidence e is observed

The measures of increased belief and disbelief can be related to probabilities:

$$\text{If } P(h|e)>P(h), \qquad MB(h|e) = \frac{P(h|e)-P(h)}{1-P(h)} \text{ and } MD(h|e)=0 \tag{3.7.1}$$

$$0 \leq MB(h|e) \leq 1$$

$$\text{If } P(h|e)<P(h), \qquad MD(h|e) = \frac{P(h)-P(h|e)}{P(h)} \text{ and } MB(h|e)=0 \tag{3.7.2}$$

$$0 \leq MD(h|e) \leq 1$$

The certainty factor associated with hypothesis h is defined as:

$$CF(h|e) = MB(h|e)-MD(h|e) \tag{3.7.3}$$

$$-1 \leq CF(h|e) \leq 1$$

It can be seen from (3.7.1) and (3.7.2) that MB(h|e)=MD(~h|e) which implies that:

$$CF(h|e)+CF(\sim h|e)=0 \tag{3.7.4}$$

Combining functions for the measures of belief and disbelief were defined in accordance with certain criteria for the behaviour of the model (Shortliffe & Buchanan, 1975):

MB(h) due to accumulating confirming evidence increases towards unity but is only equal to unity when a single piece of evidence categorically implies the hypothesis

MD(h) due to accumulating disconfirming evidence increases towards unity but is only equal to unity when a single piece of evidence categorically implies the complement of the hypothesis

CF(h) due to all evidence is always greater than CF(h) due to disconfirming evidence only

CF(h) due to all evidence is always less than CF(h) due to confirming evidence only

CF(h) remains undefined in the case where MB(h) and MD(h) would both be unity

MB(h), MD(h) and CF(h) do not depend on the order in which evidence is accumulated or on evidence that is unknown

Using these criteria the following combining functions were devised:

For accumulated evidence e1, e2

$$\begin{aligned} MB(h|e1,e2) &= 0 && \text{if } MD(h|e1,e2)=1 \\ &= MB(h|e1)+MB(h|e2)(1-MB(h|e1)) && \text{otherwise} \end{aligned} \tag{3.7.5}$$

$$\begin{aligned} MD(h|e1,e2) &= 0 && \text{if } MB(h|e1,e2)=1 \\ &= MD(h|e1)+MD(h|e2)(1-MD(h|e1)) && \text{otherwise} \end{aligned} \tag{3.7.6}$$

which indicates that the measure of belief/disbelief associated with the new evidence e2 is applied to the belief/disbelief remaining uncommited to h after e1.

For a conjunction of hypotheses:

$$MB(h1,h2|e) = MIN[MB(h1|e);MB(h2|e)] \tag{3.7.7}$$

$$MD(h1,h2|e) = MAX[MD(h1|e);MD(h2|e)] \tag{3.7.8}$$

which indicates that the overall belief in a compound hypothesis is only as strong as the least believed element and that overall disbelief is as great as the most disbelieved element.

For a disjunction of hypotheses:

$$MB(h1 \text{ or } h2|e) = MAX[MB(h1|e);MB(h2|e)] \tag{3.7.9}$$

$$MD(h1 \text{ or } h2|e) = MIN[MB(h1|e);MD(h2|e)] \tag{3.7.10}$$

which indicates that the overall belief in a disjunction of hypotheses is as strong as the most believed element and that overall disbelief is only as great as the least disbelieved element.

3.7.3 Bayesian Methods

3.7.3.1 Introduction

The conditional probability of a proposition H, given certain evidence E, can be viewed as a measure of belief in the proposition (Cheeseman, 1985). This fact forms the basis of any probabilistic inference system in which the belief in a set of hypotheses H={h1,h2,h3,...hn} given a set of evidence E={e1,e2,e3,...em} is expressed as

$$Bel(hi)=P(hi|E)=P(hi|e1,e2,e3,...em) \tag{3.7.11}$$

With any reasonably large set E, it is not practical to measure P(hi|e1,e2,e3,...em), instead conditional probabilities of the type P(ei|hj) are measured and P(hj|E) is calculated using Bayes' Theorem (Bayes, 1763):

$$P(hj|E) = \frac{P(hj).P(E|hj)}{P(E)} \tag{3.7.12}$$

where P(hj) and P(E) represent the *a priori* probabilities of hj and E.

The assumption could be made that the pieces of evidence ei are statistically independent:

$$\text{Assuming } P(ei|ek)=P(ei) \tag{3.7.13}$$

$$\text{then } P(E) = P(e1).P(e2).P(e3)....P(em) \tag{3.7.14}$$

Another assumption could be made that the pieces of evidence ei are statistically independent in the presence of hj:

$$\text{Assuming } P(ei|ek,hj)=P(ei|hj) \tag{3.7.15}$$

$$\text{then } P(E|hj) = P(e1|hj).P(e2|hj).P(e3|hj)...P(em|hj) \tag{3.7.16}$$

Combining the assumptions leading to (3.7.14) and (3.7.16) with (3.7.12):

$$P(hj|E) = \frac{P(e1|hj).P(e2|hj).P(e3|hj)...P(em|hj).P(hj)}{P(e1).P(e2).P(e3)...P(em)} \tag{3.7.17}$$

which enables the belief in hypotheses hj to be updated on the evidence given the *a priori* probabilities P(hj), P(ei) and P(ei|hj).

The number of *a priori* probabilities required can be reduced if the set of hypotheses H is exhaustive (it would be necessary to include *healthy* as a member of H). In this case the evidence probabilities can be calculated as:

$$P(ei)=P(ei|h1).P(h1)+P(ei|h2).P(h2)+...+P(ei|hn).P(hn) \quad (3.7.18)$$

The *a priori* probabilities can be calculated from large samples of the population or they can be estimated by experts (leading to *subjective* Bayesian updating of belief). Probably the most successful of the many computer applications of Bayes' Theorem in its simplest form, has been developed and tested over a period of many years at the General Infirmary, Leeds for the diagnosis of abdominal pain (Horrocks *et al*, 1972; de Dombal *et al*, 1972). Although it employs only a very simple diagnostic algorithm, the system consistently out-performs even senior clinicians when the *a priori* probabilities used in the Bayesian updating are calculated from retrospective data (Leaper *et al*, 1972). A number of interesting points regarding the use of subjective estimates of probabilities emerged from the evaluation of the system. For commonly occurring diseases clinicians were out-performed by the system using their own probability estimates; for rarer diseases the clinicians out-performed the computer. In either case the system was more accurate when it used probability estimates pooled from several clinicians than when it used estimates from a single clinician. The conclusion drawn from this (Leaper, 1972) is that real data should be used to calculate the *a priori* probabilities whenever possible.

3.7.3.2 Bayesian Updating in A Hierarchical Hypothesis Space

A method of Bayesian updating in a hierarchically organized set of hypotheses has been suggested by Pearl (1986a). Consider a set of exhaustive and mutually exclusive hypotheses H={h1,h,2h3,...hn} and an arbitrary number of subsets of H that are of interest as hypotheses themselves. A hierarchy can be constructed in which H is the root node, the individual h1,h2,h3,...hn are leaf nodes and subsets of H are arranged as intermediate nodes which are the disjunction of their immediate descendent nodes. Such a hierarchy is shown in Figure 3.20.

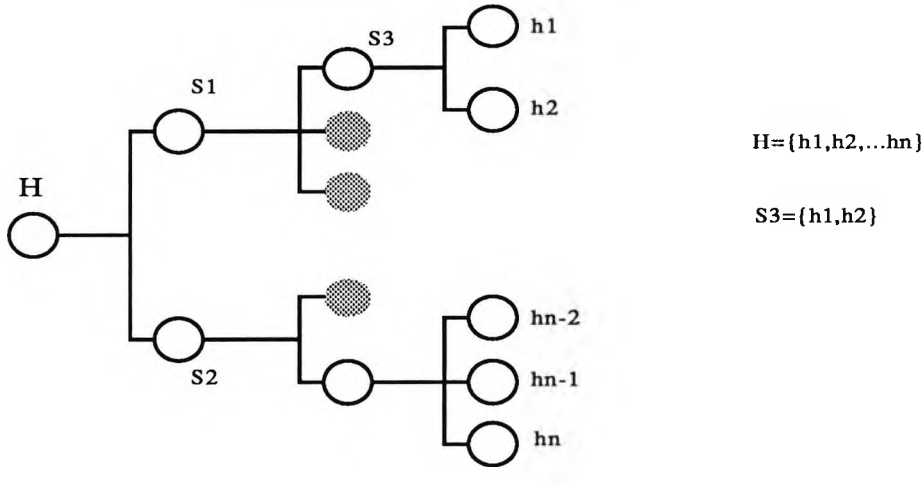


Figure 3.20 A Hierarchical Hypothesis Space

Suppose that a set of evidence E has resulted in a measure of belief in a leaf node hi and that this belief is expressed as in (3.7.11):

$$Bel(hi)=P(hi|E) \tag{3.7.19}$$

Now consider the impact of a new piece of evidence, e, which affects the belief in an intermediate node, S, without discriminating between its descendents. In other words:

$$P(e|S,R)=P(e|S) \quad \text{for } R\supseteq S \tag{3.7.20}$$

It is also assumed that e does not discriminate between hypotheses outside S:

$$P(e|\sim S,R)=P(e|\sim S) \quad \text{for } R\supseteq \sim S \tag{3.7.21}$$

Hence for the leaf nodes:

$$\begin{aligned} P(e|hi) &= P(e|S) && \text{for } hi\in S \\ &= P(e|\sim S) && \text{for } hi\in \sim S \end{aligned} \tag{3.7.22}$$

Now, Bayes' Theorem states that

$$P(hi|e) = \frac{P(e|hi).P(hi)}{\Sigma P(e|hj).P(hj)} \tag{3.7.23}$$

where the summation is over the n leaf nodes and P(hj) are the probabilities prior to the observation of e.

For the case $hi\in S$

$$P(hi|e) = \frac{P(e|S).P(hi)}{P(e|S).P(S) + P(e|\sim S).P(\sim S)} \tag{3.7.24}$$

so defining the likelihood ratio $\lambda s=P(e|S)/P(e|\sim S)$

$$P(hi|e) = \frac{\lambda s.P(hi)}{P(S) + \lambda s.P(\sim S)} \tag{3.7.25}$$

and since the hypothese in H are mutually exclusive and exhaustive, $P(\sim S)=1-P(S)$ and hence:

$$P(hi|e) = \frac{\lambda s.P(hi)}{\lambda s.P(S)+1-P(S)} \quad \text{for } hi\in S \tag{3.7.26}$$

A similar analysis for the case of $hi \in \sim S$ yields:

$$P(hi|e) = \frac{P(hi)}{\lambda s.P(S)+1-P(S)} \quad \text{for } hi \in \sim S \tag{3.7.27}$$

So defining a normalizing factor $\alpha s=[\lambda s.P(S)+1-P(S)]^{-1}$, (3.7.26) and (3.7.27) can be rewritten as

$$\begin{aligned} Bel'(hi) &= \alpha s.\lambda s.Bel(hi) \quad \text{for } hi \in S \\ &= \alpha s.Bel(hi) \quad \text{for } hi \in \sim S \end{aligned} \tag{3.7.28}$$

where Bel'(hi) is the updated belief in hi on obseravtion of e.

Since the intermediate node S is the disjunction of its mutually exclusive leaf node descendents, its measure of belief can be found by:

$$Bel(S) = \Sigma_i Bel(hi) \quad \text{for } hi \in S \tag{3.7.29}$$

The knowledge-base for a system that applied this scheme would contain values for the likelihood ratio λsi; on observation of ei, belief in S is increased if $\lambda si>1$, decreased if $\lambda si<1$ and remains unchanged if $\lambda si=1$.

Pearl (1986a) also shows that the updating of belief can be achieved by propagating messages between nodes in the hierarchy; indeed the propagation of belief in a hierarchy of hypotheses is a special case of propagation in belief networks (Kim & Pearl, 1983; Pearl, 1986b).

3.7.4 The Dempster-Shafer Theory

The Dempster-Shafer theory is the result of work by Arthur Dempster (Dempster, 1967) and Glenn Shafer (Shafer, 1976) on the mathematics of evidence. A set of mutually exclusive and exhaustive hypotheses H={h1,h2,...hn} is defined as the *frame of discernment,* and consideration is given to belief in propositions formed as subsets of H. For a frame of discernment with n elements there are 2^n possible subsets; Figure 3.21 shows the situation for H={h1,h2,h3,h4}.

The direct effect of a piece of evidence on a proposition A, which is one of the 2^n subsets of H, is measured by making a *basic probability assignment* for A in the range 0 to 1; this is denoted m(A). The basic probability assignments are made throughout H such that:

$$\Sigma_H m(A) = 1 \tag{3.7.30}$$

where the summation is over the 2^n subsets of H

$$\text{and } m(\varnothing) = 0 \tag{3.7.31}$$

The total belief in the proposition A is defined by a belief function Bel(A) which is the sum of the basic probability assignments for all propositions which are subsets of A:

$$Bel(A) = \Sigma_A m(X) \tag{3.7.32}$$

where the summation is over the subsets of A.

The belief in a proposition when several pieces of evidence are present is found by defining a combination rule for the basic probability assignments. Suppose evidence e1 assigns m1(S) to each proposition in H and e2 assigns m2(S), then the combined probability assignment for a node A, due to e1 and e2, is given by:

$$m12(A) = \Sigma m1(X).m2(Y) \tag{3.7.33}$$

So the combined probability assignment is found by summing the products of the basic probability assignments for each pair of sets X,Y in H whose intersection is A.

Considering the summation in (3.7.33) over the entire frame of discernment H:

$$\begin{aligned} \Sigma_H m1(X).m2(Y) &= \Sigma_H m1(X).\Sigma_H m2(Y) \\ &= 1 \end{aligned} \tag{3.7.34}$$

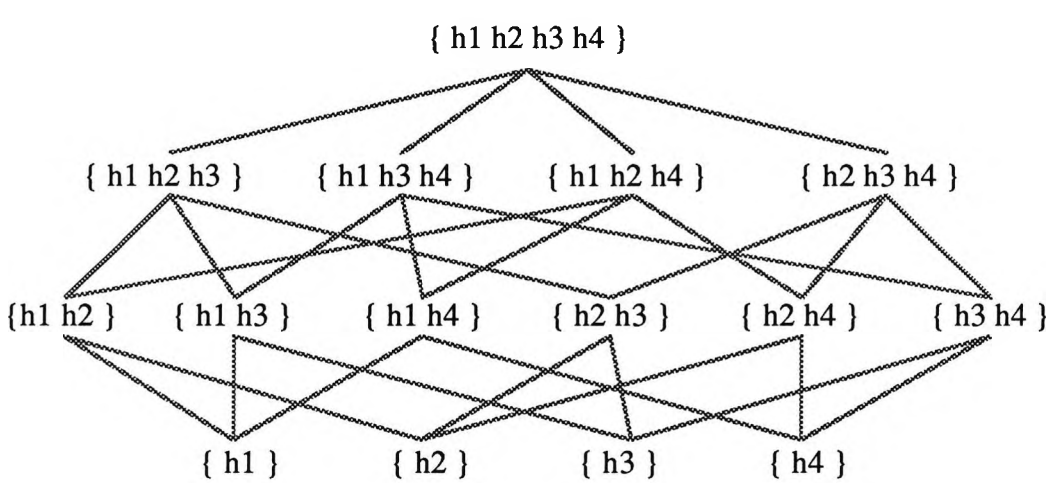


Figure 3.21 A Hierarchy of Subsets

Hence the combined probability assignment satisfies the conditions in (3.7.30). However, some intersections $X \cap Y$ will yield the empty set $\varnothing$ and in general $m12(\varnothing) \neq 0$, so the condition in (3.7.31) is violated. This problem is circumvented by normalizing the combined probability assignments so that:

$$m'12(\varnothing) = 0$$

and

$$m'12(A) = \frac{m12(A)}{1-m12(\varnothing)} \qquad (3.7.35)$$

which ensures that m'12(A) satisfies both (3.7.30) and (3.7.31).

There is no mathematical basis for the validity of this normalization process.

Unlike Probability Theory, the Dempster-Shafer theory does not necessarily assign to the complement of A (A') the belief that remains unassigned to A: instead this belief can be assigned to H itself. So for any proposition A:

$$Bel(A)+Bel(A') \leq 1 \qquad (3.7.36)$$

(the equlity holds if $m(\varnothing)=0$, which is the case in a probabilistic updating scheme)

This enables a plausibility function to be defined as:

$$Pl(A) = 1-Bel(A') \qquad (3.7.37)$$

where $Pl(A) \leq Bel(A)$

3.7.5 Possibility Theory

The fuzzy set theory presented in Section 3.6.3 as a method for data classification forms the basis of a method for making inferences in the presence of uncertainty using fuzzy composition operators (Zadeh, 1973; 1979). An example of the use of these operators will be presented in the context of the knowledge-based system CADIAG-2 (Adlassnig, 1985) which was described in Section 2.3.4. Suppose that a data variable U has been observed with a value u and has been classified as A_1, A_2, A_3 (*eg* low, normal, high) by one of the sets of membership functions defined in Section 3.6.3. so that the degree of membership of u in A_i is a_i. Now suppose that there exist rules of the form:

IF A_i
THEN D_j with o_{ij}, c_{ij}

where o_{ij} is the frequency of occurrence and c_{ij} the strength of confirmation (see Section 2.3.4).

Three fuzzy compositions can be performed in order to find the extent to which D_j is confirmed (or excluded) by the observation of U.

confirmation: $m_{jc} = MAX_i\ MIN[a_i;c_{ij}]$

positive exclusion: $m_{jpe} = MAX_i\ MIN[a_i;1-c_{ij}]$

negative exclusion: $m_{jne} = MAX_i\ MIN[1-a_i;o_{ij}]$

Confirmation (m_{jc}) is high if a symptom that is probably present is a strong indicator for the disease. Positive exclusion (m_{jpe}) is high if a symptom that is probably present is a strong contra-indication for the disease. Negative exclusion (m_{jne}) is high if a symptom that probably is not present occurs frequently with the disease.

A fuzzy membership function is defined for the proposition that the observed patient is a member of the set of patients having disease D_j. If either of the exclusion rules yields a definite exclusion (*ie* m_{jpe} or m_{jne} =1) then the diagnosis is excluded; otherwise the confirmation measure mjc is reported for the disease.

3.7.6 Discussion

When used for handling uncertainty in knowledge-based systems, all the schemes described above suffer a common problem: the certainty measures on which they are based (probabilities, likelihood ratios, certainty factors, possibilities etc) must somehow be determined. Only probability theory allows the direct measurement of these parameters, but this usually involves the collection of a prohibitively large amount of data.

The estimation of subjective certainty measures is beset with problems. Although clinicians weigh evidence as they make a diagnosis, they do not reason explicitly with numerical weights. Thus the estimated certainty measures are not based on heuristics used during diagnosis, instead they are based on heuristics invoked by the knowledge elicitation process itself. The actual numerical values of the certainty measures will therefore depend to some extent on the way in which they are elicited (Fox *et al*, 1983).

To ease the problem of estimation, some systems have associated linguistic descriptions with the certainty measures (*eg* INTERNIST used the terms *rarely, a substantial minority, roughly half, a substantial majority* and *essentially all* as descriptions of the frequency of symptoms with a disease). However, since these descriptions are converted to numerical weights within the system, they merely introduce a further source of error into the estimation; there is no formal basis for the mapping of linguistic descriptions into certainty measures, and the clinician's own cognitive model of the difference between, say, a

substantial minority and a substantial majority, is inevitably different from the relationship between the corresponding certainty measures within the system. This problem is even greater for systems using fuzzy variables - the availability of numerous linguistic descriptors, such as *never, seldom* or *often*, and the facility to calculate the impact of modifiers, such as *very* or *quite*, encourages their use without a full realization of their significance within the system.

Subjective probabilities can be elicited by asking such questions as *What proportion of cases of disease X exhibit manifestation Y?* and there is no distortion involved in the mapping of the clinician's response to the weights in the system. In summary, a clinician is more likely to give an accurate probability than any other weight of evidence (Welbank, 1983).

Probabilistic updating schemes have been criticized for the independence assumptions they make - the independence of evidence (3.7.13) and the independence of evidence given a disease (3.7.15). The most notable argument has concerned the Bayesian updating scheme in PROSPECTOR (Duda *et al*, 1979) with the claim that no updating is possible under the assumption of independence (Pednault *et al*, 1981) being refuted by several authors (Glymour, 1985; Johnson, 1986).

The two independence assumptions have been defended by Charniak (1983). He observes that the assumption of independence of evidence is made to enable the calculation of the *a priori* probability of the evidence set E in (3.7.14) and that this appears in the denominator of each updating factor. Thus errors introduced by the assumption of independent evidence affect each hypothesis to the same degree. The problem of assuming independence of evidence in a given disease state can be avoided by grouping together all dependent pieces of evidence and defining them as a single piece of evidence for the disease. This amounts to the definition of intermediate disorders based on small sets of evidence.

The Dempster-Shafer theory makes the same independence assumptions as probabilistic updating schemes. Another problem is that it is not feasible to implement the theory due to the exponential increase in computation time with the number of hypotheses (Barnett, 1981). For instance, with 20 hypotheses, 2^{20} subsets (about 1,000,000) would be involved in the calculations. Several methods for reducing the computation time have been suggested. An approximation to the Dempster-Shafer theory can be made by considering only those subsets of the hypotheses that have some semantic significance (*ie* the subsets are themselves hypotheses that may interest a clinician) and evidence that supports these hypotheses or their complements. The subsets can be organized into a hypothesis hierarchy (which is the same hypothesis space as the Bayesian scheme in Section 3.7.3.2) and the evidence combination rule in (3.7.33) is approximated by a scheme in which the number of

computations increases linearly with the number of hypotheses (Gordon & Shortliffe, 1985). Another scheme has been proposed for a full implementation of the Dempster-Shafer theory in a hierarchical hypothesis space using a method of propagation between the nodes (Shafer & Logan, 1987). Although it avoids the approximation used by Gordon & Shortliffe, it involves a greater number of computations (computation time again increases linearly with the number of hypotheses).

The combination of evidence in Possibility theory has been criticized for its reliance on MIN and MAX functions (Cheeseman, 1986). The use of the MIN function can be viewed as an assumption of complete dependence of evidence even when this is obviously not the case.

3.8 Explanation and Query Handling

3.8.1 Introduction

In a study of clinicians' attitudes towards the use of computer-aided medical consultation systems (Teach & Shortliffe, 1981) it was found that the most important attribute of such systems, in the opinion of potential users, is their ability to explain their diagnostic reasoning and therapy selection. Generally, the presentation of an explanation follows some specific query from the user and the way in which the computer handles this user-driven dialogue can be viewed as the three stage process shown in Figure 3.22. The first stage involves understanding (identifying) the query, the second is the retrieval of the information necessary to answer it and the final stage is the generation of the textual output. The following sections analyze the types of query handling that are desirable in a knowledge-based system, describe how queries can be understood using the PROLOG programming language, and review techniques for generating textual output.

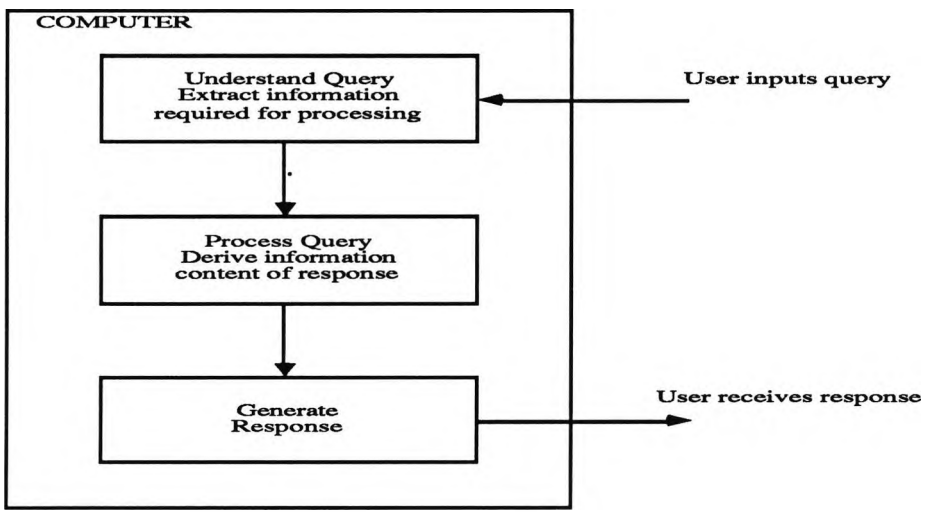


Figure 3.22 Handling a Dialogue Bewteen Computer and Clinician

3.8.2 The Types of Query Handled by Knowledge-Based Systems

Part of the process of understanding a natural language query involves the determination of the type of the query. Queries can be categorized using syntactic criteria applied to the question itself (*eg* who, what, why, how queries) or applied to the response expected (*eg* yes/no). A more useful categorization, for the purposes of knowledge-based systems, is made according to the conceptual type of the query (Lehnert, 1978). A list of 13 conceptual categories of query is shown in Table 3.1.

All queries in the same conceptual category are processed in the same way, hence in a knowledge-based system, once the conceptual category has been recognized, the appropriate query handling routine can be applied in order to generate a response. In terms of the knowledge representation required to process each category, the first two in Table 3.1 - causal consequent and causal antecedent - are handled most satisfactorily by a deep knowledge representation such as the semantic networks described in Section 3.4. The remaining categories can be handled adequately by surface-level representations, although in some instances (*eg* goal orientation, enablement) the quality of the response is improved by deeper representation.

A good example of the way in which deep causal knowledge can enhance the quality of an explanation is given by the XPLAIN system (Swartout, 1983). If the system has asked for the value of serum calcium, the user may respond by asking the question *Why?* (*ie* Why do you need to know serum calcium? - a goal orientation query). The system with a surface-level (rule-based) knowledge representation might respond by stating that its top goal is to select a therapy and that a sub-goal is to check the sensitivity of various therapies to different variables, including serum calcium.

Conceptual Category	Example
Causal Consequent	What is the result of the fall in pH in a respiratory acidosis?
Causal Antecedent	Why does pH fall in respiratory acidosis?
Goal Orientation	Why did you diagnose respiratory acidosis?
Enablement	What did you need to know in order to make the diagnosis?
Verification	Did you measure the plasma sodium?
Disjunctive	Is the pH low or normal?
Instrumental/Procedural	How did you diagnose respiratory acidosis?
Concept Completion	Who measured the blood gases?
Expectational	Why didn't you diagnose respiratory alkalosis?
Judgemental	What if pH had been high?
Quantification	What is the value of pH?
Feature Specification	What are the symptoms of respiratory acidosis?
Request	Would you diagnose this patient please?

Table 3.1 Conceptual Categories of Questions (Adapted from Lehnert, 1978)

The same query posed to the XPLAIN system, with a deeper knowledge of physiological causal relationships, might invoke the response:

The system is anticipating digitalis toxicity. Increased serum calcium causes increased automaticity, which may cause a change to ventricular fibrillation. Increased digitalis also causes increased automaticity. Thus, if the system observes increased serum calcium, it reduces the dose of digitalis due to increased serum calcium

(Swartout & Smoliar, 1987)

The EMYCIN knowledge-based system shell handles five types of query concerning a consultation, as well as general queries (mainly information requests) about the knowledge base (van Melle *et al*, 1981). Table 3.2 lists the five types of query in EMYCIN with some examples - it should be noted that some of these types subsume several of the conceptual categories in Table 3.1.

MYCIN Category/Examples	Conceptual Category
What is <parameter> of <object>	
What is the genus of organism-1?	Quantification
Is Organism-1 Corynebacterium-non-diphtheriae?	Verification
How do you know the value of <parameter> of <object>	
Did you consider bacteroides as a possibility for organism-1?	Verification
Why don't you think that the site of culture-1 is urine?	Expectational
Why did you rule out streptococcus as a possibility for organism-1?	Goal Orientation
How did you use <parameter> of <object>	
Did you consider the fact that patient-1 is a compromised host?	Verification
How did you use the aerobicity of organism-1?	Instrumental/Procedural
Why didn't you find out about <parameter> of <object>	
Did you find out about the patient's CBC?	Verification
Why didn't you need to know whether organism-1 is a contminant?	Expectational
What did <rule> tell you about <object>	
How was Rule 178 helpful when you were considering organism-1?	Instrumental/Procedural
Did Rule 116 tell you anything about infection-1?	Verification
Why didn't you use Rule 189 for organism-2?	Expectational
General Examination of Knowledge Base	
Is blood a sterile site?	Verification
What are the non-sterile sites?	Feature specification
What organisms are likely to be found in the throat?	Feature specification
Is Bacteroides aerobic?	Verification
How do you decide that an organism might be Streptococcus?	Instrumental/Procedural
Do you use Gram Stain to determine genus?	Verification
What drugs would you consider to treat E. Coli?	Feature specification

Table 3.2 Query Types in EMYCIN

3.8.3 Understanding Queries

3.8.3.1 Introduction

When a string of characters is input as a query to a knowledge-based system, it must be analyzed in order to extract the information necessary to generate a response. This analysis should identify the conceptual category of the question and the key data required by the query processor for that category. The process of analyzing a character string in this way is called *parsing*.

Several strategies have been developed for the parsing of natural language input to a computer system. The earliest language understanding systems, such as ELIZA (Weizenbaum, 1966), used the technique of *template matching* in which the input character string is compared with a series of templates until a match is found (with variables in the template being instantiated by data in the input string). Another important class of parsers are based on representations of the grammar of the language.

The basic building blocks of a language are words (*terminal symbols*) and the set of allowable words is the *lexicon* for the language. Words can be used to create compound structures (*non-terminal symbols*) such as phrases and sentences. The grammar is a set of rules (*rewrite rules*) describing relationships between terminal and/or non-terminal symbols that can be used to generate every allowable character string for the language. There are a number of different types of grammar, distinguished by the form that their rewrite rules may assume. Of particular interest to computer-based understanding systems is the *context free grammar* in which each rewrite rule describes a single non-terminal symbol in terms of a set of terminals and non-terminals; it is context free because parsing is not affected by relationships between terminal symbols. A context free grammar can generate a parse tree such as the one shown in Figure 3.23.

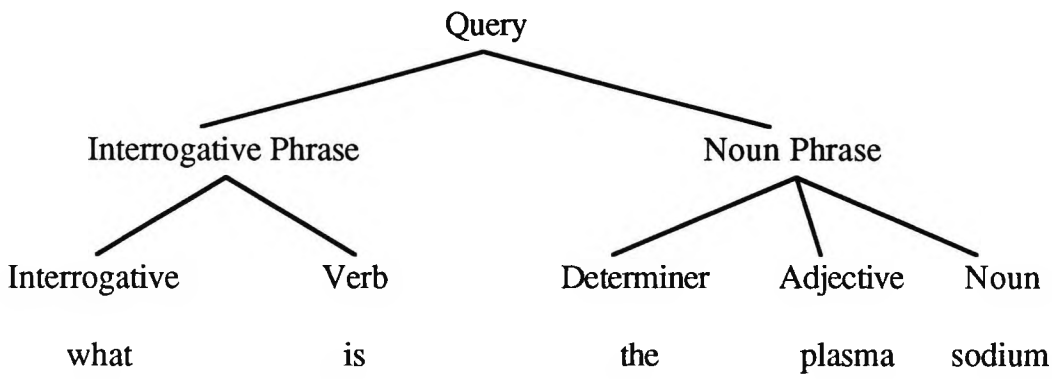


Figure 3.23 A Parse Tree

```
query --> interrogative_phrase,noun_phrase.

interrogative_phrase --> interrogative,verb.

noun_phrase --> determiner,adjective,noun.

interrogative(what) --> [what].

verb(is) --> [is].

determiner(the) --> [the].

adjective(plasma) --> [plasma].

noun(sodium) --> [sodium].
```

Figure 3.24 PROLOG Grammar Rules

Grammar-based parsers can operate *top-down*, starting with rules describing sentences, decomposing these into phrases and then into terminal symbols that are matched with the input character string. They can also operate *bottom-up* starting with words identified in the character string, organizing them into phrases and then into sentences. One method of implementing a grammar-based parser on a computer is to use the *augmented transition network* formalism (Woods, 1970). Another method which uses the programming language PROLOG is discussed in the next sub-section.

3.8.3.2 Understanding Queries with PROLOG

The computer language PROLOG (see Appendix VIII) incorporates a special notation for expressing context free grammars so that they can be used for efficient top-down parsing of natural language input. Figure 3.24 shows the PROLOG grammar rules that could be used to parse the query in Figure 3.23. The special symbol '-->' is read 'can take the form' so that the grammar rule for *query* is: *query can take the form interrogative_phrase and noun_phrase*. Grammar rules are translated by the PROLOG interpreter into ordinary PROLOG clauses - clauses for the grammar rules in Figure 3.24 are shown in Figure 3.25.

```
query(X1,X2):-interrogative_phrase(X1,Z),noun_phrase(Z,X2).

interrogative_phrase(X1,X2):-interrogative(X1,Z),verb(Z,X2).

noun_phrase(X1,X2):-determiner(X1,Z),adjective(Z,Y),noun(Y,X2).

interrogative([what|Z],Z).

verb([is|Z],Z).

adjective([plasma|Z],Z).

determiner([the|Z],Z).

noun([sodium|Z],Z).
```

Figure 3.25 PROLOG Clauses Translated From Grammar Rules

The grammar rules illustrated so far will only parse a list of words and identify it as a query - they do not extract any information from the query. This can be achieved by adding extra arguments to the grammar rules in Figure 3.24 which extend the context free grammar into a *definite clause grammar* (Pereira & Warren, 1980). The extended grammar rules that parse the query, extracting its type (information_request) and the data necessary to process it (in this case the name of a data variable) are shown in Figure 3.26.

```
query(information_request,Data_variable) -->
                        interrogative_phrase(request),noun_phrase(Data_variable).

 interrogative_phrase(request) --> interrogative(what),verb(be,Tense).

noun_phrase(Noun) --> determiner,adjective,noun(Noun).

interrogative(what) --> [what].

verb(be,present) --> [is].

determiner(the) --> [the].

adjective(plasma) --> [plasma].

noun(sodium) --> [sodium].
```

Figure 3.26 Definite Clause Grammar Extension

3.8.4 Generating Textual Output

There are two basic methods that can be used to generate textual output for a knowledge-based system. These can be described as the *grammar-oriented* and *goal-oriented* methods (Patten, 1988). Grammar oriented methods generate sentences by application of the rewrite rules of the grammar. In its simplest form this would generate every allowable sentence, checking each one to see if it conveyed the desired meaning. Obviously this would involve generating many sentences unnecessarily and the computational load can be reduced by invoking higher level knowledege at the same time as the grammar rules in order to limit the number of applicable rules.

Goal-oriented (or rule-based) methods create a plan of goals that must be persued in order to generate the desired textual output. As each goal is achieved, textual fragments are output. This approach is taken by PROSENET (Miller & Rennels, 1988) which generates prose for the ATTENDING system discussed in Section 2.3.3. Somewhat confusingly, PROSENET is based on an augmented transition network which would normally be used to store grammar rules (in fact PROSENET is not a grammar-oriented system). The network is traversed, with tests on each arc relating to the conclusions of the medical

consultation system; if the test on an arc evaluates successfully, the arc is traversed and an associated fragment of prose is output.

3.8.5 Summary

A knowledge-based system produces explanation of its conclusions in response to specific queries from the user. Understanding a query involves the identification of its conceptual category and the extraction of data which can be passed to a processing routine for that query category. The PROLOG programming language has the facility to understand queries in this way, using a definite clause grammar. Once the necessary information has been retrieved to generate the response to a query, textual output can be produced using a grammar-oriented or goal-oriented method.

3.9. Knowledge Acquisition

3.9.1. Introduction

The process of developing a knowledge-based system involves the close co-operation of one or more domain experts and the knowledge engineer(s) responsible for the computer implementation. Selection of suitable experts for the chosen domain is a crucial aspect of the development process and factors such as the degree of experience, ability to communicate reasoning processes and knowledge, availability and enthusiasm for the project should be taken into account when chosing a domain expert (Prerau, 1987).

It has been widely recognized (*eg* Grover, 1983; Welbank, 1983; Summers, 1988) that knowledge-based systems evolve as the result of a cyclic development process such as shown in Figure 3.27. Often the rigid segregation of the different phases will not be apparent in a project (Welbank, 1983) and there may indeed be instances where it is not desirable to progress from one phase to the next in a formal manner. The basic problem definition may be the result of a particular requirement identified by workers in the application domain of the system; in the medical domain, the problem definition has often been motivated as much by the research interests of the knowledge engineers as by the needs of the medical profession.

Some projects are blessed with the asset of researchers expert in both the application domain and in the field of artificial intelligence. In other cases, there must be an initial period in which knowledge engineers gain an insight into the application domain and the domain experts become acquainted with the way in which their knowledge is to be modelled. Once this first stage is completed, knowledge can be acquired to form the basis of a first prototype system. A cyclic process of knowledge base refinement then takes place as the prototype is debugged and feedback is obtained from its performance on test cases.

The knowledge acquired during this phase of development can come from a number of sources: it can be extracted from textbooks, deduced from the analysis of case data or elicited from the domain experts.

The eventual success or failure of a knowledge-based system depends to a large extent on the outcome of the knowledge acquisition process and this in turn depends on the psychology of the interaction between domain experts and knowledge engineers. Any individual whose expertise is worth capturing in a knowledge-based system is likely to lead a very busy professional life, and hence the process of eliciting knowledge must be made as efficient as possible, in order to limit demands on the expert's time.

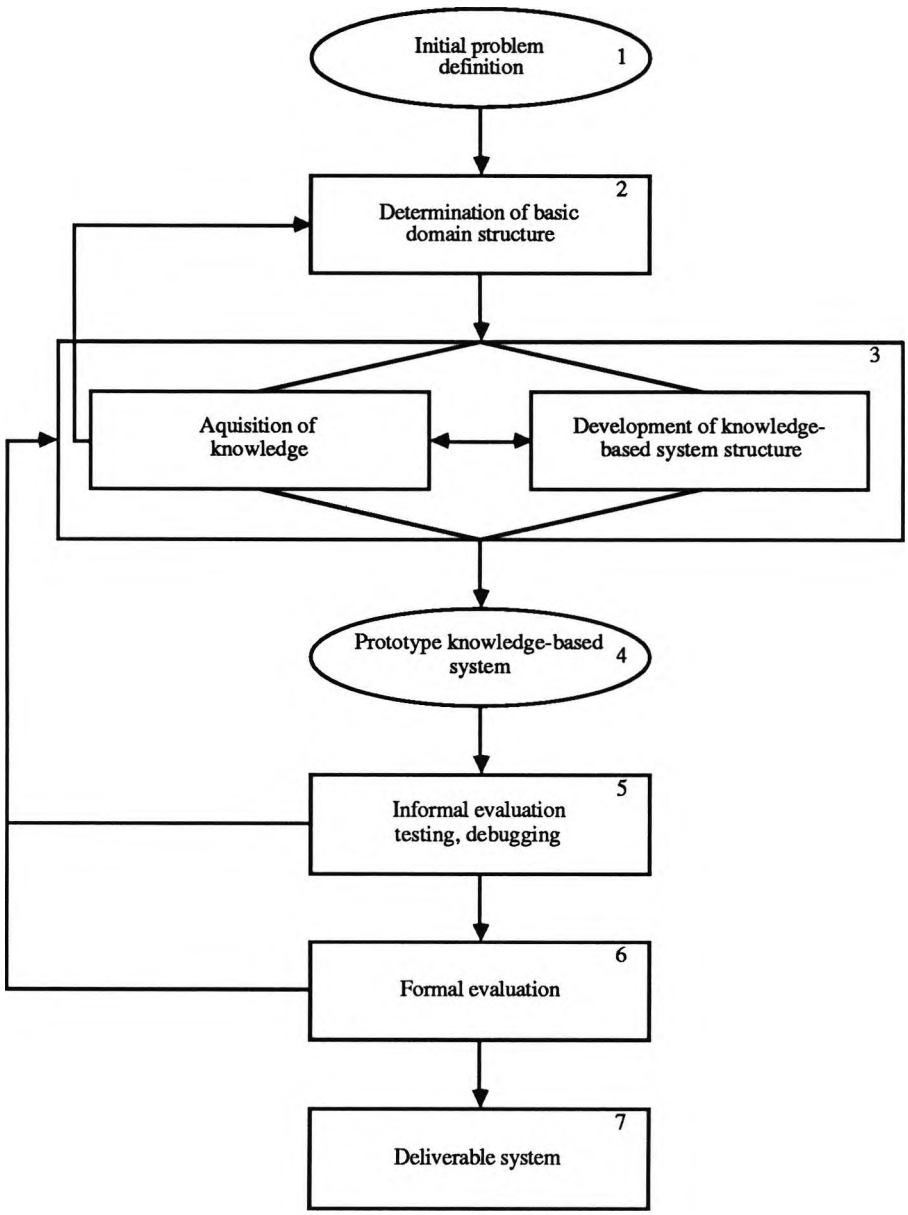


Figure 3.27 Knowledge-Based System Development Cycle

The technique used for knowledge acquisition must take account of the fact that there are different types of knowledge. *Declarative* domain knowledge is knowledge of facts - objects or concepts - and the way in which they are related. *Procedural* knowledge is knowledge of the actions needed to solve domain problems, the sequence of goals to be persued and how they should be achieved. A distinction can be made between these two types of knowledge, which are properties of the domain, and knowledge about the domain experts themselves. An expert's *strategic* knowledge includes knowledge about how to decide between alternative actions, how to handle uncertainty or what to do when data is missing or incomplete - special techniques may be required to elicit such knowledge (Gruber, 1988). Strategic knowledge can be used to determine the way in which overall control of a knowledge-based system is achieved; it may not be possible or desirable to model a human's strategy in a computer system. A detailed modelling of the cognitive processes by which an expert makes inferences is not normally attempted unless the system is designed specifically to study those processes.

The next two sections describe methods that can be used to elicit knowledge from experts in a form that can be utilized in a knowledge-based system and review some of the computer aids developed for use during the knowledge acquisition cycle.

3.9.2. Knowledge Elicitation Techniques

3.9.2.1 Introduction

The term *knowledge acquisition* refers to the complete process of acquiring knowledge, by whatever means, for incorporation in a knowledge-based system. *Knowledge elicitation* is concerned only with those knowledge acquisition procedures in which knowledge is obtained directly from human experts, through an interaction with knowledge engineers. Two basic strategies exist for knowledge elicitation (Welbank, 1983): interview of experts by knowledge engineers and observation of experts at work. There are a number of variations of the basic strategies, which are discussed in the next two sections.

Whichever knowledge elicitation method is used, the first representation of knowledge usually takes the form of a transcript of the knowledge elicitation session, either as a tape recording or as handwritten notes. The task of the knowledge engineer is then to extract the useful information from the transcript and incorporate it in a knowledge base. Several researchers have advocated the use of an intermediate knowledge representation whereby the knowledge extracted from the transcripts is expressed in a textual form of the representation method to be used in the computer system (rules, frames etc) so that it can be

easily reviewed by the expert before being incorporated into the computer knowledge base (Grover, 1983; Prerau, 1987).

3.9.2.2 Interview Strategies

An interview between knowledge engineers and domain experts is the most commonly used knowledge elicitation technique (Welbank, 1983; Shadbolt & Burton, 1989). Completely unstructured, informal interviews can be used, but generally a structured interview yields faster results. One structure proposed by Shadbolt & Burton is:

(1) Ask the expert for a brief outline of the task to be modelled by the knowledge-based system. Specifically ask for:

a description of possible problem solutions

descriptions of variables that affect the choice of solution

a list of the main rules connecting variables and solutions

(2) Consider each rule elicited in (1) and ask for the circumstances under which it is applicable. This should have the side effect of eliciting further rules from the expert.

(3) Repeat (2) until no further information is gathered.

Obviously this technique, which has been described as *distinguishing the goals* (Welbank, 1983) presumes that the general task is one of diagnostic classification and that knowledge is to be represented as rules; similar questions could be devised for other tasks or representations. Examples of the type of question that can be asked in stage (2) are given in Table 3.3.

Question	Purpose
Why would you do that?	Converts assertions to rules
How would you do that?	Generates lower order rules
When would you do that? Is <the rule> always the case?	Reveals rule applicability and may generate further rules
What alternatives to <this action/decision> are there?	Generates more rules
What if it were not the case that <currently true condition>?	Generates more rules
Can you tell me more about <any subject already mentioned>?	Generates further dialogue

Table 3.3 Questions for Knowledge Acquisition

Grover (1983) describes a similar method of *reclassification* used to elicite knowledge in the form of frames. Starting from observable objects in the domain, rules are sought from the expert to reclassify them into higher level abstract objects, these are then reclassified into yet higher levels. The abstract objects can be represented as frames in the knowledge-based system and the reclassification rules used to fill slots in those frames. A practical method of achieving the reclassification is to perform a *card sort* (Shadbolt & Burton, 1989) in which the expert repeatedly sorts cards, labelled with the domain objects, into different groupings.

Grover also describes the method of *forward scenario simulation* in which the expert choses a typical problem situation in the domain and describes the way in which he would solve that problem. This is similar to the *critical incident* technique (Flanagan, 1954) in which subjects describe their experiences in particularly memorable past incidents; both techniques resemble the observational methods of knowledge elicitation described in the next section.

3.9.2.3 Observational Strategies

Observational strategies for knowledge elicitation involve the knowledge engineer making observations of a domain expert solving problems. These can be real-life problems (*ie* the expert is observed *on the job*) or hypothetical problems of various levels of contrivance.

The process of eliciting knowledge by observing an expert at work on a real-life problem is called *protocol analysis* (Grover, 1983; Welbank, 1983; Shadbolt & Burton, 1989). The expert can be asked to make a commentary as he solves a problem and the resulting transcript can then be analyzed to extract knowledge. Alternatively, the expert can make a retrospective commentary on his problem solving behaviour whilst viewing a video of himself at work (Elstein *et al*, 1978) - this approach can be extended to a commentary by a panel of experts, which need not necessarily include the original expert.

Protocol analysis can also be performed on test cases presented to the expert in the form of a written summary. This has the advantage over observation in a real-life situation, in that the precise nature of the cases analyzed can be carefully controlled. Such a technique has been reported as part of the knowledge acquisition cycle of a system for the diagnosis of leukaemia (Fox *et al*, 1985) which was constructed using EMYCIN. Tape recordings were made of an expert diagnosing 63 documented cases of leukaemia, in the presence of two knowledge engineers who were able to ask clarifying questions where necessary. The recordings were analyzed in three stages: passages in the session transcript were isolated where they appeared to contain, in the opinion of the knowledge engineers, *substantative*

information; the content of each passage was then reduced to its underlying statement of knowledge and any duplications were eliminated; the knowledge statements were then translated into EMYCIN's IF-THEN rule format. It was found that the protocol analysis was not useful for organizing the basic structure of the domain (the CONTEXT TREE in EMYCIN - see Section 2.2.2). Instead, the basic structure was determined by an informal interview with the expert followed by a process of trial and error. It was also noticed that the knowledge obtained was mainly of a qualitative nature - precise quantitative data, particularly estimates of certainty factors, had to be obtained by separate discussion with the expert.

Protocol analysis has also been used to elicit knowledge in order to reconstruct the *cognitive causal model* of anatomy and physiology used by expert clinicians (Kuipers & Kassirer, 1984). It was observed that highly experienced clinicians have compiled knowledge into heuristic rules that may mask the underlying model being used and that it may be better to study less experienced clinicains. A single clinical case, a patient with a kidney disorder, was used as the basis for the knowledge elicitation session, in which the clinician was asked to *think aloud* as he performed a diagnosis. After this procedure had been completed, the clinician was *cross examined* by the knowledge engineers in order to clarify particular points.

The session transcript was reduced to a list of informative passages in much the same way as described above. Kuipers & Kassirer had noted that a subject is in no better position to explain his own mental processes than an outside observer (Ericsson & Simon, 1980) and hence they ignored any passages in the transcript in which the clinician attempted to describe his actual process of thinking as opposed to describing processes in the domain. The extracted statements were first analyzed to determine the basic objects - *substances, locations, forces, flows* and *concentrations* - in the clinician's model. The causal relationships between the objects were then identified. It should be noted that the knowledge engineers in this project possessed considerable knowledge of the physiological models they were eliciting and that the final computer model relied heavily on their own knowledge and knowledge from medical textbooks. This would seem to bear out the conclusion that protocol analysis *...does not give a complete and accurate picture of the expert's knowledge* (Fox *et al*, 1985).

Other observational knowledge elicitation techniques are more contrived than basic protocol analysis, forcing the expert to work in unusual ways in order to illuminate his knowledge of the domain. One such method resembles a game of *twenty questions* and is reported by Welbank (1983) for eliciting the knowledge of computer programmers. The programmers were asked to reformat a data file but were not given any details about the contents of the

file; they were encouraged to discover these by questioning the knowledge engineers. It was hoped that the questions they asked would reveal their knowledge; it was found that the technique could be used to elicit high level goals and strategies but not detailed procedural knowledge.

An evaluation of three knowledge elicitation techniques - structured interview, twenty questions and card sort - has been performed in the domain of industrial inspection lighting (Schweickert *et al*, 1987). Two knowledge engineers were able to elicit 61 rules using structured interview, 50 using twenty questions and only 10 with a card sort. Of the elicited rules, a higher percentage were validated by the expert for interview and twenty questions than for card sort. The twenty questions method was found to be time-consuming to prepare and compared with the other techniques, a lower percentage of the rules it elicited were suitable for implementation in a knowledge-based system.

3.9.3. Computer Aids in Knowledge Acquisition

3.9.3.1 Introduction

In addition to the strategies for knowledge elicitation described above, three further methods of knowledge acquisition can be identified, which involve the use of computer aids (Fox *et al*, 1985). These methods are the induction of rules or other structured knowledge following analysis of statistical or case data in the domain (Michalski & Chilausky, 1980; Blum, 1982; Weiss *et al*, 1986), similar induction following analysis of the knowledge base itself (Davis, 1979) and the use of interactive computer-based tools as an aid to knowledge acquisition. The first two methods are forms of machine learning which are not yet reliable enough to be considered as practical options for routine knowledge acquisition - for this reason they are not discussed here.

An important advantage associated with the use of knowledge acquisition tools is the facilitation of knowledge verification (Mars & Miller, 1987). Knowledge verification ensures that the knowledge base is *consistent, acccurate* and *complete* - a process which can take a great deal of time when performed as part of the system's evaluation. Three types of knowledge acquisition tools are discussed in the following sections. Knowledge base browsers or editors are the simplest implementation of a knowledge acquisition tool; they can be used interactively for knowledge elicitation, but do not contain any knowledge about how the interaction should proceed. Knowledge acquisition tools designed for a particular application can contain knowledge about the domain that is useful for guiding the knowledge elicitation session. A third category of knowledge acquistition tools are domain independent but contain knowledge about how a knowledge elicitation session should proceed.

3.9.3.2 Knowledge Base Editors and Browsers

Knowledge acquisition can be facilitated by the use of knowledge base editing environments. These provide easy (often graphical) access to a knowledge base so that both its overall structure and its detail can be surveyed. To some extent, commercial toolkits such as KEE (Kehler & Clemenson, 1984), which have some graphical knowledge editing facilities, could be said to fall into this category of knowledge acquisition tools. However, knowledge-based system toolkits and shells are generally concerned with the machine representation of knowledge and their knowledge base editing facilities operate at this level.

To be useful for knowledge acquisition, a knowledge base editor interfaces with the user at a higher level than the basic machine representation of knowledge; the user is concerned with manipulating graphical objects or natural language expressions, not rules, frames or computer languages. Facilities must then be provided for translation of the high level into the machine level representation.

Knowledge base browsers and editors designed for a particular application can be thought of as containing a model of the domain in which they operate. The difference between these tools and the domain dependent knowledge acquisition tools described in Section 3.9.3.3 is that the domain model is passive - the tool is completely user-driven and no explicit information is given about how to proceed with knowledge acquisition.

A knowledge acquisition environment for the domain of aerial image interpretation was developed with the aim of eliminating the distortion in knowledge that occurs when a knowledge engineer acts as an intermediary between domain experts and the computer (Tranowski, 1988). The system was designed to allow experts to input knowledge in a format that is tailored to the domain, not the way in which it is to be represented in the computer.

The KREME system (Abrett & Burstein, 1987) contains editors for knowledge expressed as frames, procedures & rules. The system contains meta-knowledge about the nature of and interaction between different types of knowledge. The interface is a graphical windowing system with the facility to analyze knowledge in more detail by pointing to it with a mouse (in the manner of hypertext). Frame hierarchies can be displayed as lattices of nodes and each frame can be browsed by pointing to its node in the lattice. The frame structures are the central feature of the system with rules, procedures and methods all being attached to them.

3.9.3.3 Domain Dependent Knowledge Acquisition Tools

Domain dependent knowledge acquisition tools are designed for a particular application and use knowledge of the domain to guide the expert through the process of knowledge elicitation. The tool can be considered as having a model of the domain that is parallel to the one elicited from the expert (Mars & Miller, 1987).

A knowledge acquisition tool called OPAL (Musen *et al*, 1987) has been developed to elicit knowledge about cancer treatment protocols for the ONCOCIN system (Shortliffe *et al*, 1981). Two types of knowledge are incorporated in ONCOCIN. Generators are structures which define the sequence in which a patient receives chemotherapy or radiotherapy and the transitions in patient state. The generators describe the basic treatment plan for the patient; details of particular therapies and drugs are represented as knowledge frames. Rules associated with the frames are used to determine details of the protocol according to specific patient data.

OPAL mirrors the two types of knowledge representation - generators and frames - in its graphical interface with the user. The sequence of states and therapies in a protocol can be specified by using a graphical tool in which icons representing therapies, states and decisions are selected from a palette and linked to form a flowchart. The flowchart is translated by OPAL into an ONCOCIN generator. The acquisition of knowledge to be represented as frames or rules is achieved by the use of forms.

The forms are structured representations of ONCOCIN's knowledge, presented in a manner that is familiar to clinicians (often the form replicates one of the paper forms used routinely in patient management). The forms are used by clinicians to *fill in the blanks* with information about drugs or laboratory tests - this information is then translated into frames or rules as appropriate.

3.9.3.4 Domain Independent Knowledge Acquisition Tools

Domain independent knowledge acquisition tools can be thought of as knowledge-based systems whose task domain is knowledge elicitation *ie* they contain knowledge about how to elicit knowledge. One of the most widely used knowledge elicitation methodologies incorporated into such systems is the *repertory grid technique*, based on personal construct theory (Kelly, 1955). According to this theory, the human's perception of the world is represented internally by cognitive structures called *constructs*. Contsructs are based on sets of elements and specify the differences and similarities between the elements perceived by an individual according to his past experience. Using his personal constructs, an individual attempts to predict and control his environment.

Personal constructs can be directly and accurately elicited using a repertory grid (Easterby-Smith, 1980). Such a grid consists of a set of *elements, constructs* and a method of *linking* the two together. The elements can be thought of as the fundamental objects in a cognitive process and the constructs as characteristics of these objects. A contruct is represented by two opposite poles which denote the two extremes of a characteristic on a linear scale. Figure 3.28 shows a simple repertory grid in which each column represents an element and each row a construct; note that the elements must form a homogeneous set, in this case a set of US Presidents.

	Kennedy	Nixon	Carter	Reagan	Bush	
Honest	√	X	√	X	√	Dishonest
Sharp	√	√	X	X	√	Muddled
Sensitive	√	X	√	√	X	Insensitive
Popular	√	X	X	√	√	Unpopular
Liberal	√	X	√	X	X	Conservative

(a) Simple Yes/No Rating

	Kennedy	Nixon	Carter	Reagan	Bush	
Honest	8	1	9	4	6	Dishonest
Sharp	9	7	5	1	7	Muddled
Sensitive	8	2	8	7	4	Insensitive
Popular	10	1	4	9	7	Unpopular
Liberal	9	4	7	3	3	Conservative

(b) Rating on a Scale of 1-10

	Kennedy	Nixon	Carter	Reagan	Bush	
Honest	2	5	1	4	3	Dishonest
Sharp	1	3	4	5	2	Muddled
Sensitive	2	5	1	3	4	Insensitive
Popular	1	5	4	2	3	Unpopular
Liberal	1	3	2	5	4	Conservative

(c) Rating Based on a Ranking Scheme

Figure 3.28 Repertory Grids

The constructs can be chosen by considering sets of three elements and finding ways in which any two are similar and different from the third. The similarity-difference pair form the poles of a construct. Each position in the body of the grid is filled by a link for the element-construct pair it represents. The most usual method of linkage is by adopting some sort of rating scheme. This could be a simple yes/no rating, a score (on the scale 1 to 10 say) or a ranking for each element for the given construct. These methods are illustrated in Figures 3.28a,b,c.

There are several ways in which a repertory grid can be analyzed. The grid can be *focussed* by grouping together constructs which have similar rows of links and elements with similar columns. In this way it is easy to see which constructs and elements are similar to each other (if the rows for two constructs are identical, one of them is redundant). The same information is expressed by the *correlation matrices* for the constructs and elements. The correlation matrix for constructs is formed by finding the number of matching links in the rows in each possible pairing of constructs; the matrix for elements is formed with the number of matching links between columns in the grid.

A number of computer tools have been developed to assist with the construction and analysis of repertory grids, including KITTEN (Shaw & Gaines, 1987), KRITON (Diederich *et al*, 1987) and AQUINAS (Boose, 1988). These tools are best suited to elicitation of knowledge for problems of diagnosis, classification and interpretation (Boose, 1988). They assist the user with the choice of constructs, specification of links, focussing and construction of correlation matrices and can generate knowledge bases of production rules directly from the repertory grid.

3.9.4 Summary

The process of knowledge acquisition is an important part of the development cycle of a knowledge-based system. Knowledge can be acquired from textbooks, analysis of large quantities of data or directly from domain experts. There are two basic strategies for eliciting knowledge from experts - interview and observation. Interviews can be informal and unstructured or can be structured, using techniques such as distinguishing the goals, reclassification or forward scenario simulation. Observational strategies involve the analysis of an expert engaged in problem solving (either on real-life domain problems or on contrived problems). Observation of the expert solving real-life problems is called protocol analysis and has been found insufficient to elicit a complete and accurate knowledge base. In practice, a combination of observational and interview strategies may be most effective.

Three types of computer aids to knowledge acquisition have been identified. Knowledge base editing environments allow users to input knowledge at a higher level than the internal knowledge representation but do not assist directly in knowledge acquisition. Domain dependent tools have a model of the domain which is used to guide users through the process of knowledge acquisition. Domain independent tools contain detailed knowledge about the process of knowledge acquisition itself. Several tools of this type have been developed using the technique of repertory grid analysis.

3.10 Summary

This chapter has described three general methods of knowledge representation - production rules, frames and semantic networks - and indicated how this knowledge might be manipulated in a knowledge-based reasoning system. The blackboard architecure has been presented as a method for overall control of a system which displays opportunistic problem solving behaviour and incorporates different sources of knowledge.

Two important issues, particularly relevant to knowledge-based systems for laboratory data interpretation, have been discussed. The first of these, data classification, has proved to be a problem both in knowledge-based system design and in laboratory medicine itself. Several methods have been described in which knowledge can be applied to the classification of data in order to avoid excessive information loss. The second issue concerned the handling of uncertainty. Four approaches to reasoning in the presence of uncertainty have been presented: *ad hoc* scoring schemes, probabilistic methods, application of the Dempster-Shafer Theory and fuzzy techniques from Possibility Theory. It is proposed that Probability Theory offers the only completely consistent method of handling uncertainty and it will be demonstrated in the next chapter that the apparent richness of expression that has encouraged the use of the other methods can be captured implicitly within a probabilistic framework.

A brief overview of the extensive subject of dialogue interaction between man and machine was presented as preparation for the last section in Chapter 5. Finally, the problems of acquiring knowledge for a knowledge-based consultation system were discussed.

The overall design, knowledge representation and knowledge acquisition for a knowledge-based system for the interpretation of laboratory data are presented in the next Chapter.

PART TWO

CHAPTER FOUR

THE EVOLUTION OF A KNOWLEDGE-BASED SYSTEM

4.1 Introduction

This chapter describes the evolution of the design and the process of knowledge acquisition for a knowledge-based system for the interpretation of the results of blood gas analysis. The overall guidelines for the design, as have been stated in Chapter 1, were:

(1) to make the system easy to use and to keep interaction between computer and clinician to a minimum.
(2) to combine different methods of knowledge representation and reasoning within a single system.
(3) to provide detailed explanation of the system's conclusions and of the domain of acid-base balance in general.
(4) to make the system's knowledge easily accessible to clinicians for review and update

The process of knowledge acquisition and the development of a design for the knowledge-based system followed the cycle described in Section 3.9.1 and depicted in Figure 3.27. An initial period was spent gaining background knowledge of the physiology of cardio-pulmonary homeostasis (see Appendix I for an overview) during which time a tentative system architecture was proposed. Armed with this background knowledge and initial system design, a series of interview sessions was held with critical care clinicians in the expectation of eliciting the knowledge necessary to build a prototype system; these sessions are described in Section 4.2.

In fact, the initial knowledge acquisition sessions served only to confirm the structure of the domain, the types of knowledge representation required and the architecture of the prototype system; it proved impossible to elicit the detailed knowledge required for a full prototype implementation. However, enough insight was gained to enable the development of a detailed system design, which is presented in Section 4.3. At the same time, a knowledge acquisition tool was developed in order to facilitate the elicitation of detailed domain knowledge and to make the knowledge base easily accessible for review and update by clinicians, which was one of the original project aims. Section 4.4 contains a description of this knowledge acquisition tool and Section 4.5 describes its use in knowledge elicitation sessions at two hospitals.

4.2. An Initial Phase of Knowledge Acquisition

4.2.1 Introduction

Before beginning knowledge acquisition in a clinical environment, background knowledge of the physiology of acid-base regulation and of the techniques and equipment involved in blood gas analysis was acquired by a review of the medical literature and informal discussion with clinicians (Chelsom *et al,* 1987a). Taxonomies of disorders of hypoxaemic state and acid-base balance were produced as a framework for classification as shown in Figure 4.1.

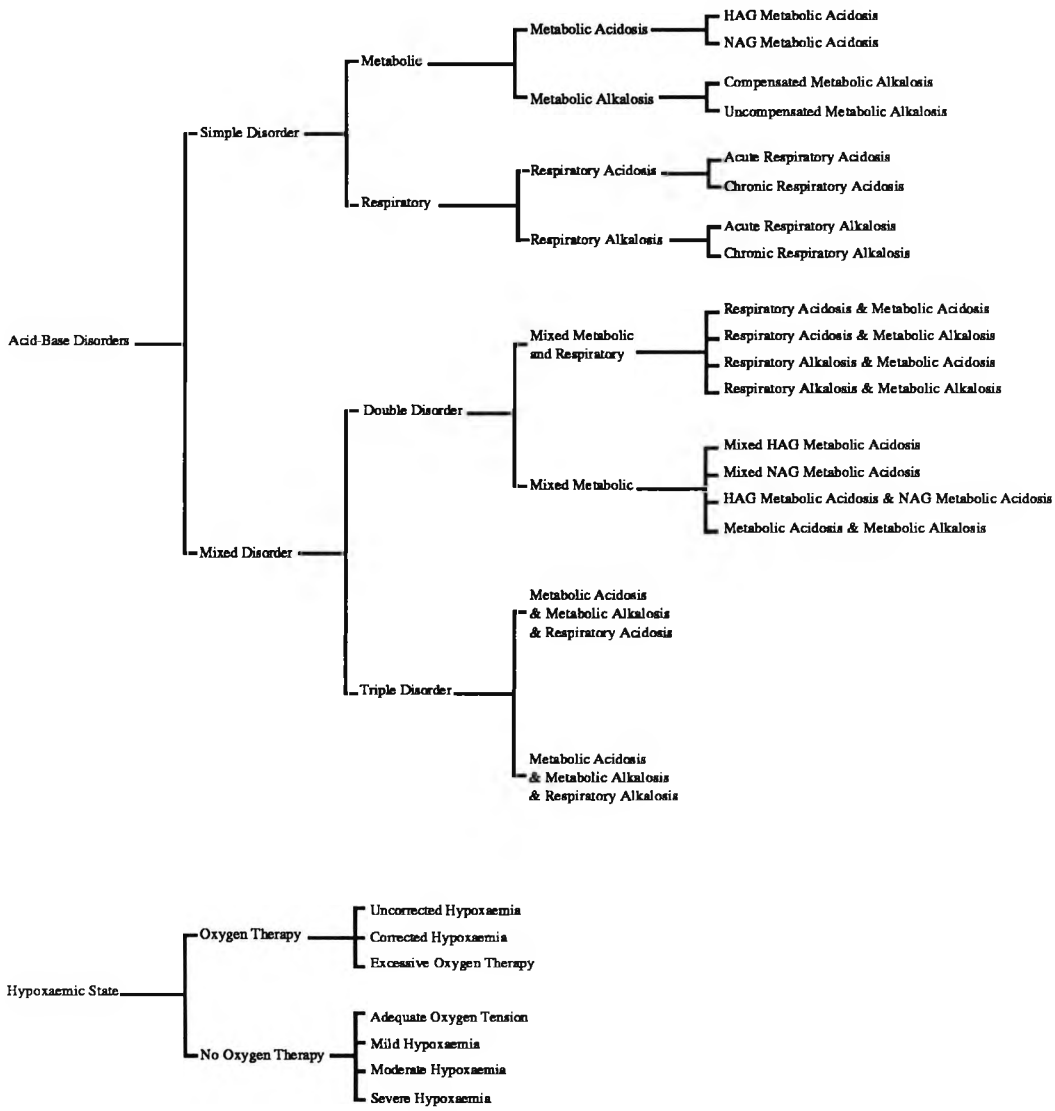


Figure 4.1 Taxonomies of Acid-Base Balance and Hypoxaemic State Disorders (HAG is High Anion Gap, NAG is Normal Anion Gap)

At this stage it was postulated that the interpretation of blood gas analysis results was achieved in two phases; blood gas data are used to infer the presence of acid-base and hypoxaemic state disorders and these conclusions are then used in conjunction with further clinical observations to diagnose clinical disease states. The interpretation could therefore be considered on two levels - a physiological level and a clinical level. Based on this conjecture, a preliminary architecture was proposed for a knowledge-based system to interpret the results of blood gas analysis (Chelsom *et al,* 1987b). A blackboard architecture (see Section 3.7) was selected for the overall control structure of the system and the interpretation of data at two levels - physiological and clinical - was modelled by splitting the blackboard into two panels. Figure 4.2 shows the preliminary design for the blackboard diagnostic system.

In order to build a prototype system, it was decided to elicit knowledge from clinical staff at the Intensive Care Unit of the Royal Free Hospital by holding a series of structured interviews (see Section 3.8). These interviews are described in the following section.

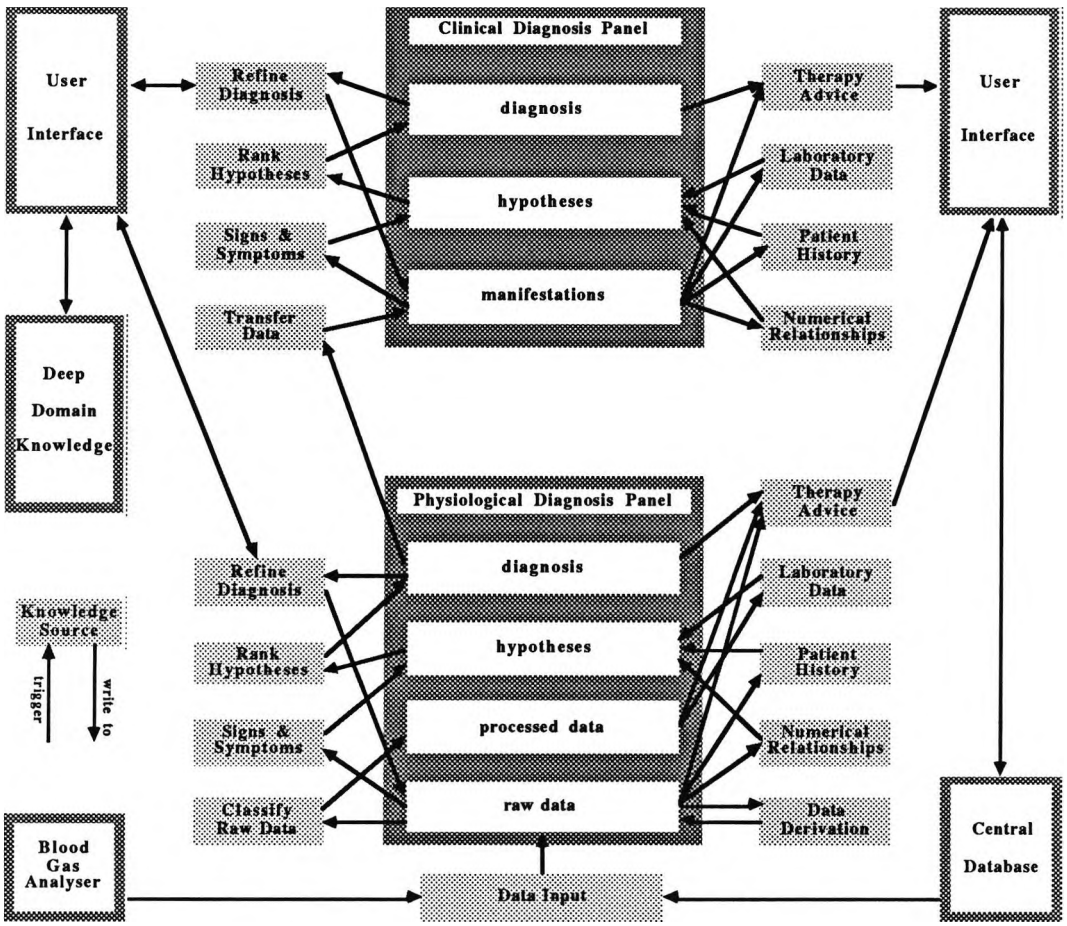


Figure 4.2 Preliminary Design for a Knowledge-Based System

4.2.2 Structured Interview Sessions

A series of five interviews was held at the Royal Free Hospital over a period of approximately one month. Each interview was attended by two knowledge engineers, the senior ICU technician and up to two registrars. The sessions were recorded on audio tape and the tapes were analyzed to extract useful information. It was expected that these sessions would:

(1) determine the general working procedures in the ICU and the role (if any) that could be played by computerized consultation systems

(2) determine the equipment available, variables measured and terminology used

(3) verify the preliminary knowledge-based system design

(4) elicit detailed knowledge of the signs & symptoms, history and temporal characteristics of acid-base and hypoxaemic state disorders.

The structure of the first interview session is shown in Figure 4.3. At the start of the interview, one knowledge engineer briefly described the project and the purpose of the series of interviews. There then followed a general discussion of the role of computers in the ICU which revealed that the clinicians were favourably disposed to their use as diagnostic and management aids, and that the specific application for blood gas analysis would be useful.

SESSION ONE: Introduction

Introduction to the project

General discussion of computers in the ICU
What are the needs?

Patient management: admission-management-discharge
What is the general pattern?

How does blood gas analysis fit into general patient management?

When is blood gas analysis called for?
Under what circumstances?
How often?

Do you use laboratory analysers or ICU-based analysers?

Review the structure of the next sessions

Figure 4.3 Structure of the First Interview Session

Overall, the session established a liaison between the clinical staff and the knowledge engineers and enabled the latter to gain an overall picture of the working practices of the ICU. In this respect, it was found that it is difficult to isolate blood gas analysis from general patient management, for the purposes of therapy and the decision was made to omit the therapy modules from the prototype implementation; a related project has dealt with the management of patients receiving mechanical ventilatory support (Summers *et al,* 1988).

The second session (Figure 4.4) was designed to define the diagnostic categories of acid-base balance and the relevant data used. The classification of disorders shown in Figure 4.1 was used as the basis for the discussion of diagnostic categories. It was found that triple and mixed metabolic disorders were not considered as diagnoses, metabolic alkaloses were not distinguished as compensated or uncompensated and that metabolic acidoses were sub-classified according to their underlying causes. This led to the simplified classification for acid-base disorders shown in Figure 4.5.

An important aspect of overall strategy was revealed at this second interview. The clinical condition was usually known before blood gases were taken; blood gas data were not used for making clinical diagnoses, rather the acid-base disorder that they indicated was compared with the disorder expected on the basis of the patient's clinical condition. This suggested a revision of the original design, to allow the the clinical diagnosis panel to operate in a top-down fashion, using the input of a clinical diagnosis to drive a critique based on the disorders diagnosed on the physiological panel.

SESSION TWO: Inputs and Outputs

Feedback from session one

Which data from the blood gas analyser do you use?

What are the units of measurement and normal ranges of these data?

Which other measured data do you use in conjunction with the blood gas data?
eg Haemoglobin, electrolytes

Which other derived data do you use?
eg Anion gap

What do you expect to learn from blood gas analysis?
ie review classification and terminology of disorders

Figure 4.4 Structure of the Second Interview Session

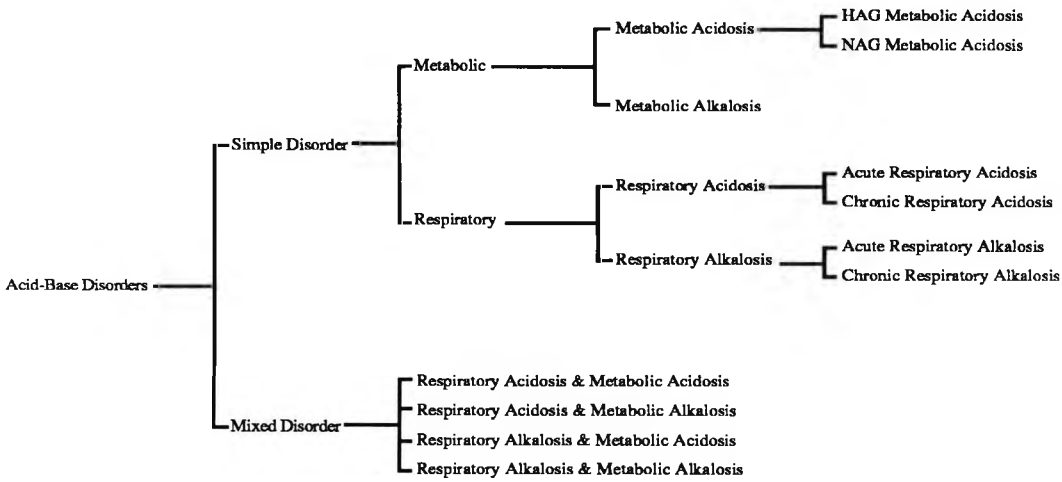


Figure 4.5 A Simplified Classification of Acid-Base Disorders

Problems with the structured interview approach began to emerge during this second session. Too often the interview lapsed into a general discussion of the domain; it became difficult to impose the intended structure on that discussion. Moreover, the questions asked by the knowledge engineers were too general, leading the clinicians to express doubts that their expertise was being adequately captured. It would, they felt, be easier to respond to more specific questions, but these could only be asked by a knowledge engineer who already possessed the detailed knowledge of the domain that the interview session was intended to reveal.

SESSION THREE/FOUR: Acid-Base Disorders

Feedback from last session

Using the list of disorders agreed in Session Two, what information do you use to identify each one?

eg
signs & symptoms — FACTS
history
temporal characteristics
causes

How do you decide that a patient has a certain disorder? — STRATEGY

Work through some example cases

Figure 4.6 The Third and Fourth Interview Sessions

SESSION FIVE: Hypoxaemia & Respiratory Function

How do you decide that a patient is hypoxaemic?

What clinical information do you use?

How do you take account of factors such as:
age & sex of the patient
inspired oxygen content

How do you decide whether to mechanically ventilate
a patient? What information do you use?

Figure 4.7 The Fifth Interview Session

The problem of directing a vague and rambling discussion that emerged in the second interview became even more pronounced in the next three sessions (Figures 4.6, 4.7) which were intended to elicit detailed knowledge of the relationships between observations and diagnoses. Whilst a general impression of these relationships could be sketched, it was not possible to gain a complete and accurate picture. At this stage, the form of knowledge representation to be used in the knowledge-based system had not been finalized and so it was not possible to use this to structure the way in which knowledge was elicited. In particular there was no framework in which the clinicians could consistently express the uncertainty inherent in the process of diagnosis. The knowledge engineers gained little knowledge from the final three sessions that they had not been able to acquire from a study of medical textbooks.

4.2.3 Summary

The conclusions that can be drawn from analysis of the structured interview sessions described above are:

(1) the structured interview was useful for establishing details of general working practice not usually published in medical texts

(2) it provided information on the laboratory data used, units of measurement, terminology and diagnostic categories

(3) it did not elicit comprehensive knowledge of relationships between observations and diagnoses, the uncertainty associated with such

relationships or the way in which data should be used to discriminate between competing hypotheses.

It was felt that enough knowledge of the structure of the task and the type of knowledge used in diagnosing acid-base disorders had been gained to enable a prototype system to be built. This prototype would work with a knowledge base structured to accommodate the types of knowledge identified in the initial knowledge acquisition phase. It was decided to develop an editing environment for the knowledge base and to use this tool in a second phase of knowledge acquisition to capture the detailed knowledge that had been impossible to elicit in the interview sessions.

Since the prototype diagnostic system would be built to operate on a knowledge base of known structure but unknown content, the opportunity existed to create a domain-independent system that could be used not only for the interpretation of blood gas data but also for any similar problems of laboratory data interpretation. The design of a knowledge-based system for the interpretation of laboratory data and details of the knowledge editing environment are presented in the next two sections.

4.3 Design of a Knowledge-Based System for Laboratory Data Interpretation

4.3.1 Overview

The overall design of a knowledge-based system for the interpretation of laboratory data is shown in Figure 4.8. The user has control over four system modules: data input, diagnosis, diagnosis presentation and dialogue management. At the start of a consultation the user inputs the clinical condition (diagnosis) of the patient and a basic set of laboratory data - the size and nature of this set depends on the data available. The blackboard diagnostic module then produces a differential diagnosis of disorders and a critique of the clinical diagnosis based on the disorders diagnosed. Although the implemented version of this system requires the user to input all data at the keyboard, the initial set of laboratory data could be transferred on-line from laboratory equipment, which would allow an initial diagnosis to be made without any interaction between computer and clinician.

Once the initial diagnosis has been made, control passes to the diagnosis presentation module which allows the user to view the differential diagnosis lists and to input signs, symptoms, patient history or further laboratory data. The diagnoses and critique can also be accessed in textual form through an interaction with the dialogue manager. This module also allows the user to ask questions about the diagnoses in order to receive explanations about the conclusions reached by the system; the facility to interrogate the knowledge base itself has been developed as part of another study (Smith, 1988).

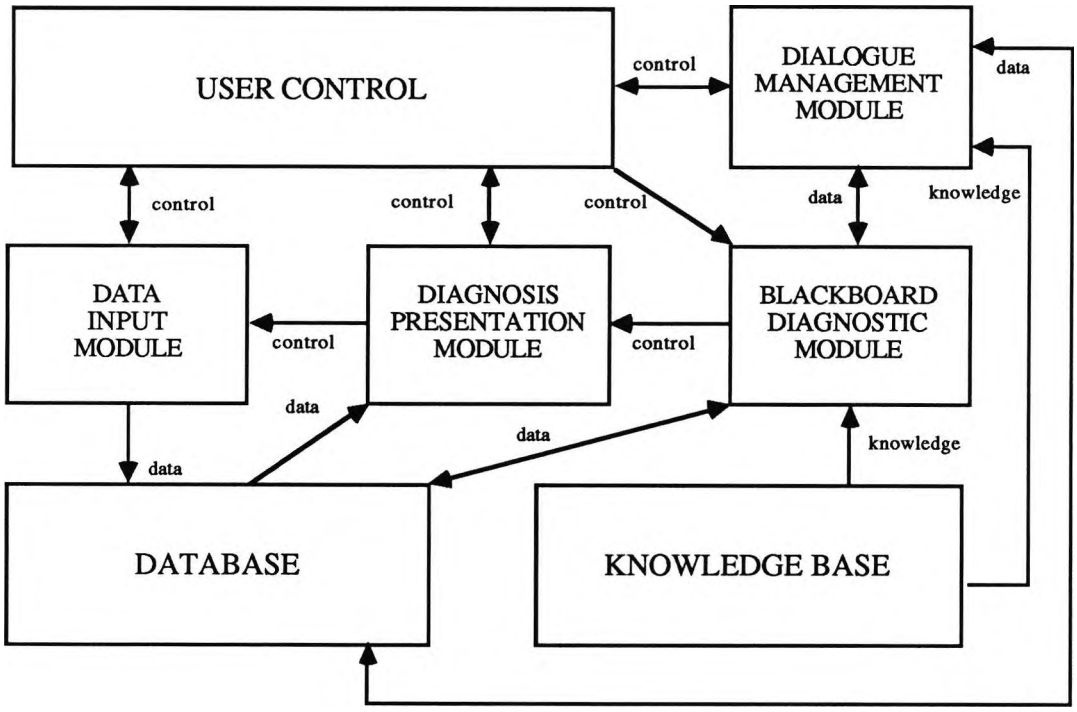


Figure 4.8 Design of a Knowledge-Based System: Overall Structure

4.3.2 The Blackboard Diagnostic Module

The organization of knowledge sources in the blackboard diagnostic module is shown in Figure 4.9. It preserves the basic architecture of the preliminary design in Figure 4.2; a dual panelled blackboard reflects the distinction between diagnosis at the physiological and clinical levels. The lower, physiological diagnosis panel operates in a bottom-up fashion inferring diagnoses from observations. Unlike the preliminary design, the clinical diagnosis panel operates top-down, critiquing the clinical diagnosis in the light of the conclusions from the lower panel.

The physiological diagnosis panel is split into four levels: *raw data, processed data, hypotheses* and *diagnoses*. Data from the database (signs, symptoms, history or laboratory data) are written on to the lowest level (raw data) by the *write data* knowledge source. The *data derivation* knowledge source monitors the raw data level and derives the value of new data when the necessary information becomes available. Laboratory data written to the raw data level are classified at the processed data level by the *classify raw data* knowledge source - the method used for this classification is presented in Section 4.3.4. Since the user may wish to alter the values of previously input data, a *truth maintenance* knowledge source monitors changes at the raw data level, ensuring that these are propagated through to any dependent derived data and to the processed data level.

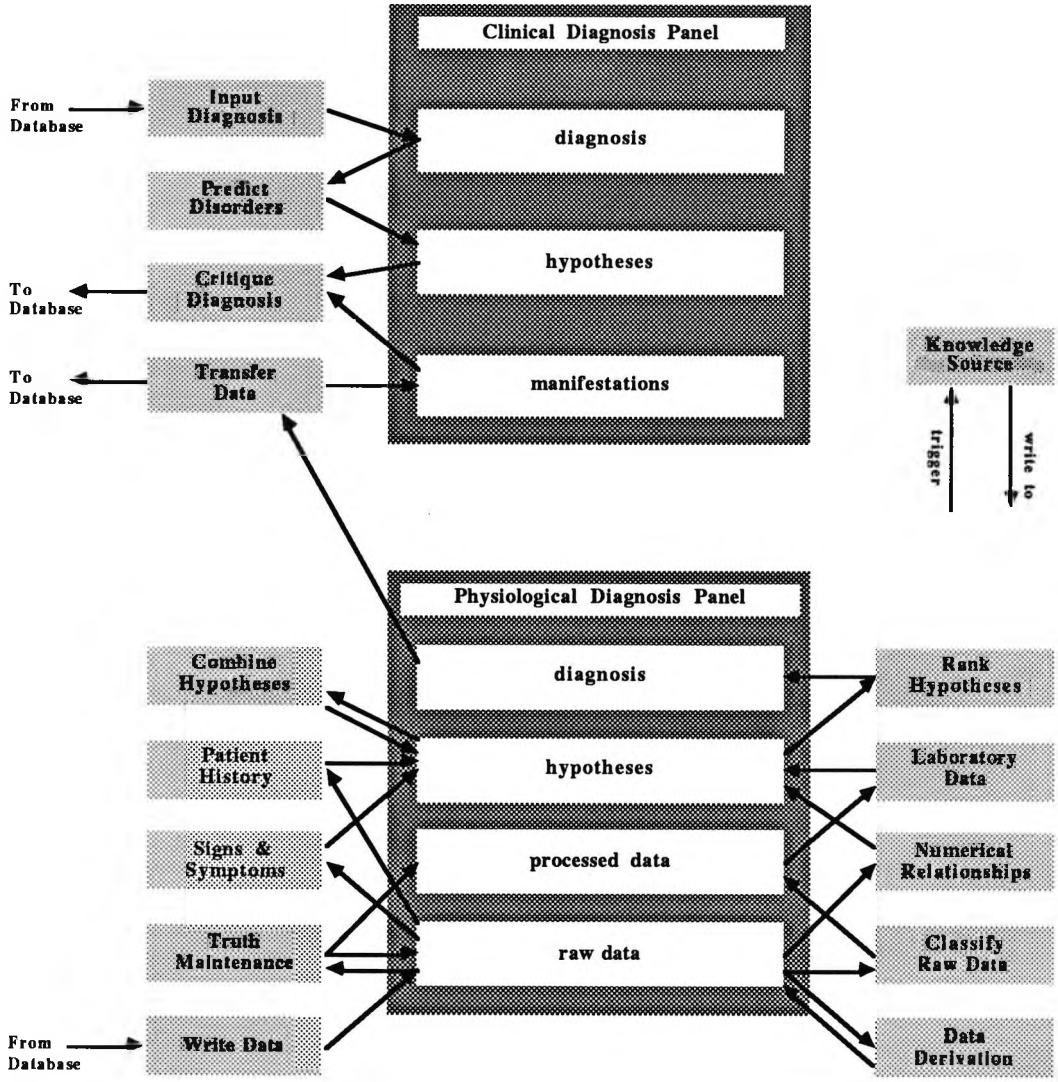


Figure 4.9 Design of the Blackboard Diagnostic Module

Four evidence handling knowledge sources - *numerical relationships, signs & symptoms, laboratory data, patient history* - generate hypotheses of physiological disorders from processed or raw data. The knowledge base contains details of the associational links between observations and disorders, which are organized into hierarchical classification structures. The effect of observations as evidence is impacted on the hypothesis hierarchies using the method that will be presented in Section 4.3.3.

Invoking of the evidence handling knowledge sources will usually generate several hypotheses of varying degrees of belief for each diagnosis in the hierarchy of a disorder class. These multiple hypotheses are aggregated by the *combine hypotheses* knowledge source to give a single hypothesis for each disorder. The hypotheses are then written by the *rank hypotheses* knowledge source as differential lists at the diagnosis level, ranked according to the degree of belief in each alternative. This knowledge source also ensures

that the diagnosis is reported at the most appropriate level of generality. For example, if the disorder metabolic acidosis is sub-classified as compensated or uncompensated and both sub-classifications are confirmed as diagnoses, then the single diagnosis of metabolic acidosis is reported.

Output of disorder diagnoses from the top level of the physiological diagnosis panel is written both to the database and to the lowest level (manifestations) of the clinical diagnosis panel. The presence of a diagnosed disorder in the database triggers the write data knowledge source so that the disorder appears as raw data at the foot of the physiological diagnosis panel, where it can act as evidence for disorders of another class (*eg* the presence of hypoxaemia can lend support to hypotheses of acid-base disorders).

On the clinical diagnosis panel, the *input diagnosis* knowledge source writes clinical diagnoses from the database to the uppermost, diagnosis level. The knowledge source *predict disorders* lists at the hypotheses level, the disorders expected to occur with each clinical diagnosis. These hypotheses are then compared with entries at the manifestation level to produce a critique of the clinical diagnosis based on the differences between the expected and observed disorders.

4.3.3 A Method for Updating Belief in a Hierarchy of Hypotheses

4.3.3.1 Introduction

The first generation of knowledge-based medical consultation systems used *ad hoc* scoring mechanisms to deal with the uncertainty inherent in medical diagnosis (see Chapter 1). These mechanisms were not only able to account for the frequency with which a manifestation occurred in a given disease state, but could also distinguish between the major and minor manifestations of a disease and the degree to which a given manifestation must be explained by the final diagnosis.

More recently, efforts have been made to structure the hypotheses generated by knowledge-based systems and to use more formal and consistent approaches to the handling of uncertainty, based on the Dempster-Shafer theory, probability theory or possibility theory (see Section 3.7). Often the commitment to a formal and consistent handling of uncertainty must be made at the expense of the flexibility embodied in the *ad hoc* methods of earlier systems.

The next section describes a method for updating belief in a hierarchical hypothesis space. It is demonstrated that many of the desirable features of the scoring mechanisms of systems such as PIP or INTERNIST-1 can be implicitly embodied within a more formal approach.

4.3.3.2 A Method For Evidence Handling

A method for evidence handling was developed to manage the impact of evidence on the belief of hypotheses organized as a hierarchy, in which the root node describes a class of diagnoses and the leaf nodes form a mutually exclusive and exhaustive set of hypotheses, H, for that class. An intermediate hypothesis, S, is the disjunction of its immediate descendents and can be thought of as a subset of H, whose members are the leaf nodes which are descendents of S.

Pearl (1986) outlines a method of Bayesian updating of belief in this type of hypothesis hierarchy , as has been described in Section 3.7.3.2. The knowledge base is specified in terms of the likelihood ratio, λ, for evidence e at a node S, where $\lambda = P(e|S)/P(e|\sim S)$. Specification of evidence for the node S provides no information for discrimination between its individual members *ie* $P(e|S)=P(e|h_i)$ where h_i is any leaf node descendent of S. The degree of belief in the leaf node h_i is defined as:

$$Bel(h_i) = P(h_i|E)$$

and E is $\{e_1,e_2,...e_n\}$, the set of evidence impacted on the hierarchy. The belief in an intermediate node is found by summing the belief of its descendent leaf nodes. Pearl demonstrates that a simple application of Bayes' Theorem can be used to define an updating factor w_i such that the impact of a new piece of evidence e_{n+1} on belief in the node h_i is given by:

$$Bel'(h_i) = P(h_i|E,e_{n+1}) = w_i.Bel(h_i)$$

where w_i is calculated in terms of the likelihhod ratio λ.

Pearl's method has been adapted to accommodate certain features of a practical knowledge base. These features are:

1) Probability assignments of the type $P(e|S_i)$ are made for nodes S_i at various levels of the hierarchy in such a way that no leaf node is a descendent of more than one S_i (*ie* $\cap S_i=\varnothing$)

2) There is no requirement that every leaf node is covered in the assignment of probabilities for a particular piece of evidence (*ie* $\cup S_i \supseteq H$)

Figure 4.10 shows a hierarchy in which the shaded nodes (S_2, S_3, S_4) have a probability assignment $P(e|S_i)$ in the knowledge base. Leaf nodes descended from S_2, S_3 or S_4 will be referred to as assigned nodes; they have been covered by the assignment of probability in the knowledge base. The remaining leaf nodes will be referred to as unassigned nodes.

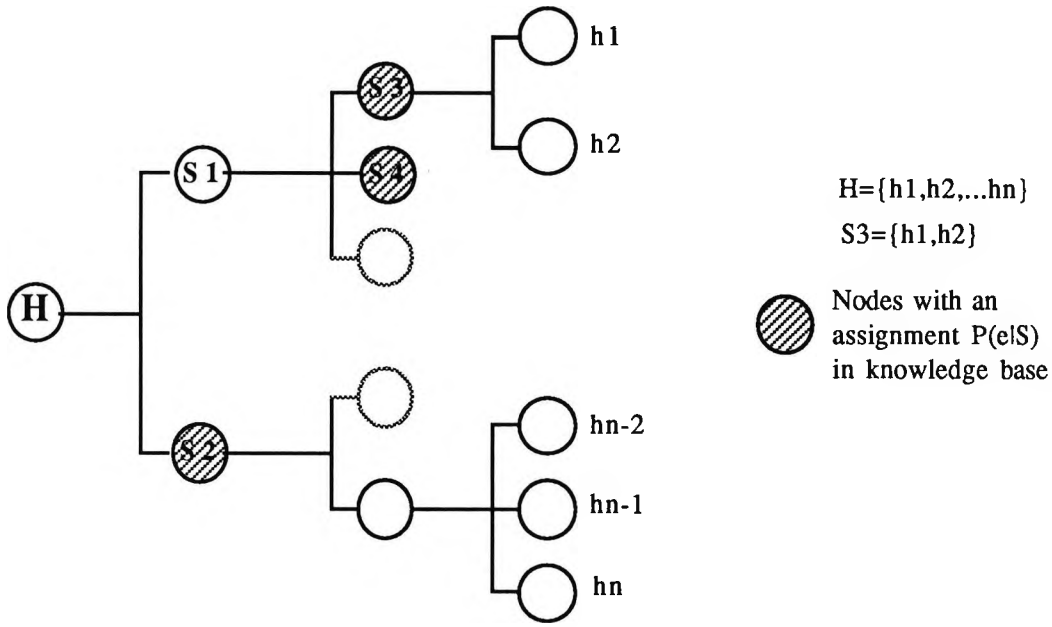


Figure 4.10 A Hierarchy of Hypotheses with Assigned Evidence

Consideration will now be given to the way in which the unassigned nodes are to be treated by the updating scheme, when the evidence e is impacted on the hierarchy. Since no indication has been given in the knowledge base about the association between an unassigned node h_u and the evidence e, it will be assumed that observation of e has no effect on the belief in h_u. Hence

$$P(h_u|e) = P(h_u) \tag{4.1}$$

where $P(h_u)$ is the probability of h_u prior to the evidence e.

Bayes' Theorem states:

$$P(h_u|e) = \frac{P(e|h_u).P(h_u)}{\Sigma_i P(e|h_i).P(h_i)} \tag{4.2}$$

where the summation Σ_i is for i=1 to n *ie* over the n leaf nodes.

Combining (4.1) and (4.2):

$$P(e|h_u) = \Sigma_i P(e|h_i).P(h_i) \tag{4.3}$$

which implies that $P(e|h_u)$ is a constant for all unassigned nodes. The summation in (4.3) can be split into the summation over the assigned nodes, denoted by Σ_a, and the sum over the unassigned nodes, denoted by Σ_u, so that (4.3) can be rewritten as:

$$P(e|h_u) = \Sigma_a P(e|h_i).P(h_i) + \Sigma_u P(e|h_i).P(h_i) \tag{4.4}$$

Now consider the summation of $P(e|h_i).P(h_i)$ over the unassigned nodes:

$$\Sigma_u P(e|h_i).P(h_i) = P(e|h_u).\Sigma_u P(h_i) \quad \text{since } P(e|h_u) \text{ is constant} \tag{4.5}$$

$$= P(e|h_u).(1-\Sigma_a P(h_i))$$

since the leaf nodes are mutually exclusive and exhaustive, and so $\Sigma_u P(h_i) = 1-\Sigma_a P(h_i)$. Substituting for $P(e|h_u)$ from (4.4) and rearranging:

$$\Sigma_u P(e|h_i).P(h_i) = \frac{(\Sigma_a P(e|h_i).P(h_i)).(1-\Sigma_a P(h_i))}{\Sigma_a P(h_i)} \tag{4.6}$$

Now consider the impact of evidence e on an assigned node h_a. Bayes' Theorem states:

$$P(h_a|e) = \frac{P(e|h_a).P(h_a)}{\Sigma_i P(e|h_i).P(h_i)} \tag{4.7}$$

where $P(h_a)$ is the probability of h_a prior to evidence e.

Splitting the summation in the denominator of (4.7), as before, into summation over the assigned and unassigned nodes:

$$P(h_a|e) = \frac{P(e|h_a).P(h_a)}{\Sigma_a P(e|h_i).P(h_i) + \Sigma_u P(e|h_i).P(h_i)} \tag{4.8}$$

Substituting for $\Sigma_u P(e|h_i).P(h_i)$ from (4.6):

$$P(h_a|e) = \frac{P(e|h_a).P(h_a)}{\Sigma_a P(e|h_i).P(h_i) + (\Sigma_a P(e|h_i).P(h_i)).(1-\Sigma_a P(h_i)\Sigma_a P(h_i))} \tag{4.9}$$

So,

$$P(h_a|e) = \frac{P(e|h_a).\Sigma_a P(h_i)}{\Sigma_a P(e|h_i).P(h_i)} . P(h_a) \tag{4.10}$$

Or $\quad Bel'(h_a) = w.Bel(h_a)$

where $$w = \frac{P(e|h_a).\Sigma_a P(h_i)}{\Sigma_a P(e|h_i).P(h_i)} \tag{4.11}$$

Hence the updating factor for assigned nodes is calculated directly from the probability assignments P(e|S) in the knowledge base.

In summary, it can be said that when a piece of evidence is impacted on the hypothesis hierarchy, the belief in a node remains unchanged if no association with the evidence has been specified in the knowledge base. The body of belief in the remaining nodes undergoes a Bayesian redistribution according to the formula in (4.10).

The updating factor in (4.10) is calculated using the probabilities of hypotheses immediately prior to the observation of evidence e. The updating of belief as successive pieces of evidence $\{e_1, e_2, e_3...\}$ are observed could be viewed as a Markov process in which each new state of belief distribution across the hypothesis hierarchy depends only on the preceding state. In this case, evidence would have a greater impact on hypotheses that were already strongly supported than on less favourable hypotheses. Whilst this may model, to some extent, the way in which a human diagnostician searches for, and weighs, evidence in favour of leading hypotheses, it was considered unsuitable because in cases where a number of pieces of evidence are reported simultaneously (the results of a laboratory test, for instance) the belief in final diagnoses would depend on the order in which the evidence was input to the system. For this reason, the updating factors used in the practical implementation of the evidence handling scheme are calculated using the *a priori* probabilities of hypotheses before any evidence was observed; each piece of evidence is treated as if it were the first observation.

4.3.3.3 Discussion

The scoring mechanism in INTERNIST-1 (Miller *et al*, 1982) used three numerical measures. The IMPORT of a manifestation was a measure of the extent to which it must be explained by the final diagnosis; the EVOKING STRENGTH of a manifestation in the profile of a disease measured the extent to which that disease explained the manifestation; the FREQUENCY of a manifestation, e, in the profile of disease S, corresponded to $P(e|S)$. Disease profiles in the Present Illness Program (Pauker *et al*, 1976), contained TRIGGER findings which could activate hypotheses, FINDINGS which could lend support to already activated hypotheses, MUST-HAVE and MUST-NOT-HAVE conditions for manifestations of the disease and IS-SUFFICIENT labels for findings pathogenic to the disease.

The prototype knowledge-based system for the interpretation of laboratory test results has two main cycles of operation. On the first cycle, the laboratory test data are entered and the belief in an initial set of hypotheses is calculated. On the second cycle, further signs, symptoms or patient history can be entered to update belief in the diagnostic hypotheses. Thus the laboratory data set acts in the same way as TRIGGERS in PIP or as manifestations with a high IMPORT value in INTERNIST-1.

INTERNIST-1's EVOKING STRENGTH and PIP's IS-SUFFICIENT findings are modelled by the distribution of probability assignments across the hierarchy of hypotheses. The EVOKING STRENGTH of a manifestation for a hypothesis will depend on the extent to which that manifestation appears in probability assignments for other hypotheses in the hierarchy; if it appears often, the implicit EVOKING STRENGTH is low; if it appears in few other assignments, the EVOKING STRENGTH is high. The IS-SUFFICIENT findings for a hypothesis are those that have assignments $P(e|S)=0$ across the rest of the hierarchy.

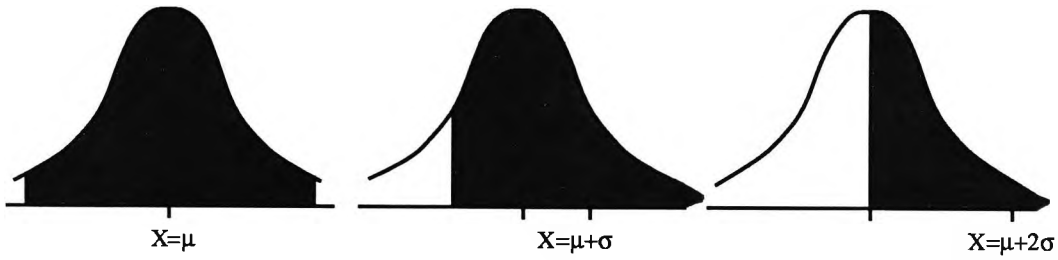
By making probability assignments for only a few nodes, specific pieces of evidence can be used to discriminate between targeted hypotheses. If, for instance, only two hypotheses, S_1 and S_2, are assigned probabilities for evidence e, with $P(e|S_1)=1$ and $P(e|S_2)=0$, then when e is observed the new belief in S_1 is the sum of the previous belief in S_1 and S_2, the new belief in S_2 is zero, and belief in other hypotheses remains unchanged.

4.3.4 Laboratory Data As Evidence

The knowledge base used for the interpretation of laboratory test results contains probability assignments regarding the level (low, normal or high) of data variables for particular diagnostic hypotheses. Thus for laboratory data variable V, the following probability assignments are stored for hypothesis S:

$$P(V \text{ is low}|S) \quad P(V \text{ is normal}|S) \quad P(V \text{ is high}|S) \tag{4.12}$$

Assuming that the probability distribution of V is known for a reference population of healthy individuals, a measurement of V can be defined as low if it falls further than 2 standard deviations (σ) below the mean value (μ), high if it falls further than 2σ above the mean and normal if it lies between these limits. Now, given that a particular value X, of variable V, has been observed in a patient, what is the probability that X is low, normal or high according to the definitions above? If the normal value of V for this patient, when healthy, is more than 2σ above X, then it has now fallen by more than 2σ and it should be classified as low. Thus the probability that X is low for this patient is the same as the probability that this patient's normal value lies more than 2σ above X. Similarly, the probability that X is normal is given by the area within $+/- 2\sigma$ of X, and the probabilty that X is high is given by the area more than 2σ below X. Using this method of classification, the probabilities that X is low, normal or high are shown in Figure 4.11 for $X=\mu$, $X=\mu+\sigma$ and $X=\mu+2\sigma$. Although the distributions shown are Gaussian, the method described is appropriate for any probability distribution.



P(X is low)	0.0228	P(X is low)	0.0014	P(X is low)	0.0001
P(X is normal)	0.9554	P(X is normal)	0.8399	P(X is normal)	0.4999
P(X is high)	0.0228	P(X is high)	0.1587	P(X is high)	0.5000

(a) **(b)** **(c)**

Figure 4.11 The Probability of a High, Normal or Low Value. (The shaded area is the probability that X is normal).

Using the probability assignments in (4.12), and the calculated probabilities that an observed value X is low, high or normal, the term $P(e|h_a)$ in (4.10) can be replaced by:

$$P(V\ low|S).P(X\ low)+P(V\ normal|S).P(X\ normal)+P(V\ high|S).P(X\ high)$$

Figure 4.12 shows the comparison of the method presented above with Tango's method (Section 3.6.4) for values of the individual difference quotient of 0.2, 1.0, 1.8 and 2.6. It can be seen that within the range of θ observed by Tango (0.56-2.35), the new method provides a more conservative classification: the probability that an observed value is normal falls off less sharply with the distance from the mean. In particular, at two standard deviations from the mean the probability is 0.5 with the new method, 0.422 for θ=0.2 and 0.204 for θ=1.

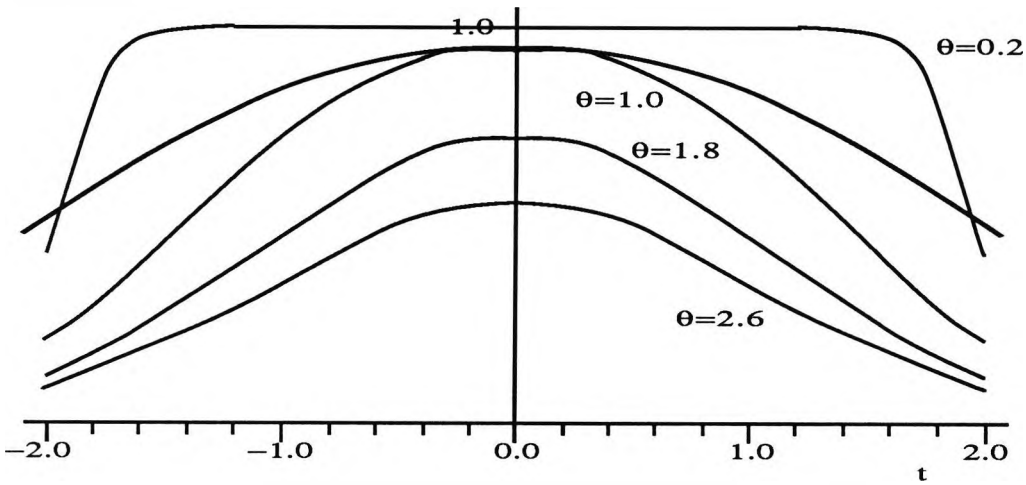


Figure 4.12 Comparison with Tango's Method of Data Classification. θ is the Individual Difference Quotient and t the number of standard deviations from the mean value of a measured variable. The new method of data classification is shown as the bold line.

4.4 FRAMEBUILDER: A Knowledge Editing Environment

4.4.1 Introduction

A knowledge editing environment, called FRAMEBUILDER, has been developed to enable clinicians to construct and browse knowledge bases suitable for use with the diagnostic system described in Section 4.3. FRAMEBUILDER is implemented in PROLOG2 (see Appendix VIII) and allows graphical, mouse-driven access to a knowledge base represented as PROLOG clauses. The advantages of using FRAMEBUILDER are these:

1) A clinician can construct a knowledge base by direct interaction with the computer, through a graphical interface - no knowledge of computer programming is required and the role of the knowledge engineer as an intermediary between experts and the computer is eliminated.

2) By making appropriate checks as each new piece of knowledge is added, it can be ensured that the knowledge base is logically consistent - this is an important aspect of the evaluation of a knowledge-based system and can be a time-consuming task if performed manually.

3) By providing a structured framework in which knowledge can be expressed, the knowledge editing environment facilitates the acquisition of detailed domain knowledge that is difficult to elicit by standard interview techniques.

4) The entire knowledge base can be browsed at any time, and modifications are made quickly and easily.

It can be seen that FRAMEBUILDER operates in a number of roles: as a tool for knowledge acquisition, as a convenient editor for an existing knowledge base, as a knowledge verification mechanism and as a means of explanation, by allowing a user to browse the knowledge on which the diagnostic system's conclusions are based.

There are a number of aspects of the use of FRAMEBUILDER (or indeed any such tool) that call for a cautious approach on the part of the clinician-user:

1) FRAMEBUILDER does not check the *completeness* of the knowledge base, only its internal *consistency*. Failure to appreciate this point could lead to over-confidence in the validity of the knowledge base.

2) Knowledge must be expressed in a certain format that may not be the most natural for a particular situation. This problem has been discussed in Section 3.9.

3) Once a knowledge-based system has been evaluated in a clinical setting, the facility to modify the knowledge within it must be carefully controlled, since any such modification would render the evaluation invalid.

4.4.2 Overview of FRAMBUILDER

The FRAMEBUILDER environment can be viewed on the three levels shown in Figure 4.13. At the level of *primitive data objects* are defined the characteristics of various laboratory data, signs and symptoms, aspects of patient history and relationships that exist between laboratory data variables. These should cover the complete set of observations that could be made for a patient. In addition, the broad classes of diseases and disorders for which diagnoses are to be made can be defined at this level.

At the second level, of *frame hierarchies*, hierarchies of clinical diseases and physiological disorders are defined; each hierarchy is headed by one of the disorders or diseases defined as a primitive data object. The hierarchies should satisfy the conditions of exhaustiveness and exclusivity specified in Section 4.3.3, but since checking of these conditions requires a complete prior knowledge of the domain, it is the responsibility of the user to ensure that they are complied with.

At the level of *frame instances*, each node in the hierarchies of diseases and disorders can be viewed as a frame of information for which characteristic observations are specified (chosen from the set defined at the level of primitive data objects) along with subjective estimates of the probability of the observation in the presence of the disorder or disease defined by the frame.

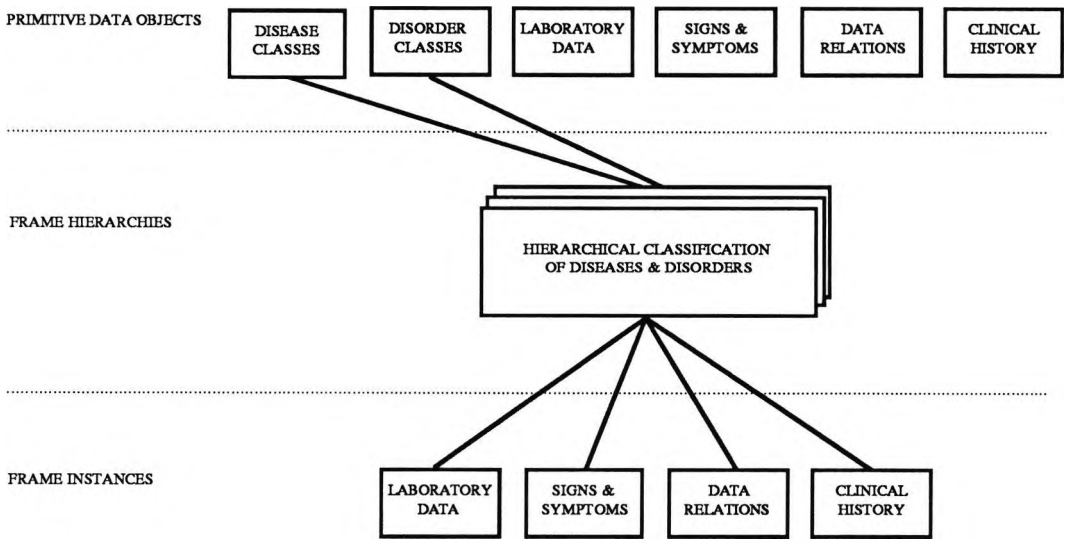


Figure 4.13 The Structure of FRAMEBUILDER

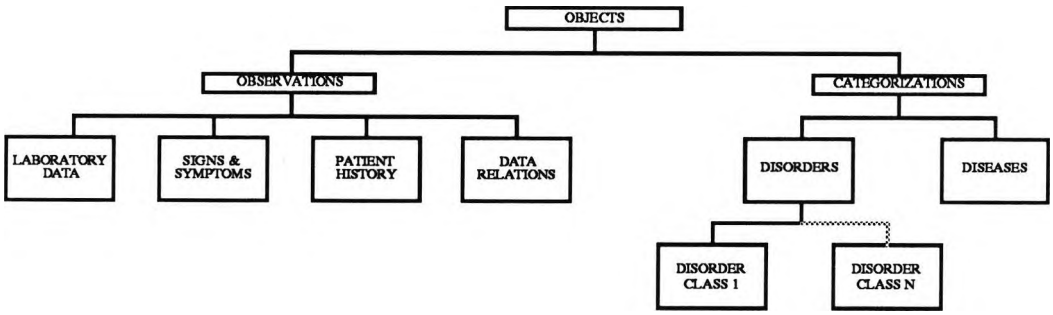


Figure 4.14 FRAMEBUILDER as a Frame-Based Organization of Knowledge

When used in its role as a knowledge acquisition tool, the three levels of FRAMEBUILDER correspond to the three levels involved in the technique of distinguishing the goals - definition of observations, definition of goals and definition of the relationships between observations and goals (see Section 3.9.2). A description of this three-levelled process of knowledge acquisition will be presented in Section 4.5.

In terms of a frame-based representation of knowledge, FRAMEBUILDER can be viewed as a tool for defining frames, the inheritance between them (through the frame hierarchies) and the possible facets and values for the frame slots (see Section 3.3). The frame-structured knowledge base created by FRAMEBUILDER is shown in Figure 4.14; the next three sections give a detailed description of the knowledge base and of FRAMEBUILDER itself.

4.4.3 Representation of Primitive Objects

A session with FRAMEBUILDER starts at the level of primitive data objects with the main command line shown in Figure 4.15. The user can select to edit disorders, diseases, laboratory data variables, signs/symptoms, relationships between variables or patient history by clicking with the mouse over the appropriate field in the command line.

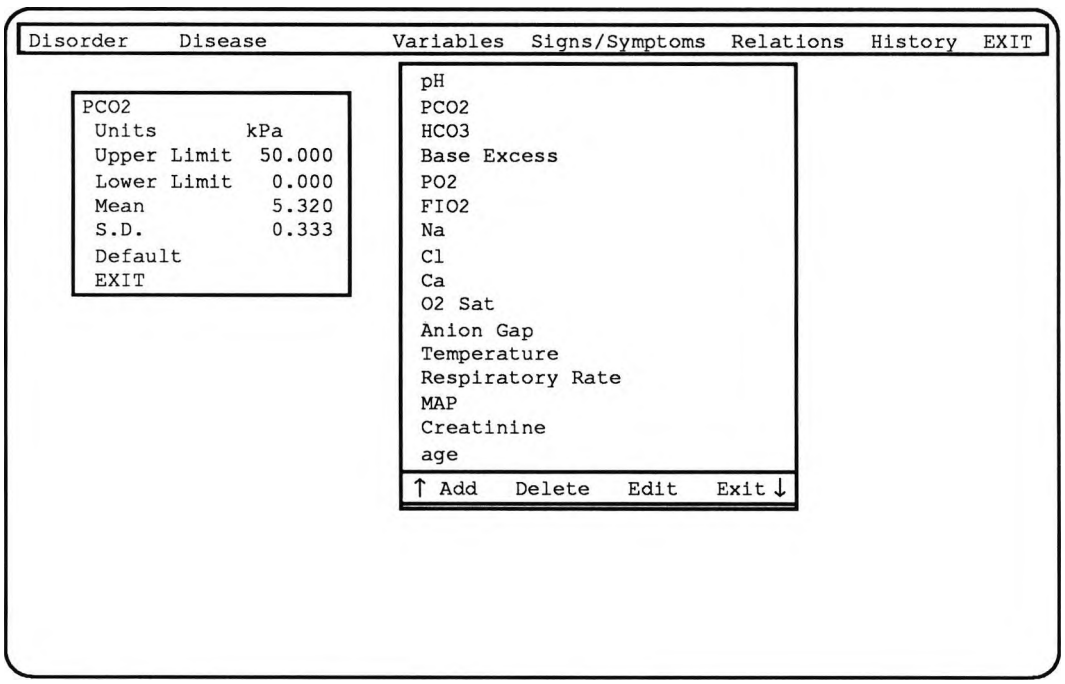
When the *variables* field is selected, a list of all laboratory data variables is displayed in a drop-down menu and the user has the option of adding new variables, deleting existing ones or editing the parameters associated with each.

```
Disorder    Disease        Variables  Signs/Symptoms  Relations  History  EXIT
```

Figure 4.15 The Main Command Line in FRAMEBUILDER

These parameters are the units of measurement, the upper and lower limits (defining the allowable range for input), the mean and standard deviation for a reference population (a Gaussian distribution is assumed) and the default value (if any). Figure 4.16a shows the screen after the user has chosen to edit the parameters of pCO_2; Figure 4.16b shows the PROLOG representation of pCO_2 as a laboratory data variable.

As can be seen in Figure 4.16b, the definition of a laboratory data variable produces an entry for that variable as a noun in the system's lexicon. This lexicon is used for the understanding of user queries in the dialogue management module of the diagnostic system. This process will be discussed in Section 5.7.



(a) FRAMEBUILDER Display

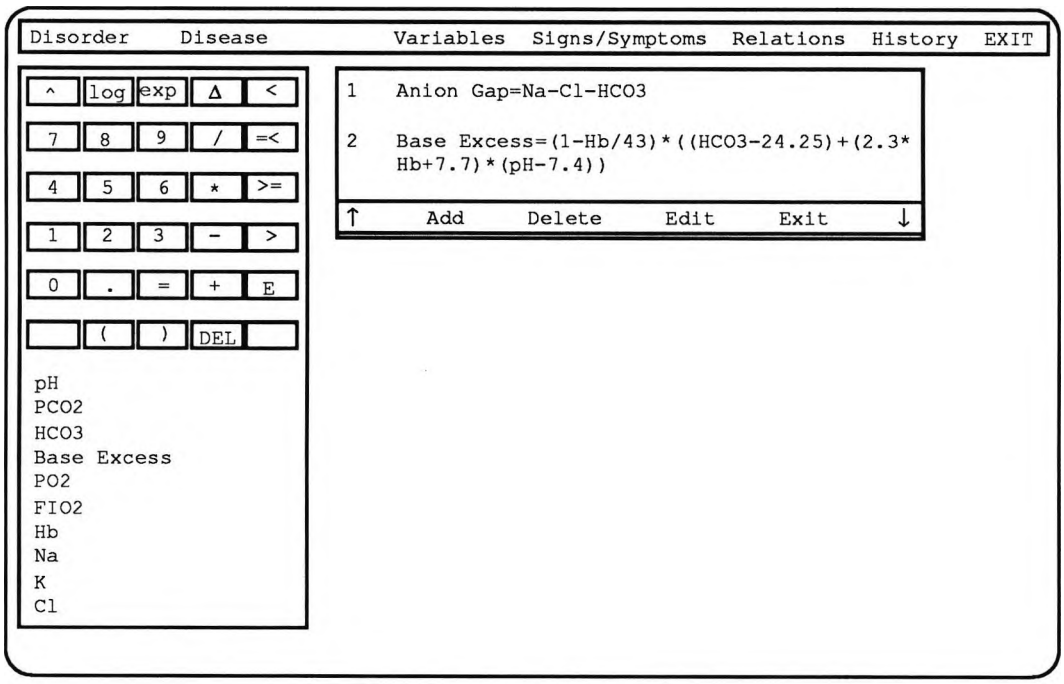
```
variable("PCO2")      data_parameter("PCO2",units,"kPa").
                      data_parameter("PCO2",upper_limit,50.000).
                      data_parameter("PCO2",lower_limit,0.000).
                      data_parameter("PCO2",mean,5.320).
                      data_parameter("PCO2",standard_deviation,0.333).
                      data_parameter("PCO2",default,none).
noun("PCO2",variable) --> [pco2]                    (lexicon entry)
```

(b) PROLOG Representation

Figure 4.16 Primitive Data Objects: Laboratory Data

The mathematical relationships that exist between laboratory data variables can be defined by selecting *relations* in the main command line. The relationships are used to derive data values and consist of an equation whose left hand side is a single data variable and whose right hand side is an expression involving one or more other variables. The relationships are edited by clicking the mouse over appropriate fields in a keypad of digits, operators and a list of data variables as shown in Figure 4.17a. Figure 4.17b shows the way in which relationships are represented in PROLOG.

Selection of the *signs/symptoms* field, produces a drop-down menu display of all signs and symptoms in the knowledge base. New signs or symptoms can be added, existing ones deleted and the attributes of each one can be edited. When a new sign/symptom is added, it is assigned a single attribute, unknown. Further attributes can be added by selecting the edit option in the drop- down menu and clicking the mouse over the appropriate sign/symptom. Figure 4.18a shows the editing of attributes for the symptom *vasodilatation* and the representation in PROLOG is given in Figure 4.18b.

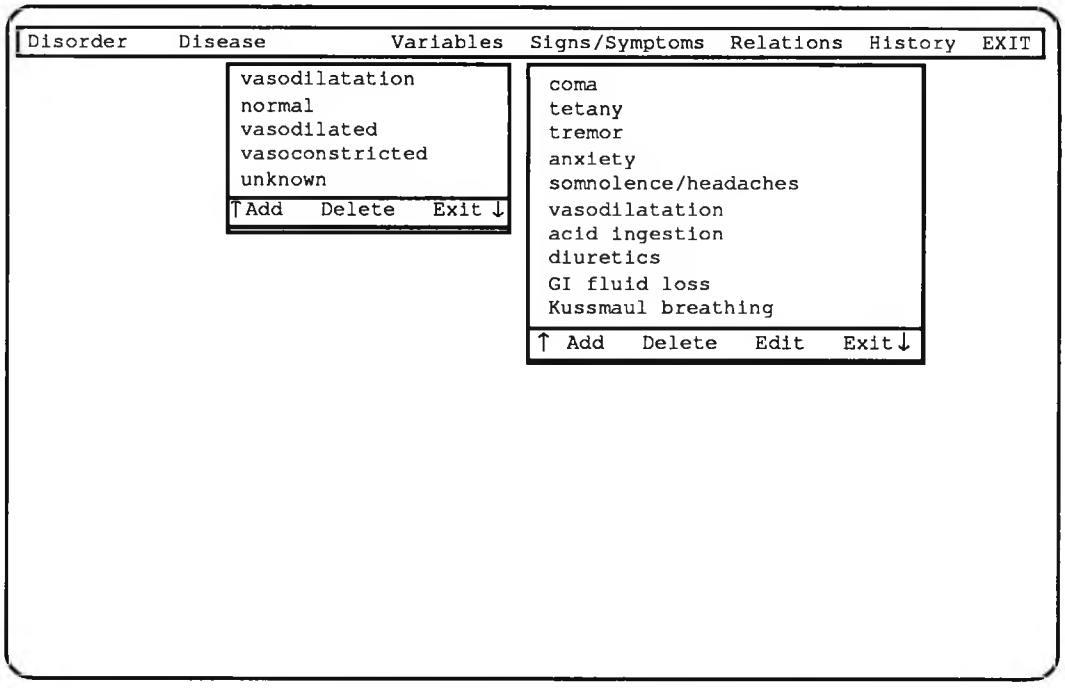


(a) FRAMEBUILDER Display

```
relation("1",["Anion Gap","=","Na","-","Cl","-","HCO3"]).

relation("2",["Base Excess","=","(","1","-","Hb","/","4","3",")","*",
          "(","(","HCO3","-","2","4",".","2","5",")","+","(","2",".","3",
          "*","Hb","+","7",".","7",")","*","(","pH","-","7",".","4",")",")"]).
```

(b) PROLOG Representation

Figure 4.17 Primitive Data Objects: Data Derivation Relationships



(a) FRAMEBUILDER Display

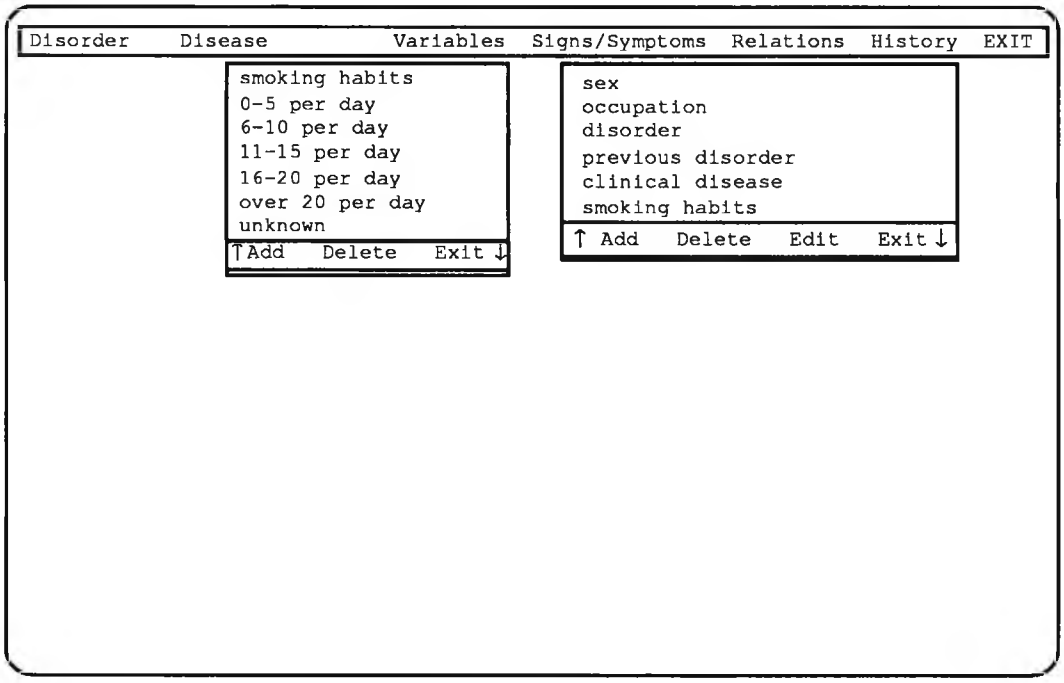
```
symptom("coma",["unknown","absent","present","unknown"]).
symptom("tetany",["unknown","absent","present","unknown"]).
symptom("tremor",["unknown","absent","present","unknown"]).
symptom("anxiety",["unknown","absent","present","unknown"]).
symptom("somnolence/headaches",["unknown","absent","present","unknown"]).
symptom("Kussmaul breathing",["unknown","absent","present","unknown"]).
symptom("shock",["unknown","absent","present","unknown"]).
symptom("vasodilatation",["unknown","normal","vasodilated","vasoconstricted","unknown"]).
symptom("acid ingestion",["unknown","absent","present","unknown"]).
symptom("dieuretics",["unknown","absent","present","unknown"]).
 noun("vasodilated",attribute(symptom,"vasodilatation"))-->[vasodilated].   (lexicon entry)
```

(b) PROLOG Representation

Figure 4.18 Primitive Data Objects: Signs and Symptoms

The definition of items of patient history is achieved in exactly the same way as signs/symptoms, after clicking the mouse over the *History* field. However, as Figure 4.19 shows, there are three system-defined items of history; *current disorder*, *previous disorder* and *clinical diagnosis*. The attributes of these special objects cannot be edited in the same way as other items of history - their definition and use is described in Section 4.4.5

The two fields on the left of the main command line are used to define classes of physiological disorders and clinical diseases, which provide the link between the primitive data object level and the second level of frame hierarchies. Each disorder or disease defined as a primitive data object forms the root of a hierarchy of the type described in Section 4.3.3.



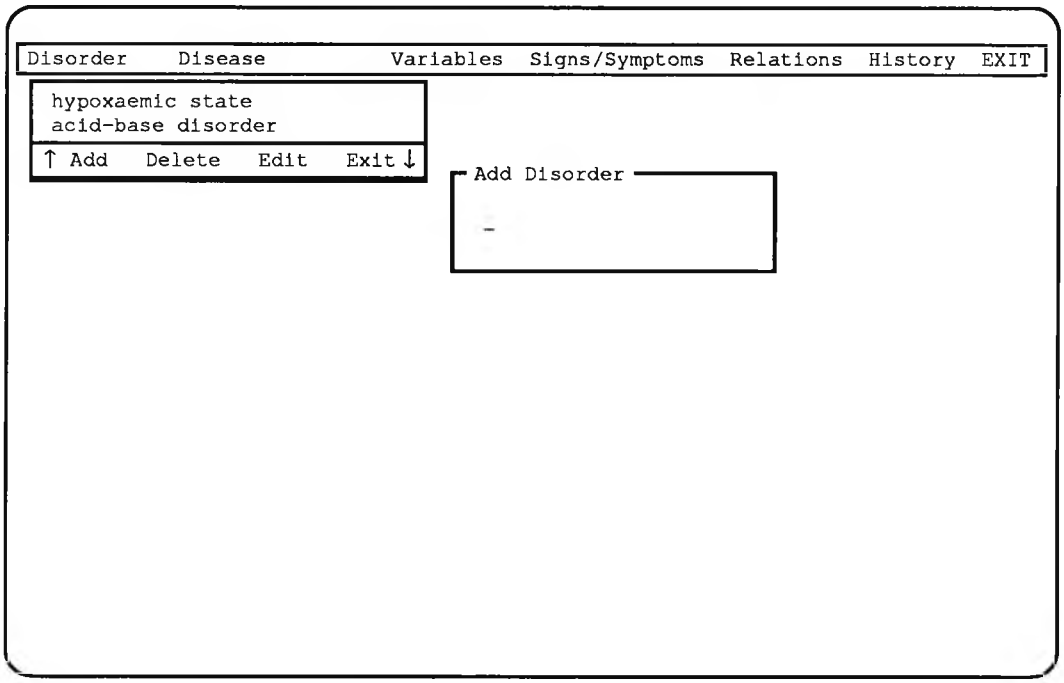
(a) FRAMEBUILDER Display

```
history("sex",["unknown","female","male","unknown"]).
history("occupation",["unknown","asbestos fitter","unknown"]).
history("disorder",["unknown","unknown"]).
history("previous disorder",["unknown","unknown"]).
history("clinical disease",["unknown","unknown"]).
history("smoking habits",["unknown,"0-5 per day","6-10 per day",
        "11-15 per day","16-20 per day","over 20 per day","unknown"]).

noun("0-5 per day",attribute(history,"smoking habits")-->[0-5,per,day].   (lexicon entry)
```

(b) PROLOG Representation

Figure 4.19 Primitive Data Objects: Patient History

Figure 4.20a shows the drop-down menu of disorders and the way in which a new disorder is added. The representation of disorders and diseases forming the roots of hierarchies is shown in Figure 4.20b. By selecting *edit* in the drop-down menu of disorders or diseases and then clicking over one of the entries, the hypothesis hierarchy for that entry can be edited. This process is described in the next section.



(a) FRAMEBUILDER Display

```
disorder("hypoxaemic state").

disorder("acid-base disorder").
```

(b) PROLOG Representation

Figure 4.20 Primitive Data Objects: Disorder Classes

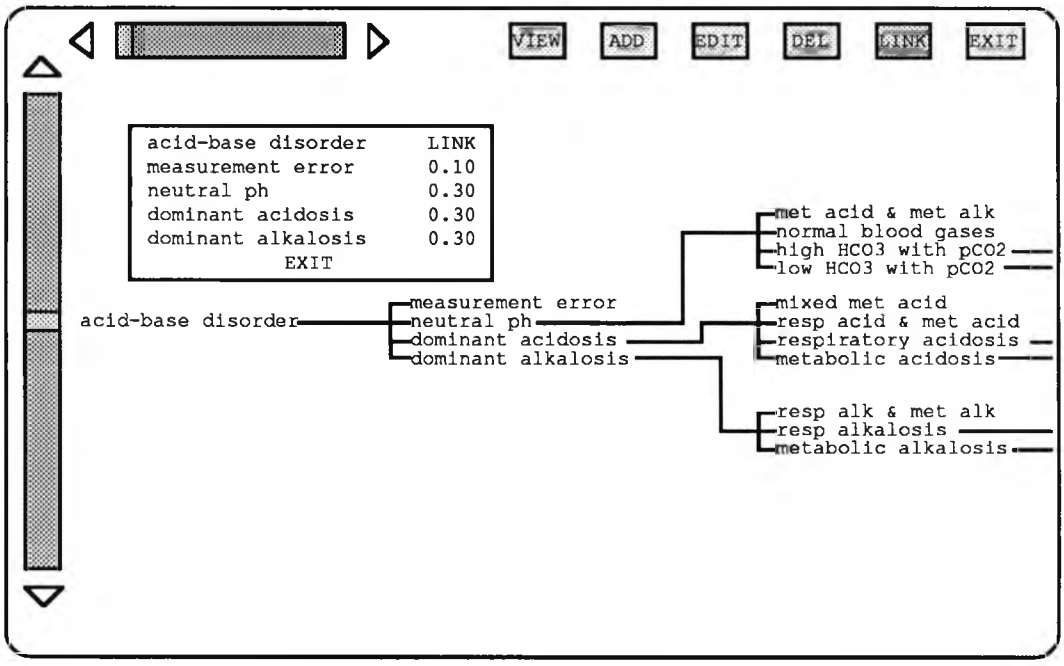
4.4.4 Representation of a Structured Hypothesis Space

The second level of FRAMEBUILDER is a graphical representation of the hypothesis hierarchies defined in Section 4.3.3. The display seen on entry to the hierarchy for acid-base disorders is shown in Figure 4.21a.

The user can scroll around the hierarchy (up-down and left-right) by clicking on the arrows at the left or top edges of the display and the editing function can be set by clicking over one of the fields at the top right. Nodes can be added to the hierarchy in two ways. To add a new node as a descendent to an existing leaf node, the function *link* is selected and the mouse clicked over the desired leaf node; the name of the new node can then be input and it is added to the hierarchy, linked to the former leaf node. The *add* function can be selected to add an additional descendent to a non-leaf node; the mouse is clicked over one of the existing descendents and the name of the new node is then input. Nodes can be removed by selecting *delete* and clicking the mouse over the appropriate node. If the *link* function is

selected and the mouse is clicked over a non-leaf node, the weights on the links to its descendent nodes can be assigned - these weights are used to calculate the *a priori* probabilities of each hypothesis in the hierarchy. The PROLOG representation of the hierarchy in Figure 4.21a, including the weights on the links, is shown in Figure 4.21b.

The third level of the FRAMEBUILDER environment is entered by selecting the *view* function and clicking over one of the nodes in the hierarchy; the knowledge frame for that node can then be viewed and edited.



(a) FRAMEBUILDER Display

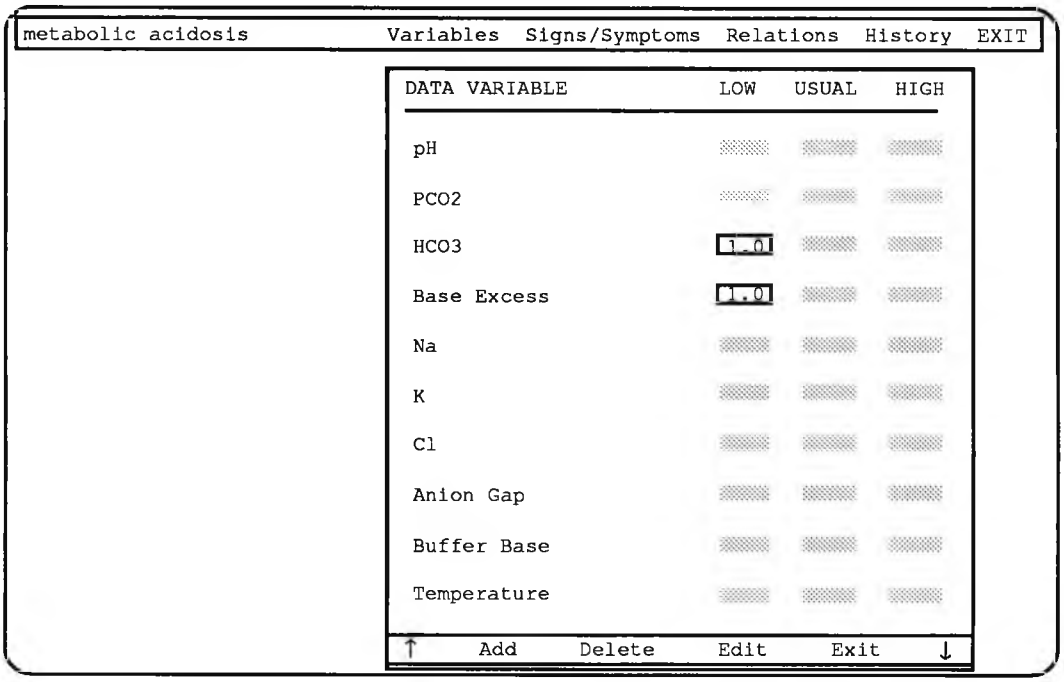
```
disorder("acid-base disorder","acid-base disorder",["measurement error","neutral ph",
                                        "dominant acidosis","dominant alkalosis"]).
disorder("acid-base disorder","neutral ph",["met acid & met alk","normal bllod gases",
                                        "high HCO3 with pCO2","low HCO3 with pCO2"]).
disorder("acid-base disorder","dominant acidosis",["mixed met acid","resp acid & met acid",
                                        "respiratory acidosis","metabolic acidosis"]).
disorder("acid-base disorder","dominant alkalosis",["resp alk & met alk","resp alkalosis",
                                        "metabolic alkalosis"]).

link("acid-base disorder","acid-base disorder","measurement error",0.1).
link("acid-base disorder","acid-base disorder","neutral pH",0.3).
link("acid-base disorder","acid-base disorder","dominant acidosis",0.3).
link("acid-base disorder","acid-base disorder","dominant alkalosis",0.3).
```

(b) PROLOG Representation

Figure 4.21 The Frame Hierarchy

4.4.5 Frame Representation

The command line for a knowledge frame mirrors the main command line, allowing the selection of variables, signs/symptoms, relations and history for that frame. Figure 4.22a shows the drop-down menu displayed after the selection of *variables* in the frame for metabolic acidosis. The variables displayed are those that have a mean and standard deviation defined as attributes and the probabilities of observing low, usual or high levels are set by clicking over the appropriate field. Checks at this stage ensure that the same data variable does not appear in two related frames (for instance the level of Base Excess cannot be specified for uncompensated metabolic acidosis if it already appears in the frame for metabolic acidosis). Checks also ensure that $\Sigma P(Level|H_i)=1$ for any specified variable. The PROLOG representation of laboratory data levels in the frame for metabolic acidosis is shown in Figure 4.22b.



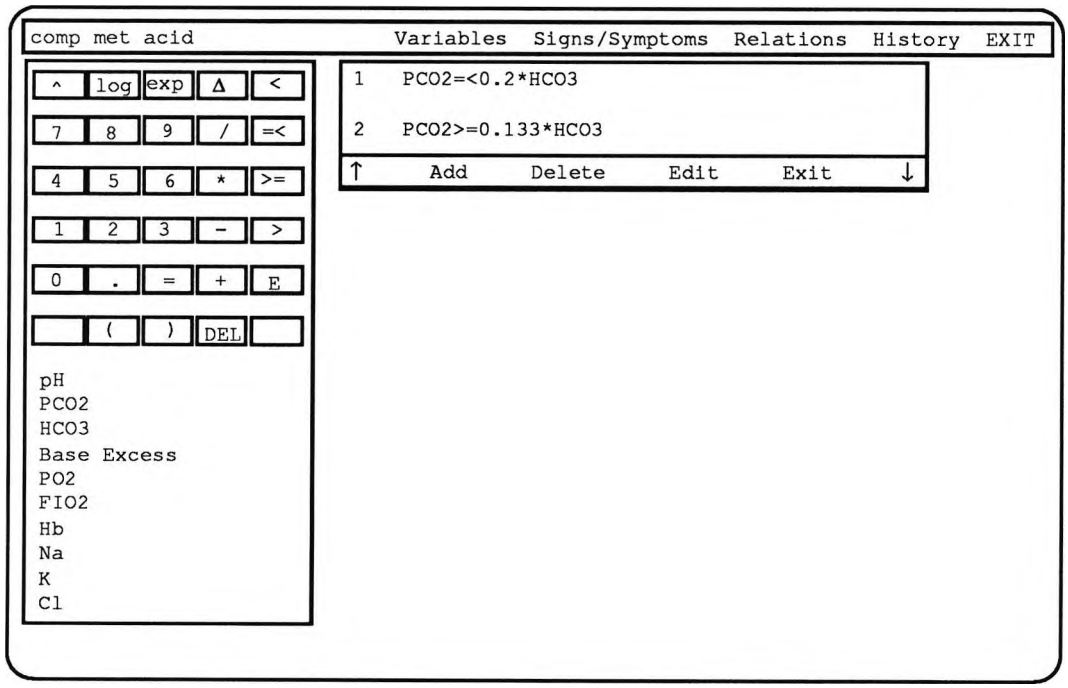
(a) FRAMEBUILDER Display

```
frame_variable("metabolic acidosis","acid-base disorder","HCO3",low,1.000).
frame_variable("metabolic acidosis","acid-base disorder","Base Excess",low,1.000).
```

(b) PROLOG Representation

Figure 4.22 Frame Instances: Laboratory Data

Relationships between data variables that are characteristic of a hypothesis are built up using a keypad, in the same way as the primitive data relationships. However, the left and right hand sides of relationships for frames can both be expressions beween arbitrary numbers of variables and they can be related by any one of {=,<,=<,>,>=}. The Δ symbol on the keypad can be used to denote changes in the value of a variable over time. For instance, the relationship shown in Figure 4.23 is between the change in pCO_2 and the change in HCO_3^-



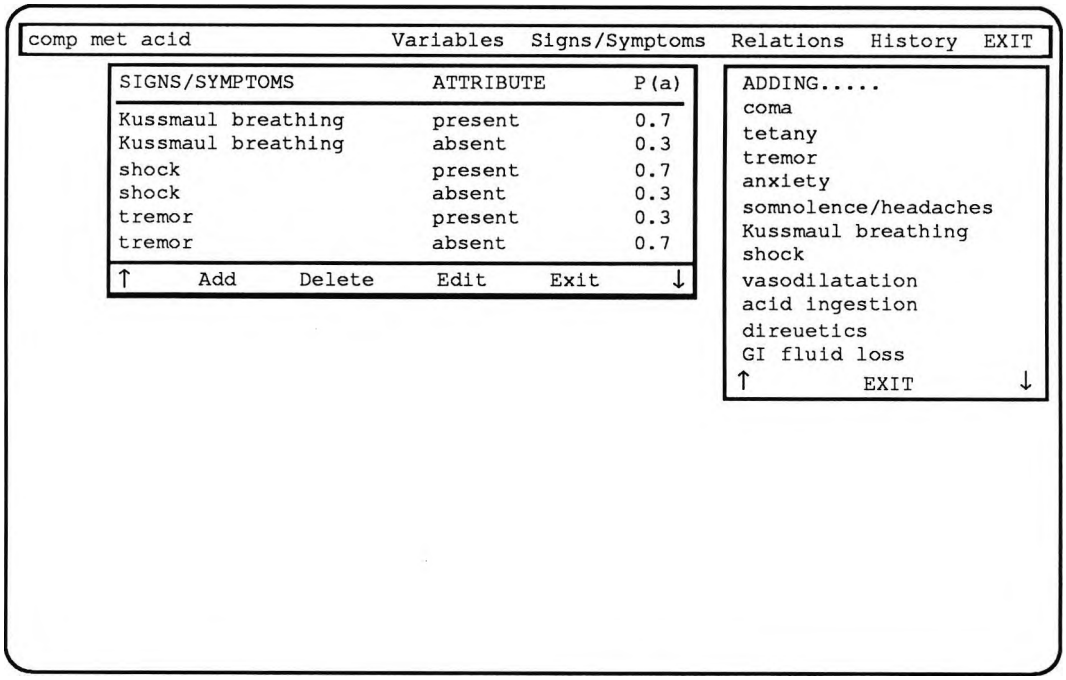
(a) FRAMEBUILDER Display

```
frame_relation("comp met acid","acid-base disorder","1",["PCO2","=<",
                                                          "0",".","2","*","HCO3"]).

frame_relation("comp met acid","acid-base disorder","2",["PCO2",">=",
                                                         "0",".","1","3","3","*","HCO3"]).
```

(b) PROLOG Representation

Figure 4.23 Frame Instances: Data Relationships

Figure 4.24a shows the drop-down menu for the specification of the signs and symptoms of a frame. The signs and symptoms are added by selecting with the mouse from the set defined at the primitive data object level. A newly added sign/symptom is displayed with the attribute unknown and probability 1 (the probability is the conditional probability of the sign/symptom given the frame hypothesis). The same sign or symptom can appear more than once in a frame with different attributes. Checks are performed to ensure that the same sign/symptom is not specified in two related frames and that for any sign/symptom, S, $\Sigma P(S|H_i)=1$, where the sum is over the attributes specified for S in the frame. Figure 4.24b shows the PROLOG representation of the signs and symptoms of a frame.



(a) FRAMEBUILDER Display

```
frame_symptom("comp met acid","acid-base disorder","Kussmaul breathing","present",o.7).
frame_symptom("comp met acid","acid-base disorder","Kussmaul breathing","absent",0.3).
frame_sympotm("comp met acid","acid-base disorder","shock","present",0.7).
frame_symptom("comp met acid","acid-base disorder","shock","absent",0.3).
frame_symptom("comp met acid","acid-base disorder","tremor","present",0.3).
frame_symptom("comp met acid","acid-base disorder","tremor","absnet",0.7).
```

(b) PROLOG Representation

Figure 4.24 Frame Instances: Signs and Symptoms

Finally, the history slots for a frame can be specified in exactly the same way as signs and symptoms (Figure 4.25a). As was mentioned in Section 4.4.2, there are three special history facets defined by the system: *disorder, previous disorder* and *clinical disease.* The disorder facet allows a hypothesis from any other disorder class (*ie* from any class other than the class of the current frame) to be specified in the history slot of the frame. This provides a mechanism for linking the hypothesis hierarchies so that the confirmation of one disorder can act as evidence for the confirmation/disconfirmation of a disorder in another class. It is also the mechanism by which the disorders expected with clinical diseases are specified in frames of disease hierarchies (these are used in the critiquing process of the diagnostic system).

```
comp met acid                Variables  Signs/Symptoms  Relations  History  EXIT

ADDING.....                  HISTORY                ATTRIBUTE        P(h)
sex
occupation                   metabolic acidosis     was present      0.3
disorder                     ↑     Add     Delete     Edit     Exit     ↓
previous disorder
clinical diagnosis
↑          EXIT        ↓
```

(a) FRAMEBUILDER Display

```
frame_history("comp met acid","acid-base disorder",
                        "metabolic acidosis","was present",0.3).
```

(b) PROLOG Representation

Figure 4.25 Frame Instances: Patient History

The *previous disorder* facet allows the specification of the temporal progression of disorders in a class. For instance, the fact that a compensated metabolic acidosis is likely to be preceded by an uncompensated or partially compensated metabolic acidosis is captured by specifying the latter two disorders as previous disorder facets in the frame for compensated metabolic acidosis.

Nodes from the hierarchies of diseases can be specified as characteristics of disorders by clinical disease facets in disorder frames. If the current disease state of the patient is to be critiqued in the light of the disorders diagnosed, then diseases should not be specified as evidence in the disorder frames (since the critique would then be redundant). Hence it would not normally be necessary to use the clinical disease facet in disorder frames - there are, however, occasions on which this can be useful, as will be demonstrated in Chapter 7.

4.4.6 Summary

FRAMEBUILDER allows a knowledge base of PROLOG clauses to be created, edited and browsed through a graphical interface, that requires no knowledge of computer programming. In this respect it provides an excellent means by which clinicians can develop and maintain a knowledge-based system without relying on an intermediate knowledge engineer. Observations, diagnostic hypotheses and associations between observations and hypotheses are organized in frame-like structures. A limited knowledge of temporal dependencies can be captured by the use of the Δ symbol in data relationships and the previous disorder facet in hypothesis frames.

As a knowledge acquisition tool, FRAMEBUILDER falls into the category of domain independent knowledge editing environments described in Section 3.9.3.2, although its three-level structure contains implicit knowledge of the knowledge acquisition technique of *distinguishing the goals* described in Section 3.9.2.2. The use of FRAMEBUILDER for knowledge acquisition is described in the next section.

4.5 Knowledge Acquisition Using FRAMEBUILDER

FRAMEBUILDER was used in knowledge elicitation sessions with a clinical biochemist at the West Middlesex Hospital. In the first session, the knowledge engineer demonstrated the use of the knowledge editing environment which had been loaded with some of the knowledge from the first phase of acquisition. This included the hypothesis hierarchy and the relationships for data derivation shown in Table 4.1.

The expert first constructed the new hypothesis hierarchy shown in Figure 4.26, which can be compared with the hierarchy in Figure 4.5. He preferred to make an initial sketch by hand - this was then transferred to FRAMEBUILDER for minor alterations and additions. The node for *normal blood gases* was included in order to make the hypotheses exhaustive for the class of acid-base disorders, and the node for *measurement error* was defined in the manner suggested by the MUNIN system (see Section 2.4.2). The measurement error frame contains all possible observations, with equal probability assignments, so that a measurement error will be diagnosed if all the other hypotheses have been ruled out.

Five items of laboratory data - pH, pCO2, HCO_3^-, Base Excess, Anion Gap - were specified as evidence for disorder hypotheses across the entire hierarchy. For the first four of these data, the low, normal or high level expected with a particular disorder were assigned categorically (*ie* P(level|disorder)=1). For Anion Gap, some disorders were considered to have a normal level (P(normal|disorder)=1) and others were considered as having either a normal or high level (assigned as P(normal|disorder)=0.5, P(high|disorder)=0.5).

Van Slyke Equation: Base Excess=(1-Hb/43).((HCO3⁻-24.25)+(2.3.Hb+7.7).(pH-7.4))

Henderson-Hasselbalch Equation: $HCO_3^-=0.23.pCO_2.10^{(pH-6.1)}$

Buffer Base $=Na^+ + K^+ - Cl^-$

Anion Gap$=Na^+ + K^+ - Cl^- - HCO_3^-$

Table 4.1 Relationships For Data Derivation

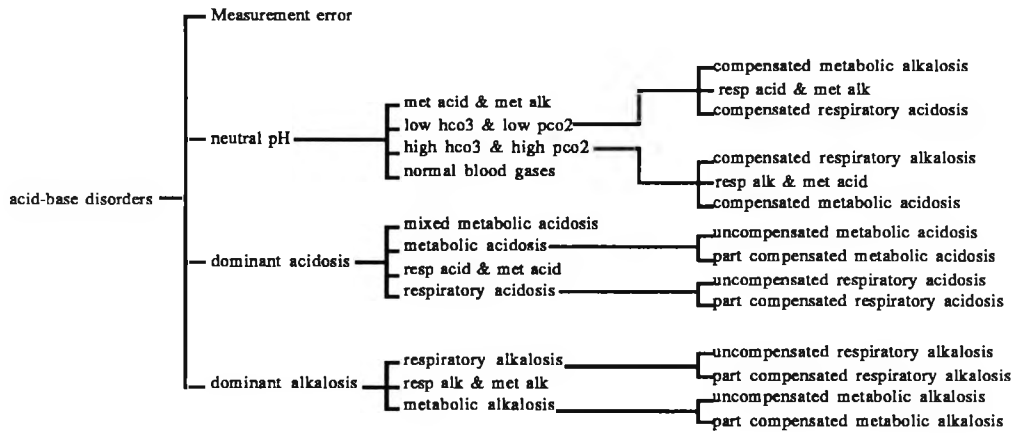


Figure 4.26 The Revised Hypothesis Hierarchy for Acid-Base Disorders

In the second session, some hypothetical cases were run, using the knowledge base constructed in the first session. It was found that the basic model was working correctly, but diagnoses were often reported at a general level, not in terms of specific leaf node hypotheses. For example, the diagnosis *high HCO3 with pCO2* would be reported rather than one of the leaf nodes *comp met alk, resp acid & met alk* or *comp resp alk.* The problem, therefore, was to find evidence that would discriminate between the members of these subsets of leaf nodes. This was achieved by specifying electrolyte levels (Na^+, K^+, Cl^-) and the expected degree of compensation for simple metabolic and respiratory disorders, indicated by the expressions in Table 4.2.

Compensated Metabolic Acidosis	$\Delta pCO_2 > 0.2.\Delta HCO_3^-$
	$\Delta pCO_2 < 0.6.\Delta HCO_3^-$
Compensated Metabolic Alkalosis	$\Delta pCO_2 > 0.033.\Delta HCO_3^-$
	$\Delta pCO_2 < 0.133.\Delta HCO_3^-$
Compensated Respiratory Acidosis	$\Delta HCO_3^- > 3.\Delta pCO_2 - 4$
	$\Delta HCO_3^- < 3.\Delta pCO_2 + 4$
Compensated Respiratory Alkalosis	$\Delta HCO_3^- > 3.75.\Delta pCO_2 - 4$
	$\Delta HCO_3^- < 3.75.\Delta pCO_2 + 4$

Table 4.2 Compensation Limits (From Schreck *et al*, 1986)

	Compensated Respiratory Alkalosis	Respiratory Alkalosis & Metabolic Acidosis	Compensated Metabolic Acidosis	Compensated Respiratory Acidosis	Respiratory Acidosis & Metabolic Alkalosis	Compensated Metabolic Alkalosis
coma	x	√	√	√	√	x
tetany	√	√	x	x	√	√
anxiety	x	√	√	√	√	x
somnolence/headache	x	x	x	√	√	x
vasodilatation	normal	vasoconstriction	vasoconstriction	vasodilation	vasodilation	normal
acid ingestion	x	√	√	x	x	x
diuretics	x	x	x	x	√	√
GI fluid loss	x	√	√	x	x	x
Kussmaul breathing	x	√	√	x	x	x
shock	x	√	√	x	x	x
pregnancy	√	√	x	x	x	x

Table 4.3 Signs and Symptoms

Clinical signs and symptoms were also used for discrimination between the leaf nodes. A list of some 50 useful signs and symptoms was used as a basis for knowledge elicitation at the Royal Free Hospital, with a consultant anaesthetist in the ICU. From the list of 50, only those shown in Table 4.3 were considered to be of sufficient predictive power to warrent inclusion in the knowledge base. The expert was reluctant to estimate probability assignments for the symptoms and observed that although the presence of a symptom might suggest a particular disorder (or disorders), the actual probability of the symptom given the disorder was very low. In effect he was able to estimate P(disorder|symptom) but not P(symptom|disorder) - a contradiction of the generally accepted situation (Welbank, 1983).

The signs and symptoms were elicited in the form of the grid in Table 4.3: a tick indicating the cases in which a disorder is suggested by the presence of the sign or symptom. The grid was translated into FRAMEBUILDER entries by assigning probabilities according to Table 4.4. The assignments were intended not to be realistic estimates, but to capture the fact that the observation of these signs and symptoms increases belief in certain hypotheses although their absence does not completely rule out those same hypotheses; the probabilistic inference mechanism is being used to implement an *ad hoc* scoring of hypotheses.

Symptom indicates hypothesis	Symptom does not indicate hypothesis
P(symptom is present)=0.8	P(symptom is present)=0.2
P(symptom is absent)=0.2	P(symptom is absent)=0.8

Table 4.4 Probability Assignments for Signs and Symptoms

4.6 Summary

The design of a knowledge-based system for the diagnosis of physiological disorders, based on laboratory data and clinical observations, has been described. The control of the diagnostic process is achieved by a blackboard architecture, allowing for independent development of knowledge sources and providing the framework for an opportunistic problem solving process which can easily be interrupted and resumed as necessary. Two important knowledge sources handle the classification of data and the impact of observations on diagnostic hypotheses. It has been shown that a Bayesian updating of belief in a hierarchy of hypotheses can be achieved with a knowledge base consisting only of estimates of the frequency of observations in given disorder states and implicity captures the behaviour of systems requiring a wider range of parameter estimates. A detailed description of the implementation of the design is given in the next chapter.

To complement the diagnostic system, a knowledge editing environment has been developed. This allows the entire knowledge base to be browsed and edited by a purely graphical interaction and is suitable for the direct development of knowledge-based systems by clinicians. Knowledge acquisition sessions have been described for the developement of a system for the interpretation of blood gas analysis results, in which the FRAMEBUILDER environment was used as a tool for knowledge base construction. It has been shown that the probabilistic framework can be used not only for well defined estimates (and ultimately measurements) of probabilities but also for more *ad hoc* definitions of symptom-disorder associations.

CHAPTER FIVE

IMPLEMENTATION OF A KNOWLEDGE-BASED SYSTEM

5.1 Introduction

The practical implementation of the blackboard diagnostic module (in PROLOG2) differs slightly from the conceptual design presented in Section 4.3. The hypothesis level on the physiological diagnosis panel is split into two sections: the *sub-hypothesis* level contains the hypotheses generated by individual pieces of evidence, the *hypothesis* level contains the accumulated hypothesis for each disorder based on all the evidence. The *sum_hypothesis* knowledge source performs the transformation from the sub-hypothesis level, where there may be more than one entry for each disorder, to the hypothesis level where there is at most one entry for each disorder.

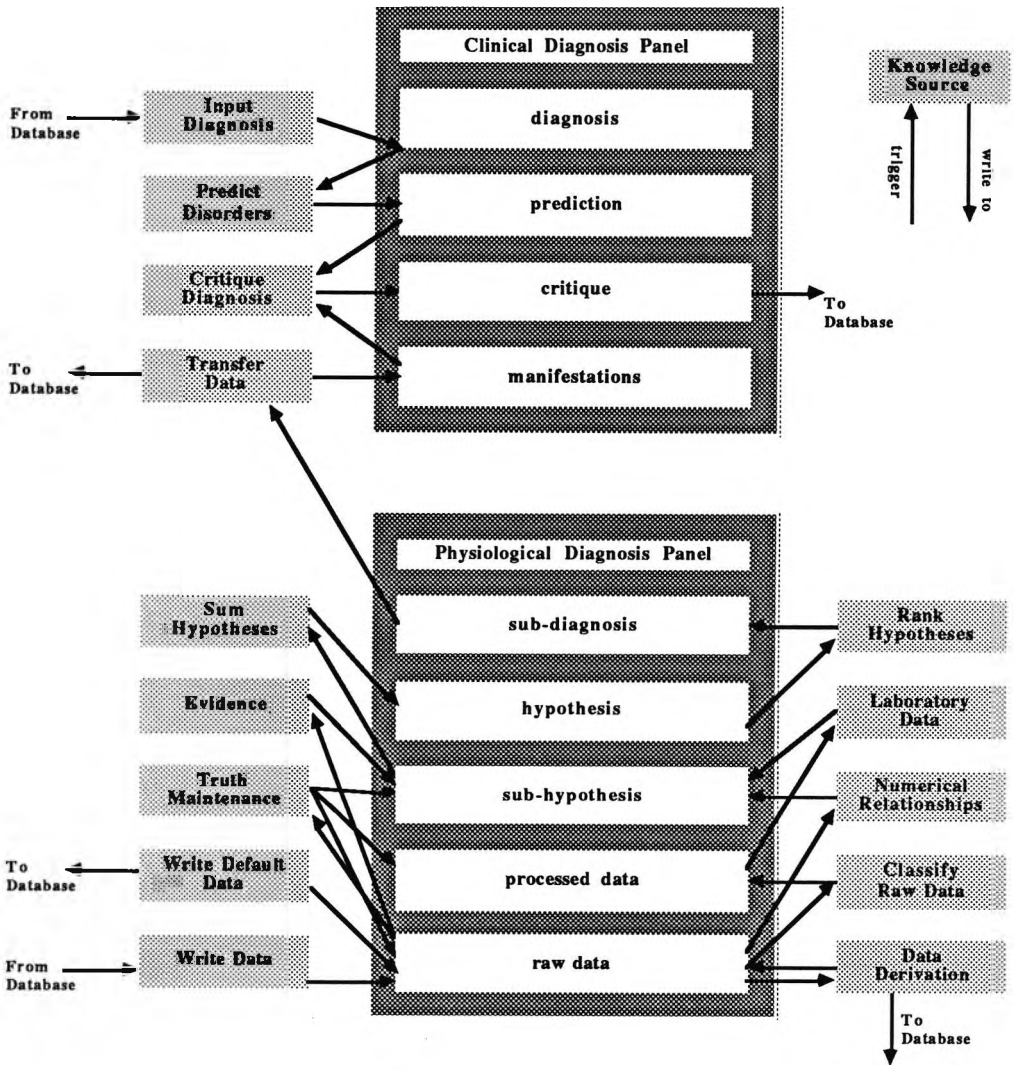


Figure 5.1 The Implemented Design of the Blackboard Diagnostic Module

The single knowledge source write_data shown in Figure 4.9 has been split into the two knowledge sources write_raw_data and write_default_data. Evidence from signs/symptoms and patient history is processed in exactly the same manner; hence the two knowledge sources in Figure 4.9, have been combined into a single *evidence* knowledge source. The diagnosis level on the physiological panel has been renamed *sub-diagnosis* in order to distinguish it from the diagnosis level on the clinical diagnosis panel. The final structure of the blackboard diagnostic module, as it has been implemented, is shown in Figure 5.1.

The next section describes the way in which the blackboard architecture has been realized in PROLOG and the following four sections deal with the knowledge sources in more detail. Finally, the dialogue interaction module is described in Section 5.7.

```
blackboard(Blackboard,diagnosis,Root,Disease).
blackboard(disorder,diagnosis,"Impaired Lung Motion","Pleural Effusion").

blackboard(Blackboard,prediction,Disease,Disorder).
blackboard(disorder,prediction,"Pleural Effusion","comp resp acid").

blackboard(Blackboard,critique,Disorder_Root,Disease,Disorder,Critique_type).
blackboard(disorder,critique,"acid-base disorder","Pleural Effusion","comp resp acid",expected).

blackboard(Blackboard,manifestations,Root,Disorder).
blackboard(disorder,manifestations,"acid-base disorder","comp met acid").
blackboard(disorder,manifestations,"acid-base disorder","part comp met acid").
```

```
blackboard(Blackboard,sub_diagnosis,Root,Disorder,Belief).
blackboard(disorder,sub_diagnosis,"acid-base disorder","comp met acid",0.500).
blackboard(disorder,sub_diagnosis,"acid-base disorder","part comp met acid",0.500).

blackboard(Blackboard,hypothesis,Root,Hypothesis,Belief).
blackboard(disorder,hypothesis,"acid-base disorder","comp met acid",1.525).
blackboard(disorder,hypothesis,"acid-base disorder","part comp met acid",1.525).

blackboard(Blackboard,Hypothesis_type,Root,Hypothesis,Evidence,Belief).
blackboard(disorder,variable_hypothesis,"acid-base disorder","dominant acidosis","pH",1.024).
blackboard(disorder,relation_hypothesis,"acid-base disorder","comp met acid",
                                                [change("HCO3"),change("PCO2")],1.007).
blackboard(disorder,evidence_hypothesis,"acid-base disorder","comp met acid","coma",1.115).

blackboard(Blackboard,classified_data,Data_name,Classification,Probability).
blackboard(disorder,classified_data,"pH",high,0.000).
blackboard(disorder,classified_data,"pH",usual,0.655).
blackboard(disorder,classified_data,"pH",low,0.355).

blackboard(Blackboard,raw_data,Data_type,Data_name,Value,Status).
blackboard(disorder,raw_data,variable,"pH",7.350,measurement).
blackboard(disorder,raw_data,variable,"Base Excess",-2.000,derivation).
blackboard(disorder,raw_data,variable,"FIO2",21.000,default).
blackboard(disorder,raw_data,symptom,"coma","present",measurement).
```

Figure 5.2 PROLOG Representation of Blackboard Data Entries

5.2 Blackboard Representation and Control

Blackboard entries are represented as PROLOG terms with the functor *blackboard* and 4 to 6 arguments. Figure 5.2 shows the representation of entries at each level. The first two arguments in each term specify the name of the blackboard and the level of the entry. The entire blackboard data structure can be copied simply by changing the first argument in each term; this is a very useful facility since the state of a problem solution is completely determined by the data on the blackboard and a list of executed knowledge sources.

Each knowledge source is represented as a pair of PROLOG clauses; the first defines the triggering conditions, the second defines the actions to be performed when the knowledge source is executed. In fact this second clause is normally split into a sequence of action sub-goals and comprises between about 5 and 50 lines of PROLOG code. Figure 5.3 shows the PROLOG representation of a knowledge source with the simple example of write_raw_data.

TRIGGER CLAUSE FORMAT

```
trigger(BLACKBOARD,KNOWLEDGE_SOURCE,CONDITIONS,ACTIVATION):-
check for activation,
condition 1,
condition 2,
    .
    .
condition n,
instantiate conditions,
check for previous execution.
```

EXAMPLE

```
trigger(Blackboard,write_raw_data,Condition_list,Active_ks):-
actice_ks([write_raw_data,data_handler],Active_ks),
current_data(Type,Facet,Value),
hypothesis_type(Type,_),
Condition_list=[Type,Facet,Value],
not blackboard(Blackboard,raw_data,Type,Facet,Value,S).
```

KNOWLEDGE SOURCE CLAUSE FORMAT

```
Knowledge_source(BLACKBOARD,KNOWLEDGE_SOURCE,CONDITIONS,ACTIVATION):-
goal 1,
goal 2,
    .
    .
goal n,
record execution of knowledge source.
```

EXAMPLE

```
knowledge_source(Blackboard,write_raw_data,[Type,Facet,Value],
                        [write_raw_data,data_handler,evidence_handler]):-
entry_display(Variable,Value,"",Entry_display),
window(blackboard4,text(Entry_display)),
retractall(blackboard(Blackboard,raw_data,Type,Facet,_,_)),
asserta(blackboard(Blackboard(raw_data,Type,Facet,Value,measurement)),
retractall(executed_ks(Blackboard,write_raw_data,[Type,Facet,_])),
retractall(executed_ks(Blackboard,derive_data,[Type,Facet,Value])),
retarctall(executed_ks(Blackboard,write_default_data,[Type,Facet,Value])),
assert(executed_ks(Blackboard,write_raw_data,[Type,Facet,Value])).
```

Figure 5.3 PROLOG Repesentation of Knowledge Sources

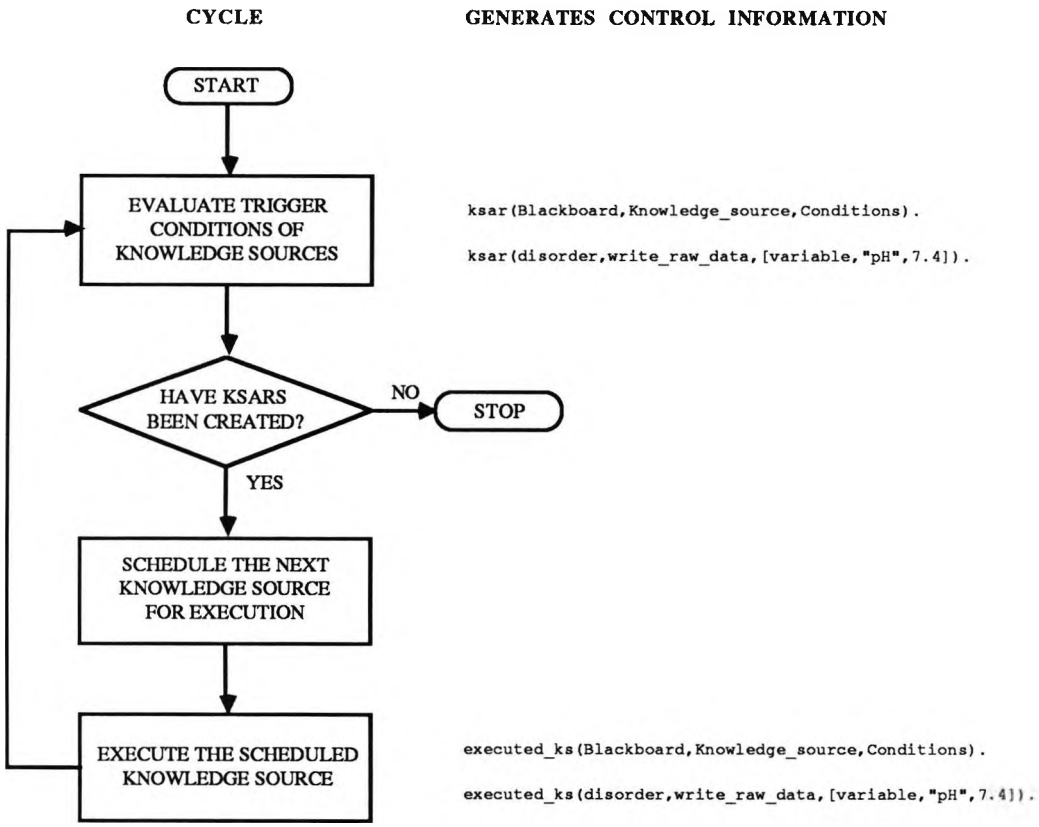


Figure 5.4 The Control Cycle

The control cycle of the blackboard system is shown in Figure 5.4. The first step in the cycle is to evaluate the trigger conditions for each knowledge source in order to generate a set of Knowledge Source Activation Records (KSARs). This process is made faster by passing to the trigger clauses a list of knowledge sources activated by the last executed knowledge source (stored as the Activation argument in the knowledge_source clause). The first test performed by each trigger clause is a check on the Activation list to see if the knowledge source is currently active; if it is not, the trigger fails immediately. Each successfully evaluated trigger clause generates a list of instantiated variables as its Conditions argument which is stored in the KSAR generated.

When all possible KSARs have been generated (there may be more than one, with differently instantiated conditions, for each knowledge source trigger) the *scheduler* selects one for execution. The corresponding knowledge_source clause is then evaluated using the instantiated condition list and the KSAR is recorded as an executed_ks (this prevents the same KSAR from being regenerated by the trigger clauses). Finally, the unexecuted KSARs are deleted and the cycle begins again; it terminates when no KSARs are generated in the first step.

The speed of the system execution depends on the strategy employed by the KSAR scheduler; three strategies have been implemented so that their effect on the efficiency of the system can be compared (see Section 6.4.5). The *full_schedule* strategy scans all the KSARs generated and selects the one triggered at the lowest (or highest) blackboard level in order to implement a bottom-up (or top-down) strategy. This could be considered as the simplest form of a sophisticated scheduler (*ie* one that makes a scheduling decision based on the set of all possible KSARs on each cycle).

A sophisticated scheduler is slow because it requires many KSARs to be generated on each cycle but executes only one (the others are discarded and many may be regenerated on the next cycle). By ordering the knowledge sources so that those triggered at the lowest blackboard level have their trigger conditions evaluated first, the bottom-up strategy of the full_schedule can be implemented by executing the first generated KSAR. This control strategy has been implemented as the *first_triggered* strategy.

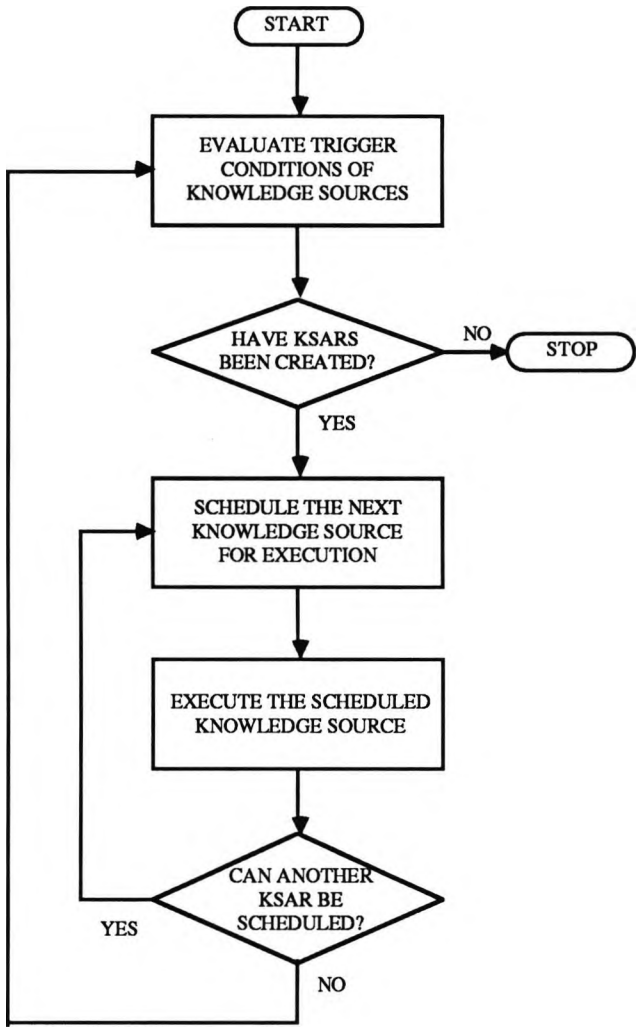


Figure 5.5 Control Cycle with Improved Efficiency

A third control strategy, the *mixed_schedule*, operates using the modified control cycle shown in Figure 5.5. Like the full_schedule strategy this is a sophisticated scheduler working with the set of all possible KSAR generations at each cycle. Just as in the full_schedule strategy, the KSAR triggered at the lowest level is executed; knowledge of the nature of the knowledge sources is then used to execute further KSARs without the need for regeneration. If the execution of a knowledge source modifies the blackboard only at levels higher than that at which it was triggered, any KSARs generated on the current cycle from the same knowledge source will also be generated on the next cycle. Hence it is possible to schedule these KSARs for execution without the need to re-evaluate trigger conditions. Once all such KSARs have been executed, the cycle can begin again with the generation of a new KSAR set.

The following four sections describe the implementation of the knowledge sources themselves and indicate how they are used to create the blackboard data structure shown in Figure 5.2. Table 5.1 provides a summary of the knowledge sources, the blackboard levels at which they are triggered, the levels at which they write entries and the knowledge sources they activate.

KNOWLEDGE SOURCE	TRIGGERED AT LEVELS	WRITES TO LEVELS	ACTIVATES LEVELS
data handlers			
truth_maintenance	database, raw_data	raw_data, classified_data, sub_hypothesis	data_handler, sum_hypothesis
write_raw_data	database	raw_data	write_raw_data, write_default_data, derive_data, classify_data, evidence_handler
derive_data	raw_data	raw_data	write_default_data, derive_data, classify_data, evidence_handler
write_default_data	raw_data	raw_data	write_default_data, derive_data, classify_data, evidence_handler
classify_data	raw_data	classified_data	classified_data,evidence_handler
evidence handlers			
relation evidence	raw_data	sub_hypothesis	evidence_handler, sum_hypothesis, clinical_diagnosis
variable_evidence	classified_data	sub_hypothesis	variable_evidence, evidence, sum_hypothesis, clinical_diagnosis
evidence	raw_data	sub_hypothesis	evidence, sum_hypothesis, clinical_diagnosis
sum_hypotheses	sub_hypothesis	hypothesis	sum_hypothesis, rank_hypothesis
rank_hypotheses	hypothesis	sub_diagnosis	rank_hypothesis, transfer_data, clinical_diagnosis
transfer_data	sub_diagnosis	database, manifestations	data_handler, clinical_diagnosis
clinical diagnosis			
write_disease_diagnosis	database	diagnosis	predict_disorders
predict_disorders	diagnosis	prediction	critique_diagnosis
critique_diagnosis	manifestations, prediction	critique	critique_diagnosis

Table 5.1 Knowledge Sources in the Diagnostic Module

5.3 Patient Data

5.3.1 Data Structures

During a consultation session, the system maintains a patient specific database comprising four data types: variables (laboratory data variables), symptoms (actually signs or symptoms), history and diseases. Additionally, the data can be defined as:

current_data: data currently available for use in diagnosis
archive_data: time stamped data from previous sessions
personal_data: demographic patient data (hospital id, name, sex, age, occupation)

Between sessions, archive data are stored in an individual file for each patient; personal data for all patients are stored in a single file. At the start of a session, the patient's personal and archive data are recovered and certain current data are set. The personal data of age, sex and occupation can all be useful during diagnosis and so these are transferred from personal to current data. Similarly, the archived data about a patient's underlying clinical disease state (which is likely to remain constant between sessions) and previously diagnosed disorders are set as items of history in the current data. The clinical diagnosis is also set as the special disease data type in the current data (this reflects the dual role it can play - either as evidence for the diagnosis of disorders or as the context in which to critque this diagnosis). The PROLOG structures of personal_data, archive_data and current_data are shown in Figure 5.6.

```
current_data(Data_type,Data_name,Value).

current_data(variable,"pH",7.32).
current_data(symptom,"coma","present").
current_data(history,"metabolic acidosis","present").
current_data(disease,"diabetes mellitus","present").


personal_data(Identifier,Data,Value).

personal_data("PT01","age",27).
personal_data("PT01","occupation","asbestos fitter").


archive_data(Date,Time,Data_type,Data_name,Value).

archive_data(050889,1205,variable,"pH",7.32).
archive_data(050889,1205,history,"metabolic acidosis","present").
```

Figure 5.6 PROLOG Representation of Patient Data

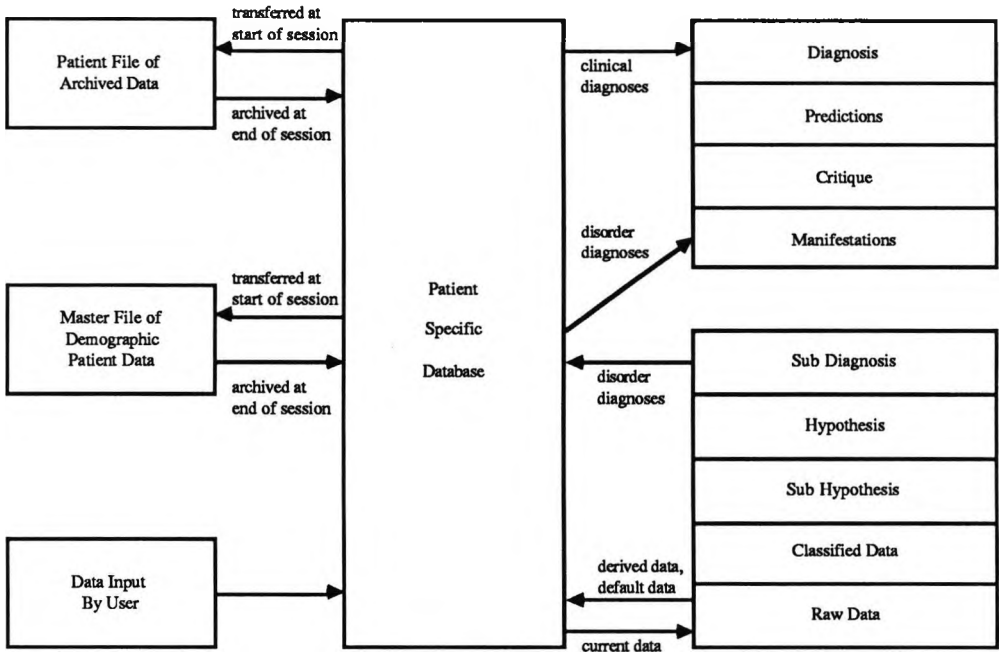


Figure 5.7 Transfer of Data Between Files, Database and Blackboard

The patient specific database is kept separate from patient data on the blackboard; data are transferred from the database to the raw_data and diagnosis blackboard levels and from the raw_data and sub_diagnosis levels back to the database. An overview of the data transfer between the files, database and blackboard is shown in Figure 5.7.

The four sub-sections below, describe the knowledge sources which manipulate raw patient data (collectively known as *data handlers*).

5.3.2 Data Transfer From Database to Blackboard

The raw_data knowledge source monitors the database for the variable, symptom and history data types and is triggered by any such data that exist in the database but not on the blackboard, or by data that have a different value in the database than on the blackboard. Write_raw_data operates simply by replacing any existing value of the data at the raw_data blackboard level with the new value found in the database. The write_disease_diagnosis knowledge source operates in the same way, monitoring the database for data of the disease type and transferring them to the diagnosis level of the blackboard.

The knowledge source write_default_data monitors laboratory data variables that have default values specified in the knowledge base; it is triggered after write_raw_data and derive_data and if neither has been able to provide a data value, the default is written as an entry at the raw_data level. The mapping of data representations from the database on to the blackboard is shown in Figure 5.8.

```
current_data(Type,Name,Value).
```

↓ write_raw_data

```
blackboard(Blackboard,raw_data,Type,Name,Value,measurement).
```

```
data_parameter(Variable,default,Value).
```

↓ write_default_data

```
blackboard(Blackboard,raw_data,variable,Variable,Value,default).
```

```
current_data(disease,Root_disease,Disease_name)>
```

↓ write_disease_diagnosis

```
blackboard(Blackboard,diagnosis,Root_disease,Disease_name,1.000).
```

Figure 5.8 Mapping of Data from Database to Blackboard

5.3.3 Truth Maintenance

Any changes in the values of data in the database are detected by write_raw_data and the entries at the raw_data blackboard level are updated accordingly. Entries at the classified_data and sub_hypothesis levels are automatically updated because the knowledge sources that produced them are re-triggered by the changes at the raw_data level. Similarly, any necessary changes to default or derived data are made by simple re-triggering of the appropriate knowledge source. Thus truth maintenance is largely achieved by a passive process of knowledge source re-triggering.

If, however, the user changes a data value to 'unknown' then it is completely removed from the database and write_raw_data does not detect a change. This problem is solved by the truth_maintenance knowledge source which is triggered by data that have entries as raw_data on the blackboard but no corresponding entry in the database. Truth_maintenance removes any relevant entries from the classified_data and sub_hypothesis levels and also removes raw_data entries that were derived using data that are now unknown.

```
relation(Relation_no,[Variable,'='|Expression]).
```

```
derivation(Variable,Expression,Dependents_list).
```

For Example:

```
relation('1',['Anion Gap','=','Na','+','K','-','HCO3',-'Cl'].
```

```
derivation('Anion Gap',['Na','+','K','-','HCO3','-','Cl'],
                                    ['Na','K','HCO3','Cl']).
```

Figure 5.9 Pre-processing for Data Derivation

5.3.4 Data Derivation

The knowledge base contains expressions of the relationships between variables that can be used to derive data values. These relationships are of the form:

$$\text{Data variable} = \text{Expression} \tag{5.3.1}$$

where the expression involves one or more other data variables. Some pre-processing is performed on the relationships as they are stored in the knowledge base in order to isolate the variable on the LHS of (5.3.1) and to identify the variables involved in the RHS expression. The results of this pre-processing are shown in Figure 5.9. The data_derivation knowledge source is triggered when all the items in the Dependents_list appear as entries at the raw_data level; the Expression is then evaluated and the derived data variable is written on the blackboard. The data transformations occurring in this process are shown in Figure 5.10.

```
blackboard(Blackboard,raw_data,variable,Variable1,Value1,Status1).
blackboard(Blackboard,raw_data,variable,Variable2,Value2,Status2).
```

```
derivation(Variable,Expression,[Variable1,Variable2]).
```

derive_data

```
blackboard(Blackboard,raw_data,variable,Variable,Value,derivation).
```

Figure 5.10 Data Mapping Achieved by derive_data Knowledge Source

```
blackboard(Blackboard,raw_data,variable,Name,Value,default).
```

↓ write_default_data

```
current_data(variable,Name,Value).
```

```
blackboard(Blackboard,raw_data,variable,Name,Value,derivation).
```

↓ derive_data

```
current_data(variable,Name,Value).
```

```
blackboard(Blackboard,sub_diagnosis,Root,Disorder,Belief).
```

↓ transfer_data (if Belief≥0.95)

```
current_data(history,Disorder,'present').
```

Figure 5.11 Mapping of Data from Blackboard to Database

5.3.5 Data Transfer From Blackboard to Database

Default and derived data are written to the database, as well as to the blackboard, by the appropriate knowledge source; the only other transfer of data from the blackboard is achieved by the transfer_data knowledge source. Transfer_data takes each diagnosed disorder from the sub_diagnosis level and writes it at the manifestation level of the clinical diagnosis panel. In addition, any disorders that have been diagnosed with a belief greater than 0.95 are written to the database as patient history. The mapping of data structures from the blackboard to the database is shown in Figure 5.11

5.3.6 Data Classification

The method of data classification introduced in Section 4.3.4 is implemented in the classify_raw_data knowledge source. It is triggered by variables that have a value written at the raw_data level and a mean and standard deviation specified as data parameters in the knowledge base.

From Figure 4.11, if a laboratory data variable V is measured as X and the reference population mean and standard deviation are μ and σ respectively:

$$\begin{aligned} P(X \text{ is low}) &= P(u|u \geq X+2\sigma) \\ &= 1-\phi\left(\frac{X+2\sigma-\mu}{\sigma}\right) \end{aligned} \tag{5.3.2}$$

$$P(X \text{ is normal}) = P(u | X-2\sigma \le u \le X+2\sigma)$$
$$= \phi\left(\frac{X+2\sigma-\mu}{\sigma}\right) - \phi\left(\frac{X-2\sigma-\mu}{\sigma}\right) \tag{5.3.3}$$

$$P(X \text{ is high}) = P(u | u \le X-2\sigma)$$
$$= \phi\left(\frac{X-2\sigma-\mu}{\sigma}\right) \tag{5.3.4}$$

where ϕ is the standard Gaussian distribution function.

The classify_data knowledge source calculates the probabilities in (5.3.2), (5.3.3) and (5.3.4) using a look-up table of the standard Gaussian distribution function and writes entries at the classified_data level of the blackboard. The operation of the knowledge source is represented in Figure 5.12.

5.4 Handling Evidence

5.4.1 Overview

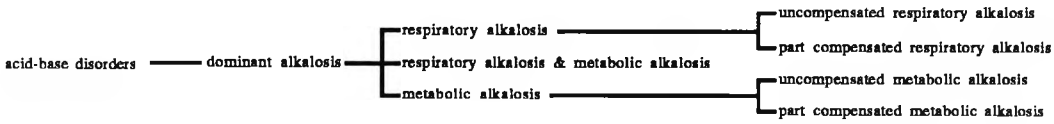
Three knowledge sources (collectively called evidence handlers) implement the method of evidence propagation presented in Section 4.3.3. Before the session begins, some pre-processing is performed on the hypothesis hierarchies stored in the knowledge base. The *a priori* probability of each hypothesis is calculated by multiplying the weights of the links from the hypothesis to the root node of the hierarchy; the set of descendent leaf nodes is also formed for each hypothesis. The results of the pre-processing are shown in Figure 5.13.

```
data_parameter(Variable,mean,Mean).
data_parameter(Variable,standard_deviation,SD).
```

↓ classify_data

```
blackboard(Blackboard,classified_data,Variable,low,Pl).
blackboard(Blackboard,classified_data,Variable,normal,Pn).
blackboard(Blackboard,classified_data,Variable,high,Ph).
```

Figure 5.12 Data Mapping Achieved by classify_data Knowledge Source




```
link("acid-base disorder","acid-base disorder","dominant alkalosis",0.3).
link("acid-base disorder","dominant alkalosis","resp alkalosis",0.33).
link("acid-base disorder","resp alkalosis","uncomp resp alk",0.5).
link("acid-base disorder","resp alkalosis","comp resp alk",0.5).
```

↓

```
apriori(disorder,"acid-base disorder","uncomp resp alk",0.0495).
singleton_descendents(disorder,"acid-base disorder","resp alkalosis",
                                        ["uncomp resp alk","comp resp alk"]).
```

Figure 5.13 Pre-processing for Hypothesis Hierarchies

Figure 5.14 shows the algorithm used to implement the updating function in (4.10). Operating separately on each hypothesis class, the algorithm first forms a list of all hypotheses that have a probability assignment in the knowledge base for the piece of evidence under consideration (Step 1).

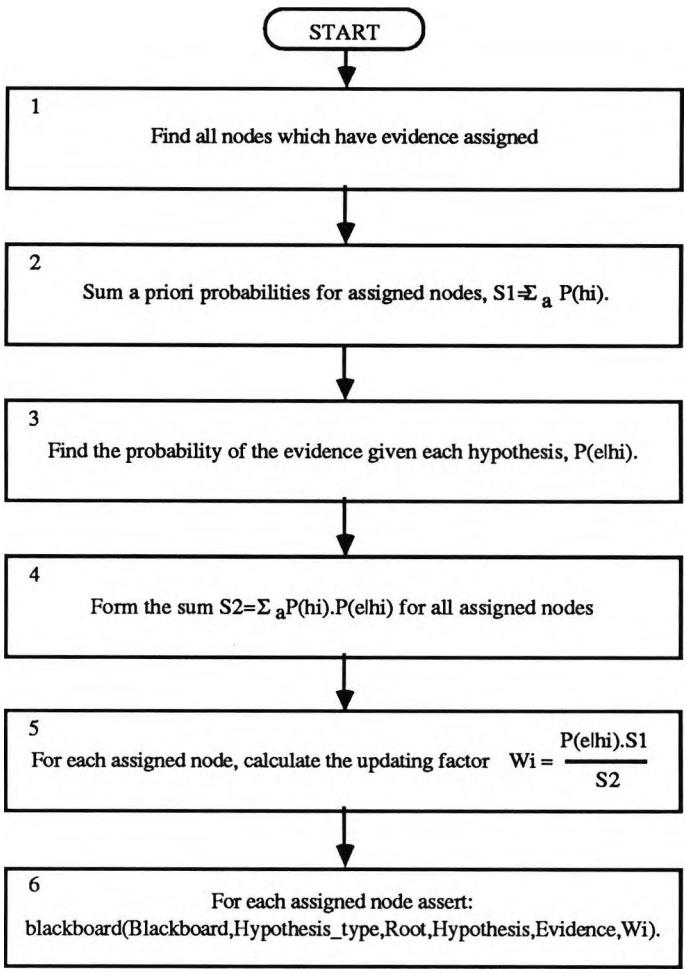


Figure 5.14 Algorithm for Evidence Handlers

The *a priori* probabilities of each hypothesis, $P(h_i)$, are summed to S1, the probabilities of the evidence given each hypothesis, $P(e|h_i)$, are found and the products $P(e|h_i).P(h_i)$ are summed to S2 (steps 2, 3 and 4). The updating factor in (4.11) is then calculated as $w_i=P(e|h_i).S1/S2$ and an appropriate entry is written at the sub_hypothesis level of the blackboard.

For different types of evidence there are variations in Steps 1 and 3 of the basic algorithm which are described in the next three sub-sections.

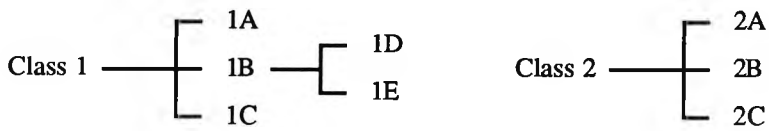
5.4.2 Signs/Symptoms and History

The evidence knowledge source is triggered by entries at the raw_data level of the data types symptom or history. A piece of evidence is a facet-attribute pair, for example ("GI fluid loss","present") or ("smoking habits","0-5 per day"). Recalling from Figures 4.20b and 4.21b that the knowledge base contains entries for hypothesis frames of the type

frame_symptom(Hypothesis,Root,Facet,Attribute,Probability).

frame_history(Hypothesis,Root,Facet,Attribute,Probability).

the list of hypotheses with assignments for the evidence (Step 1) is formed from all hypotheses for which the evidence facet matches a Facet in the knowledge base. If the evidence attribute also matches an Attribute in the knowledge base then $P(e|h_i)$ is simply recovered as the Probability argument; if no match for the attribute is found then $P(e|h_i)=0$.




```
frame_history("2C","Class 2","1E",present,0.7).
frame_history("2C","Class 2","1D",present,0.3).
frame_history("2B","Class 2","1C",present,1.0).
```

evidence is:

```
blackboard(Blackboard,raw_data,history,"1B",present).
```

P(evidence|2C)=0.7+0.3=1.0

P(evidence|2B)=0.0

Figure 5.15 Finding $P(e|h_i)$ for Diseases and Disorders Added as History

For diseases or disorders that have been specified as pieces of history evidence, the situation is a little more complicated. The list in Step 1 is formed from all hypotheses which have a Facet in the knowledge base that is in the same class of diseases or disorders as the evidence facet. The probability $P(e|h_i)$ is found by summing the Probability arguments of all knowledge base entries for the hypothesis for which the Facet is the same as or a descendent of the evidence facet and the attributes are the same; an example is shown in Figure 5.15.

5.4.3 Laboratory Data Variables as Evidence

Entries describing variables at the classified_data level trigger the variable_evidence knowledge source. The knowledge base contains entries of the type

frame_variable(Hypothesis,Root,Variable,Level,Probability)

and so the list in Step 1 is formed by finding hypotheses matching the Variable argument with the evidence variable. The Probability argument expresses P(Variable is Level|Hypothesis) and so the probability of the evidence given the hypothesis is:

$$P(e|h_i) = P(low|h_i).P(low|e)+P(normal|h_i).P(normal|e)+P(high|h_i).P(high|e)$$

where P(low|e), P(normal|e) and P(high|e) are found at the classified_data level of the blackboard. An example calculation is shown in Figure 5.16.

BLACKBOARD

```
blackboard(Blackboard,classified_data,V,low,0.35).
blackboard(Blackboard,classified_data,V,normal,0.65).
blackboard(Blackboard,classified_data,V,high,0.0).
```

KNOWLEDGE BASE

```
frame_variable("H1","Root",V,low,0.5).
frame_variable("H1","Root",V,normal,0.5).

frame_variable("H2","Root",V,normal,0.5).
frame_variable("H2","Root",V,high,0.5).
```

↓

$$P(V|H1) = 0.35*0.5 + 0.65*0.5 + 0.0*0.0 = 0.5$$

$$P(V|H2) = 0.35*0.0 + 0.65*0.5 + 0.0*0.5 = 0.325$$

Figure 5.16 Calculation of $P(e|h_i)$ for Classified Data Variables

5.4.4 Relationships as Evidence

Hypothesis frames in the knowledge base contain relationships between laboratory data variables that are characteristic of the hypothesis (see Figure 4.19a,b). The general form of these relationships is:

[LHS Expression] [Comparator] [RHS Expression]

where the comparator is one of {=,<,>,=<,>=} and the two expressions contain laboratory data variables. Pre-processing of the relationships identifies the comparator, separates the LHS and RHS expressions and forms the list of variables involved in them. The special symbol Δ, representing the change in value of a variable, must also be identified; in the current implementation ΔV is taken as *the change in V from its mean value* and so ΔV is replaced by (V-μV) wherever it occurs in the relationship. It could also be interpreted as *the change in V since it was last measured*, in which case ΔV could be replaced by V-Va, where Va is the value retrieved from archive_data. The results of pre-processing for some typical relationships are shown in Figure 5.17.

```
frame_relation("mild hypoxaemia","hypoxaemic state","1",
                              ["PO2",">=","7",".","9","8"]).
```

```
relation_occurrence("hypoxaemic state",["PO2"]).

relation_evidence("mild hypoxaemia","hypoxaemic state","1",
                              ["PO2'],["7",".","9","8"],">=").
```

```
frame_relation("comp met alk","acid-base disorder","1",
["\127","PCO2","=<","0",".","0","3","3","*","\127","HCO3"]).
```

```
relation_occurrence("acid-base disorder",["PCO2",HCO3"]).

relation_evidence("comp met alk","acid-base disorder,"1",
["(","PCO2","=","5.32",")"],
["0",".","0","3","3","*","(","HCO3","-","24",")"],"=<").
```

Figure 5.17 Pre-processing for Relationships

It can be seen from Figure 5.17 that the occurrence of each set of variables {V1,V2,...Vn} that appears in some relationship is recorded. These sets are used to trigger the relation_evidence knowledge source when all the variables they contain become available at the raw_data level. The evidence considered by the triggered knowledge source is then *a relationship between the variables {V1,V2,...Vn}*. The set of hypotheses in Step 1 of Figure 5.14 is generated by finding all hypotheses featuring in a relation_evidence entry with the same set of relationship variables. The algorithm for finding $P(e|h_i)$ is shown in Figure 5.18. There may be more than one relationship between the same variable set for a given hypothesis h_i: if all the relationships hold then $P(e|h_i)=1$; if any one of the relationships does not hold, $P(e|h_i)=0$.

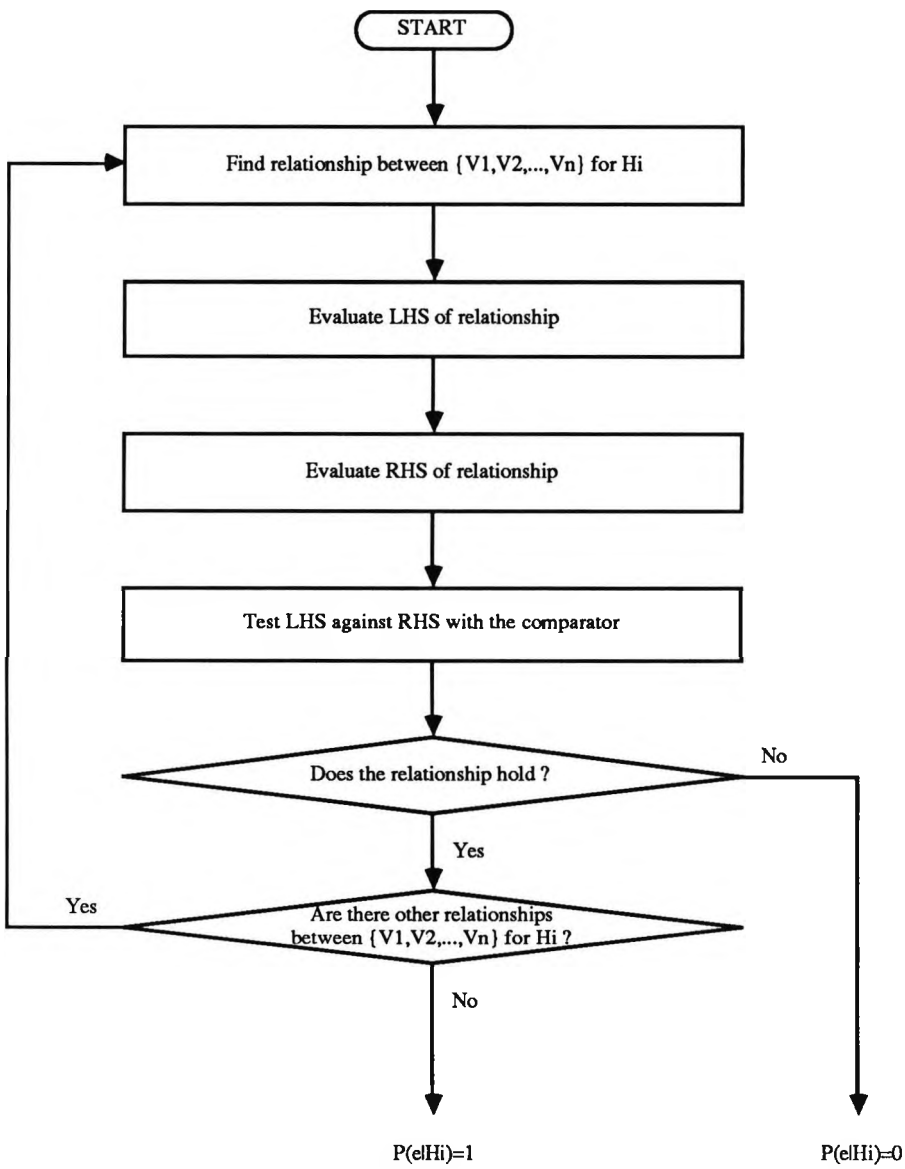


Figure 5.18 Algorithm for Calculating $P(e|h_i)$ for Relationship Evidence

5.5 Combining Hypotheses

The three knowledge sources described in the last section calculate the updating factors for the belief hypotheses and write them to the sub_hypothesis level of the blackboard. This section describes how the hypotheses are combined and reported as differential diagnosis lists.

The hypotheses at the sub_hypothesis level for a particular class of disorders can be at any level in the hierarchy and there may be more than one entry for each hypothesis. The sum_hypothesis knowledge source is triggered by the entries at the sub_hypothesis level and writes the updated belief of leaf nodes in the hierarchy at the hypothesis level. The algorithm used by sum_hypothesis is shown in Figure 5.19.

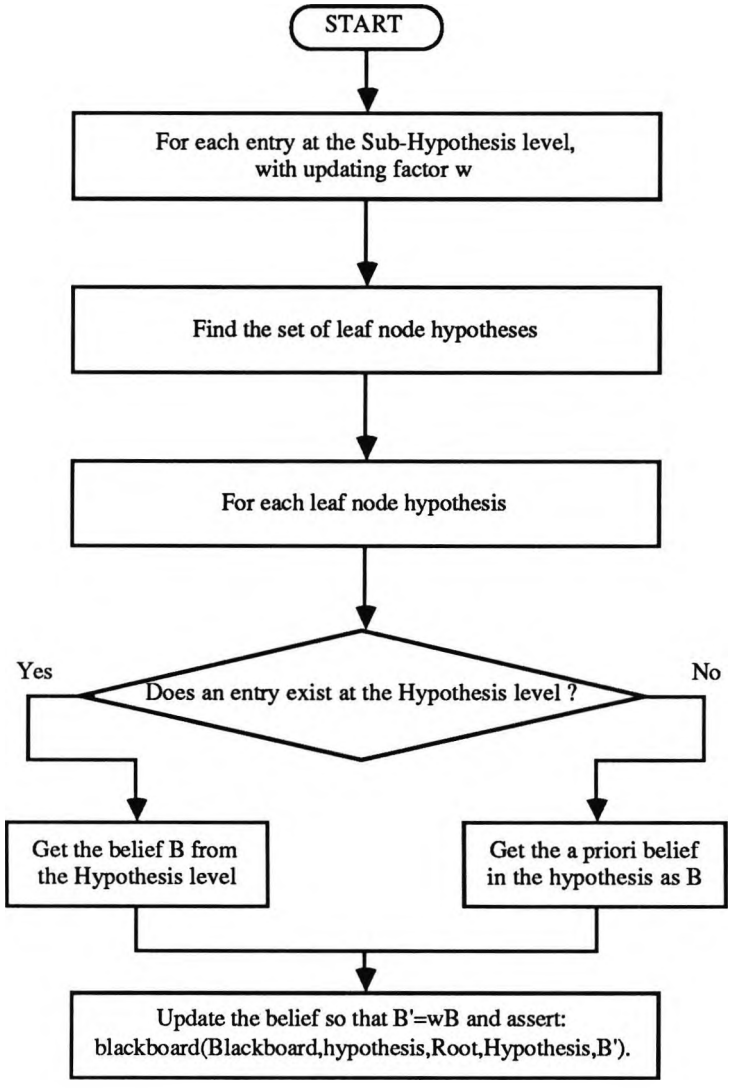
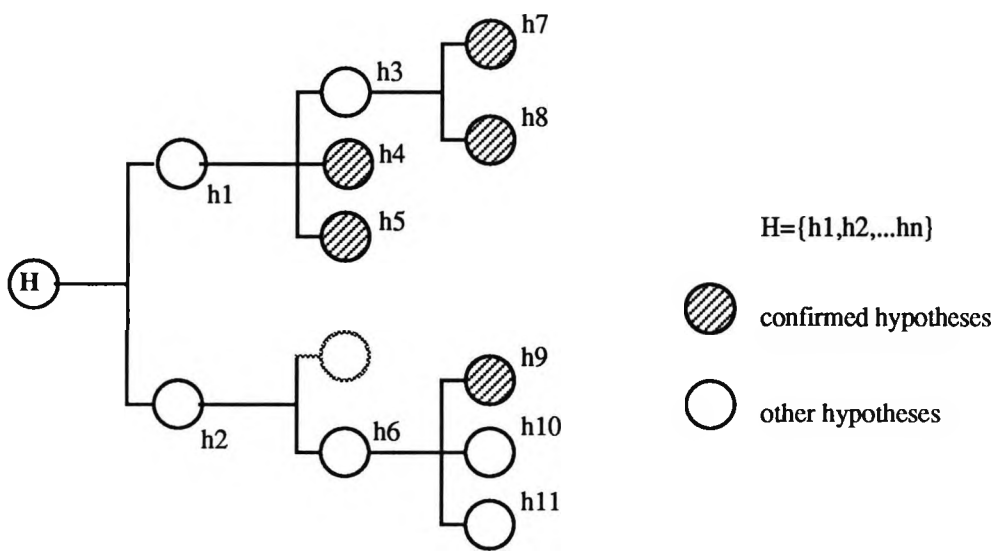


Figure 5.19 Algorithm for sum_hypothesis Knowledge Source

For each hypothesis at the sub_hypothesis level, with associated updating factor w, the belief in each of its descendent leaf nodes is updated by w. If an entry for the leaf node already exists at the hypothesis level then it is the belief specified there that is updated; otherwise the *a priori* belief is updated and written to the hypothesis level. This method of updating ensures that however high its *a priori* probability may be, a hypothesis does not appear at the hypothesis level of the blackboard until some evidence is observed that affects its belief.

The rank_hypothesis knowledge source is triggered by entries at the hypothesis level and reports diagnoses in rank order at the sub_diagnosis level of the blackboard. Rank_hypotheses ensures that diagnoses are reported at the most appropriate level of abstraction in the disorder class hierarchies. If each of the descendent leaf nodes of a hypothesis appears at the hypothesis level of the blackboard with belief increased from its *a priori* value, then they are combined to form a single diagnosis with belief equal to the sum of the beliefs of the leaf nodes. This is illustrated by an example in Figure 5.20.




```
blackboard(Blackboard,hypothesis,H,hn,Beln).
```

↓

```
blackboard(Blackboard,sub_diagnosis,H,h1,Bel1).

blackboard(Blackboard,sub_diagnosis,H,h9,Bel9).
```

Figure 5.20 Combining Hypotheses at the sub_diagnosis Level

5.6 Diagnosis Critique

A critique of the disorders diagnosed at the sub_diagnosis level of the physiological diagnosis panel is achieved by the top-down operation of the clinical diagnosis panel; the structure of the entries on this panel has been shown in Figure 5.2. Section 5.3 described how the patient's underlying clinical condition, input by the user, is written at the diagnosis level by the write_disease_diagnosis knowledge source and how disorders from the sub_diagnosis level are transferred to the manifestation level of the clinical panel by the knowledge source transfer_data.

The knowledge source predict_disorders finds in the knowledge base all the disorders that could be associated with the patient's diseases and writes them as entries at the prediction level of the clinical diagnosis panel. The action of predict_disorders is shown in Figure 5.21.

The critique is produced by comparing entries at the manifestation level with those at the prediction level. For each class of disorders appearing as manifestation entries, there are two types of critique that could be produced. If any one of the manifestations matches one of the predictions, the critique identifies those disorders consistent with the diagnosed diseases and those that are inconsistent. If none of the disorders in a class are consistent with the diseases diagnosed, the critique identifies the disorders that are expected with the diseases. The action of the critique_diagnoses knowledge source is shown in Figure 5.22.

```
blackboard(Blackboard,diagnosis,Root,Disease1).
blackboard(Blackboard,diagnosis,Root,Disease2).


frame_history(Disease1,Root,DisorderX,"present",ProbabilityX).
frame_history(Disease1,Root,DisorderY,"present",ProbabilityY).
frame_history(Disease2,Root,DisorderZ,"present",ProbabilityZ).
```

```
blackboard(Blackboard,prediction,Disease1,DisorderX).
blackboard(Blackboard,prediction,Disease1,DisorderY).
blackboard(Blackboard,prediction,Disease2,DisorderZ).
```

Figure 5.21 Action of the predict_disorders Knowledge Source

The critiquing implemented by the clinical diagnosis panel is a fairly simple comparison of the disorders diagnosed with those expected from the patient's clinical condition. The critique could be made more sophisticated by taking into account the probabilities of disorders with given diseases and the belief in each disorder diagnosed.

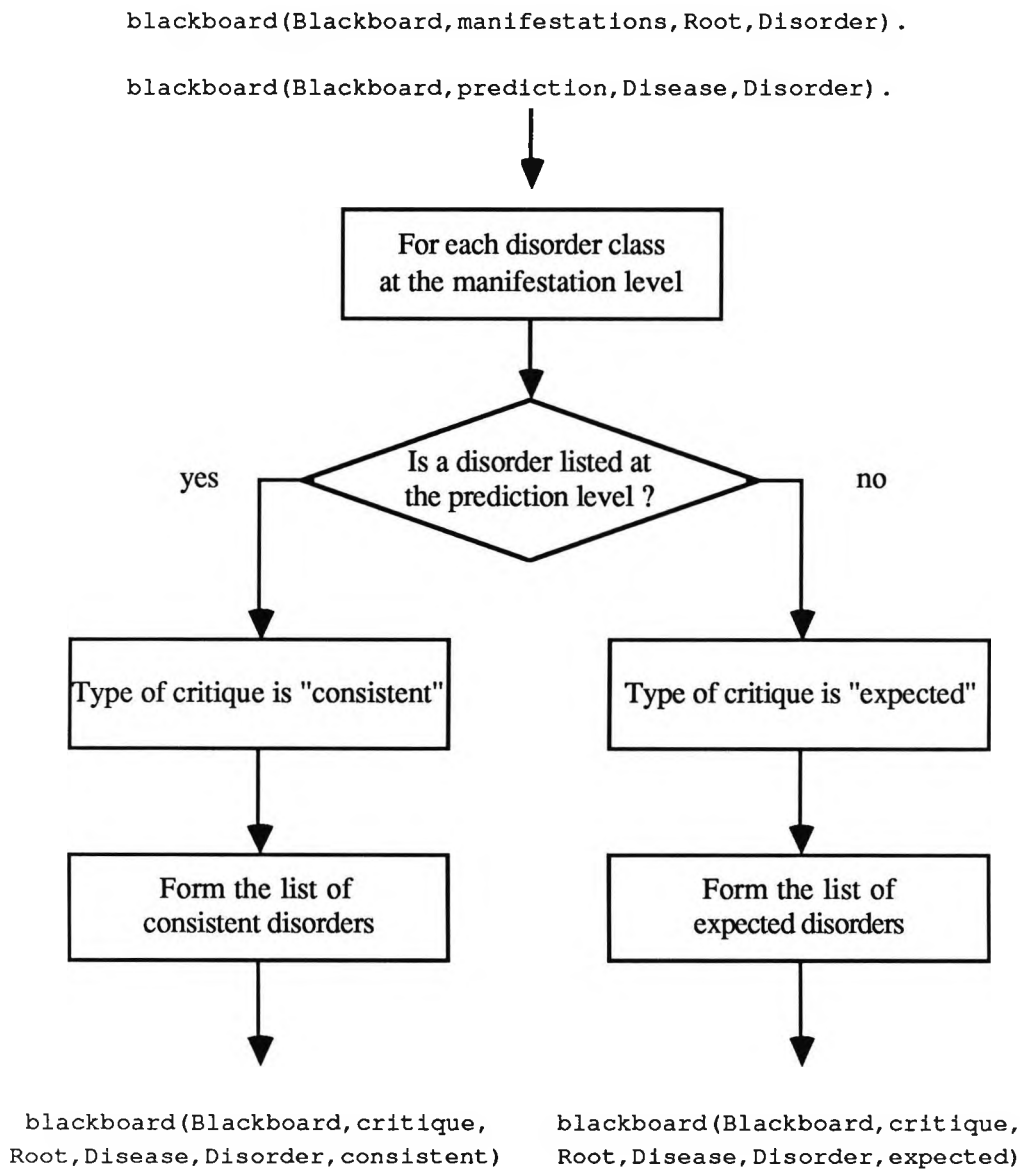


Figure 5.22 Action of the critique_diagnosis Knowledge Source

5.7 Dialogue Interaction

5.7.1 Introduction

The preceding sections in this chapter have described the blackboard diagnosis module of the knowledge-based system depicted in Figure 4.8; this section describes the dialogue management module, which handles the presentation of diagnostic results in textual form, the processing of user queries and the output of responses to those queries. The ouput of text, either for diagnosis results or query responses, is achieved in two phases. First, the content of the text is generated and stored in temporary data structures using the predicate output - the arguments of this predicate can be single words, groups of words, punctuation descriptions or special instructions to output lists. The second phase of text output is to retract the output predicates in the order that they were generated, displaying the text and inserting the appropriate punctuation and formatting. An example of the transformation of output predicates to text is shown in Figure 5.23.

The remaining sections of this chapter describe the methods used to output the diagnoses of disorders and to handle four types of user query: requests for information, requests for explanation of the diagnosis, requests for details of the way in which an observation affected the diagnosis and suppositions about further observations.

5.7.2 Textual Presentation of Diagnoses

The results of the diagnosis of physiological disorders appear at the sub_diagnosis level of the blackboard as differential lists in which each possible diagnosis has an associated belief in the range [0,1]. The presentation of these diagnoses in a textual form is achieved by a rule-based text generator (see Section 4.8).

```
output(new_paragraph).
output([my,diagnosis,for,"Fred Bloggs",is]).
output(list(["compensated metabolic acidosis"],
            ["compensated respiratory alkalosis"]).
output(end_text).
```

```
My diagnosis for Fred Bloggs is compensated metabolic
acidosis or compensated respiratory acidosis.
```

Figure 5.23 Form Output Data Structures to Text

$P(D\|e) > 0.95$	certainly true
$0.7 < P(D\|e) \leq 0.95$	likely
$0.35 < P(D\|e) \leq 0.7$	quite likely
$P(D\|e) > 0.1$	possible

Table 5.2 Translation of Numerical Belief Measures

The first step towards text generation is to form a list of the diagnoses for each class of disorders, in which the numerical belief measures are translated into linguistic terms using the criteria in Table 5.2. This should be compared to the translation of fuzzy measures in CADIAG-2 which is featured in Table 2.1. Using these criteria, it can be seen that there are only five possible forms for the diagnosis list of any disorder class; these are shown in Table 5.3.

For each list of diagnoses, rules can be written for the generation of text, which take into account the type of the list and the context in which that passage of text is presented. The context is set by each generating rule and depends on the current and the preceding types of diagnosis list. The entire rule set required to generate the textual output of diagnoses is shown in Table 5.4.

Type 1	< One Certain Diagnosis >
Type 2	<One Likely Diagnosis> <Zero, One or Two Possible Diagnoses>
Type 3	<Two Quite Likely Diagnoses> <Zero or One Possible Diagnoses>
Type 4	<One Quite Likely Diagnosis> <Zero to Six Possible Diagnoses>
Type 5	<Zero to Nine Possible Diagnoses>

Table 5.3 Forms of Differential Diagnosis List

If this is the first diagnostic statement about <patient>
and the differential diagnosis list for disorder class <Root> is Type 1
then output: *My diagnosis for <patient> is <Certain Diagnosis>*

If this is the first diagnostic statement about <patient>
and the differential diagnosis list for disorder class <Root> is Type 2
then output: *The most likely diagnosis for <patient> is <Likely Diagnosis>*

If this is the first diagnostic statement about <patient>
and the differential diagnosis list for disorder class <Root> is Type 3
then output: *My diagnosis for <patient> is <Likely Diagnosis 1> or <Likely Diagnosis 2>*

If this is the first diagnostic statement about <patient>
and the differential diagnosis list for disorder class <Root> is Type 4
then output: *<patient> could have <Likely Diagnosis> or possibly <List of Possible Diagnoses>*

If this is the first diagnostic statement about <patient>
and the differential diagnosis list for disorder class <Root> is Type 2
then output: *I am unable to diagnose <Root> using the available information*

If the last diagnostic statement was Type 1, 2 or 3
and the differential diagnosis list for disorder class <Root> is Type 1
then output: *with <Certain Diagnosis>*
and output: *<new sentence>*

If the last diagnostic statement was Type 4, 5 or <new sentence>
and the differential diagnosis list for disorder class <Root> is Type 1
then output: *<new sentence>*
and output: *<he/she> also has <Certain Diagnosis>*

If the last diagnostic statement was Type 1
and the differential diagnosis list for disorder class <Root> is Type 2
then output: *and probably <Likely Diagnosis>*
and output: *<new sentence>*

If the last diagnostic statement was Type 2
and the differential diagnosis list for disorder class <Root> is Type 2
then output: *and <Likely Diagnosis>*
and output: *<new sentence>*

If the last diagnostic statement was Type 3, 4, 5 or <new sentence>
and the differential diagnosis list for disorder class <Root> is Type 2
then output: *<new sentence>*
and output: <he/she> probably has*<Certain Diagnosis>*

If the last diagnostic statement was Type 1, 2 or 3
and the differential diagnosis list for disorder class <Root> is Type 3
then output: *and either <Likely Diagnosis 1> or <Likely Diagnosis 2>*
and output: *<new sentence>*

Table 5.4 Rules for the Output of Diagnoses
(continued on the next page)

If the last diagnostic statement was Type 4,5 or <new sentence>
and the differential diagnosis list for disorder class <Root> is Type 3
then output: *<new sentence>*
and output: *<he/she> has either <Likely Diagnosis 1> or <Likely Diagnosis 2>*

If the last diagnostic statement was Type 1, 2, 3, 4 or <new sentence>
and the differential diagnosis list for disorder class <Root> is Type 4
then output: *<new sentence>*
and output: *<he/she> could also have <Likely Diagnosis> or <List of Possible Diagnoses>*

If the last diagnostic statement was Type 5
and the differential diagnosis list for disorder class <Root> is Type 4
then output: *<new sentence>*
and output: *<he/she> could have <Likely Diagnosis> or <List of Possible Diagnoses>*

If the last diagnostic statement was Type 1, 2, 3, 4 or <new sentence>
and the differential diagnosis list for disorder class <Root> is Type 5
then output: *<new sentence>*
and output: *I am unable to diagnose <Root> using the available information*

If the last diagnostic statement was Type 5
and the differential diagnosis list for disorder class <Root> is Type 5
then output: *or <Root>*
and output: *<new sentence>*

Table 5.4 Rules for the Output of Diagnoses
(continued from the previous page)

5.7.3 Handling User Queries

5.7.3.1 Overview

User queries are processed in the three stages shown in Figure 3.22 (understand the query, retrieve the answer, present the response). The first stage, of understanding the query, is accomplished using PROLOG's grammar rule representation to parse the user's typed input (see Section 3.8.2). There are five types of sentence recognized by the dialogue parser: *basic_command, information_request, explanation_request, impact_request* and *supposition.* As well as recognizing the query type, the parser extracts the information contained in the query that is necessary for retrieval of the answer; this information is passed to the query handling routine for the identified query type.

A *core* lexicon of basic words and phrases is augmented by the application-specific lexicon, built up during interaction with the FRAMEBUILDER knowledge editing tool, to produce a *run-time* lexicon used by the dialogue parser. The core lexicon contains details of synonyms for words that may appear in the application-specific lexicon; this feature is used to replace abbreviations by their full descriptions in the dialogue output.

What is the <Facet> ?

What is the value of <Facet> ?

What is <his/her> <Facet> ?

What is the level of <Facet> ?

Table 5.5 Information Requests

The basic commands recognized by the dialogue parser scroll the dialogue window up or down and enable the user to exit from the dialogue module. Throughout the interaction, the system keeps a record of the context in which the queries are posed; they can refer either to the actual diagnosis or to hypothetical diagnoses made in response to suppositions on the part of the user. The methods used to process the four types of query, once they have been recognized by the dialogue parser, are described in the following sub-sections.

5.7.3.2 Information Requests

The requests for information about patient observations that are recognized by the dialogue parser are shown in Table 5.5. The parser extracts the tense of the query, the type of information required (variable, symptom or history), its name and whether a value or level (in the case of variables) is required. This information is retrieved from the raw_data or classified_data levels of the blackboard as required.

5.7.3.3 Suppositions

The user can ask about hypothetical diagnostic situations by altering the values of patient observations from within the dialogue module. The types of supposition recognized by the dialogue parser are shown in Table 5.6. Information about the type of observation to be set, its name and value are passed from the parser to the supposition handling procedure. The algorithm used by this procedure is shown in Figure 5.24.

What if <Facet> was <Value> ?

Suppose that <Facet> was <Value>

What if <Facet> was <Level> ?

Suppose that <Facet> was <Level>

Table 5.6 Suppositions

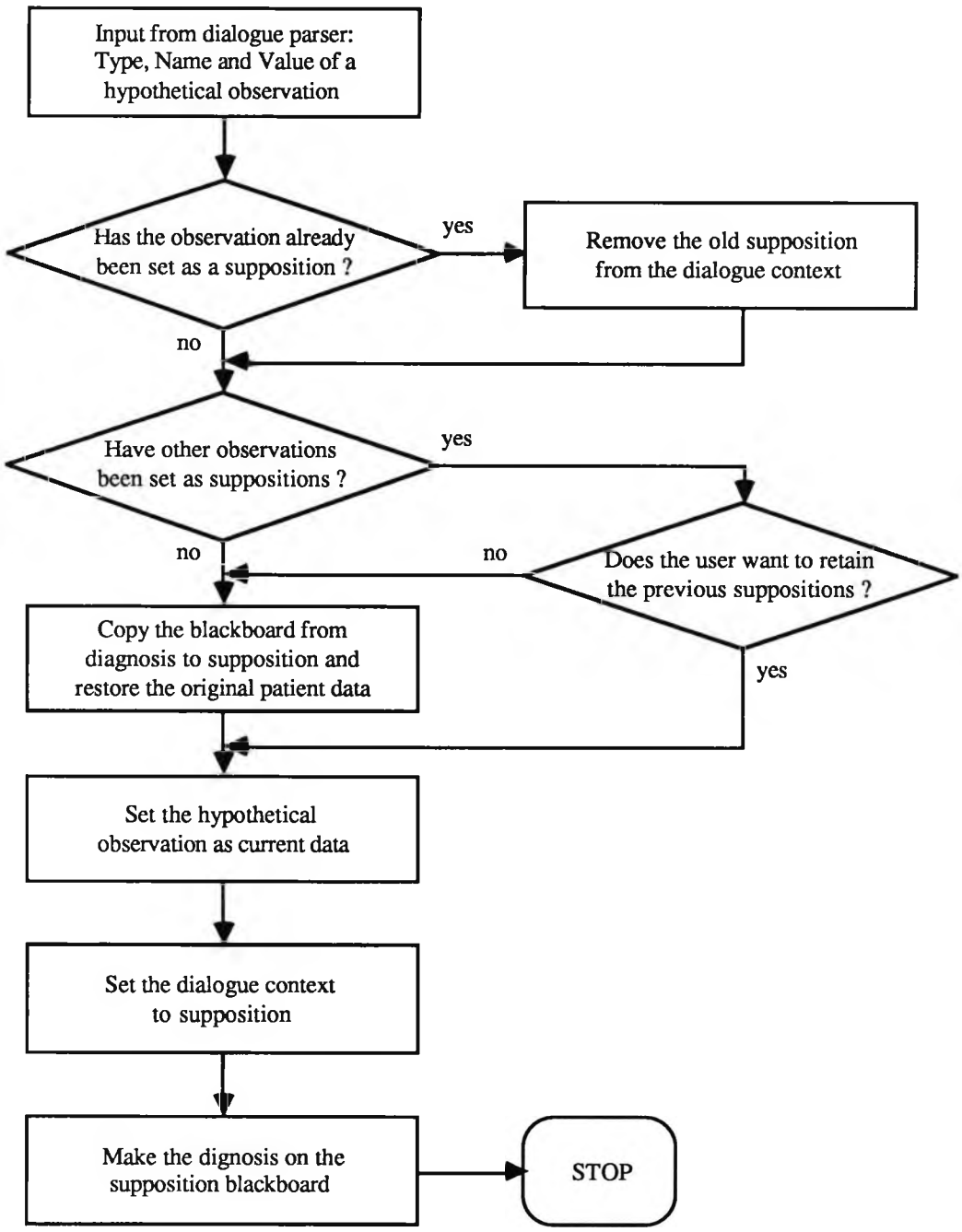


Figure 5.24 Algorithm for Processing Suppositions

On entry to the dialogue interaction module, the current patient specific data are stored in a temporary database. Hypothetical observations can be made simply by copying the entire blackboard data structure to a *supposition* blackboard and asserting the hypothetical data as current_data on the database; the original diagnosis remains preserved on the diagnosis blackboard and the original patient data can be restored from the temporary database as required.

A new diagnosis, based on the hypothetical data can be made using the blackboard diagnostic module operating on the supposition blackboard. This diagnosis, once made, is output in the manner described in Section 5.7.2 and the dialogue context is set to record that a hypothetical situation exists, based on the suppositions made by the user. Any subsequent queries made by the user are processed in the hypothetical context until a request is made to return to the original diagnosis. These subsequent queries may include further suppositions, in which case the user is asked if he wishes to retain the hypothetical context already created or to return to the original diagnosis before making the new supposition.

5.7.3.4 Explanations

By asking the question *why?*, the user can obtain an explanation of the diagnosis (either the original or hypothetical diagnosis, depending on the dialogue context) in terms of the evidence that supported each item in the differential list. Similarly, a request can be made for the evidence confirming a single, specified diagnosis or for the reason why a particular diagnosis was not made (in terms of the disconfirming evidence). A summary of the explanation requests recognized by the dialogue parser is given in Table 5.7.

The algorithm in Figure 5.25 is used to answer queries about why a diagnosis was or was not made. First, all the evidence relevant to the diagnosis is retrieved from the sub_hypothesis level of the blackboard. Evidence is relevant if it affects the belief in any node related to the disease in the disorder hierarchy.

Why ?
Why <Disorder> ?
Why did you diagnose <Disorder> ?
Why was <Disorder> diagnosed ?
Why not <Disorder> ?
Why didn't you diagnose <Disorder> ?
Why wasn't <Disorder> diagnosed ?

Table 5.7 Explanation Requests

For each piece of relevant evidence, the updating factor, ws, for belief in the diagnosis is calculated using the scheme shown in Figure 5.26. If ws=1 the evidence had no effect on the diagnosis; if ws>1 the evidence can be output as confirming evidence; if ws<1 the evidence can be output as disconfirming evidence.

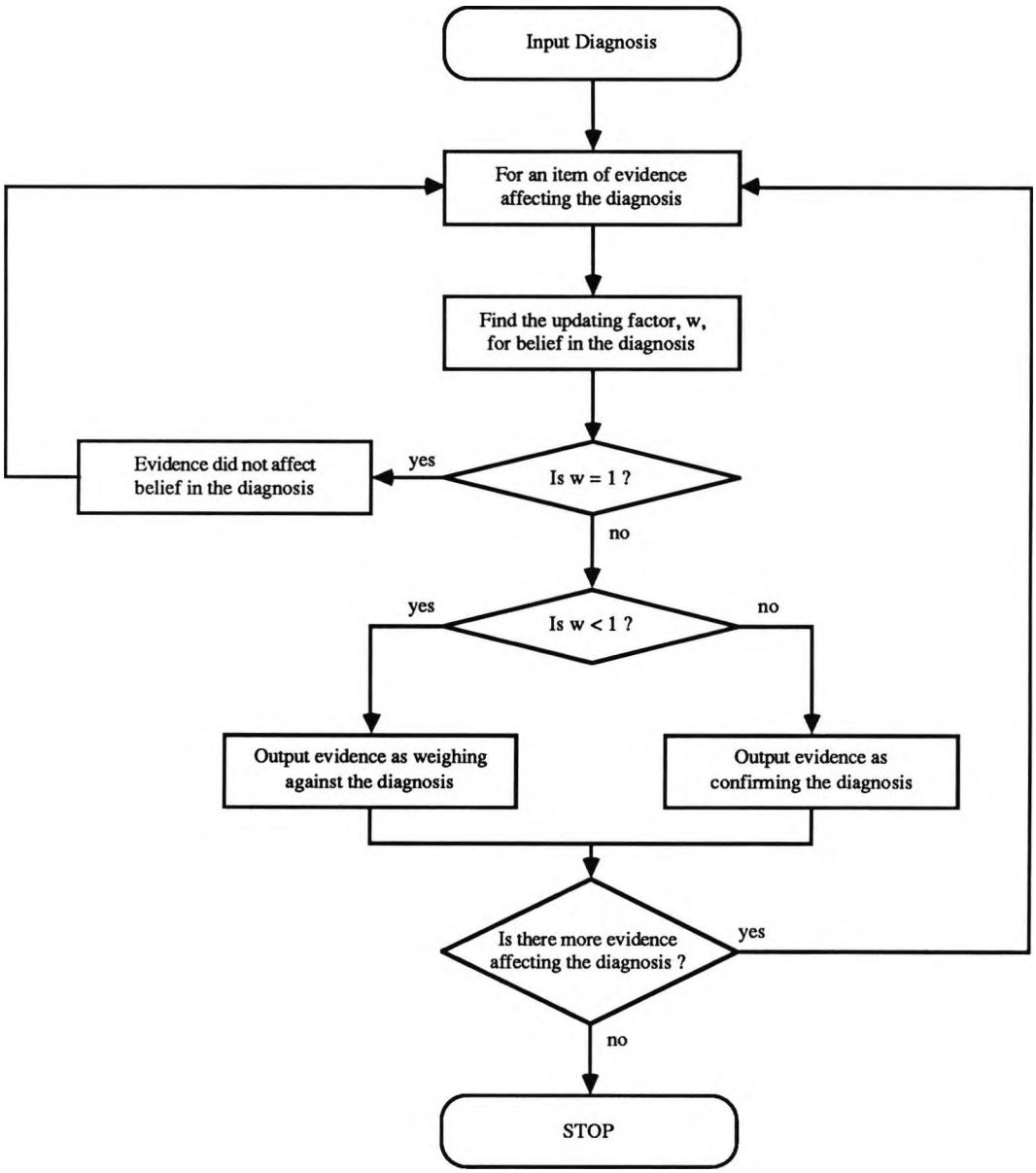


Figure 5.25 Algorithm for Explaining a Diagnosis

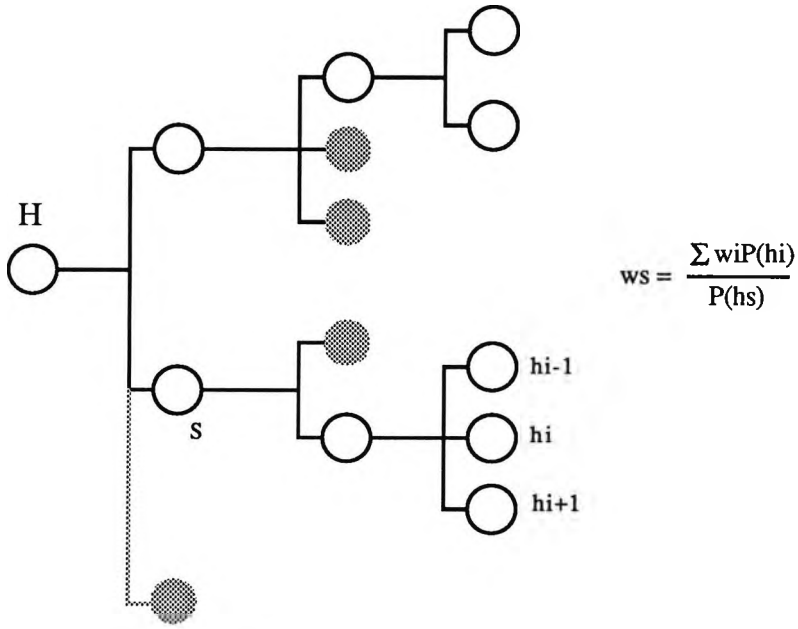


Figure 5.26 Finding the Updating Factor for a General Hypothesis Node

For a leaf node hypothesis, the updating factors for each piece of relevant evidence are stored at the sub_hypothesis level of the blackboard. For a general node, S, in the hierarchy, the updated belief in the hypothesis after a piece of evidence has been impacted is given by the sum of the updated beliefs in its descendent leaf nodes. Hence the updating factor for S can be defined as the ratio of this sum to the *a priori* belief in S. This is illustrated in Figure 5.26.

5.7.3.5 Finding The Effect of Evidence

The questions shown in Table 5.8 are recognized by the dialogue parser as requests to find the effect that a particular piece of evidence had on the disorders diagnosed. The parser identifies the data type and name of the evidence and for symptom and history evidence the effect on diagnoses is easily retrieved from the sub_hypothesis level of the blackboard.

How did <Facet> affect the diagnosis ?
What was the effect of <Facet> ?
How did <Facet> affect <Disorder> ?
How did <Facet> affect the diagnosis of <Disorder> ?
How was <Disorder> affected by <Facet> ?

Table 5.8 Requesting the Effect of Evidence

The updating factor, ws, for the belief in each disorder hypothesis S is found and output using the convention:

ws>1 evidence increased belief in S
ws<1 evidence decreased belief in S
ws=1 evidence did not affect belief in S
ws=0 evidence ruled out S as a diagnosis

For evidence from laboratory data variables, the situation is a little more complicated since the variable may appear as evidence due to its classified level and/or in one or more relationships for a single hypothesis. In this case an overall updating factor for the hypothesis is calculated as the product of the factors for each entry at the sub_hypothesis level in which the variable features. The effect of the variable as evidence for the hypothesis is then output by applying the convention defined above to the overall updating factor.

5.8 Summary

This chapter has described how a knowledge-based system was implemented using the programming language PROLOG. The basic control mechanism of a blackboard system was modelled by four PROLOG clauses (knowledge_source/4, trigger/4, ksar/3, executed_ks/3) and three simple control strategies were developed. Knowledge sources have been implemented to transfer data to and from the blackboard, classify laboratory data, impact evidence on hypothesis hierarchies, report the diagnosis in the most appropriate form and critique the diagnosis in the light of information about the clinical condition of the patient.

The grammar rule notation in PROLOG was used to implement an interface through which diagnoses can be presented in a textual form and the user can receive explanations of the system's conclusions in response to specific queries.

The performance and evaluation of the knowledge-based system, working in the domains of blood-gas analysis and hyperlipidaemia, forms the subject matter of the next two chapters.

PART THREE

CHAPTER 6

EVALUATION OF A KNOWLEDGE-BASED SYSTEM

6.1 Introduction

Knowledge-based systems, like any other computer software products, should undergo thorough testing before being released for general use, to ensure that they perform according to the original design specification. Conventional computer software is most usually designed to accomplish tasks which are computationally so complex that they cannot be performed by humans, or to relieve humans of the burden of tasks that are routine but time-consuming. Knowledge-based systems are designed to accomplish tasks that are currently performed by human experts and as a consequence they must be evaluated in ways which are not normally applicable to computer software. Generally speaking there are no right or wrong solutions to the problems tackled by knowledge-based systems - the correctness of any solution is a matter of subjective judgement and there may be more than one satisfactory solution to a given problem.

The next two sections discuss the problems of evaluation for knowledge-based systems in medicine both as final products delivered into a clinical setting and as prototypes emerging from the laboratory. This discussion is followed by a description of the evaluation of the system presented in the preceding chapters. The conclusions drawn from the evaluation of are presented in the final section of this chapter.

6.2 Evaluation in Clinical Practice

If a knowledge-based system is to be used routinely in clinical practice, it seems reasonable that it should undergo the same type of stringent evaluation that precedes the introduction of a new drug. First it should be thoroughly evaluated in the laboratory to verify its safety and performance under test conditions. Once it has passed this first phase of evaluation, a carefully controlled clinical trial should take place in which the outcome for a group of patients managed using the system is compared with the outcome for a group managed according to existing practice.

One of the few computer-aided diagnosis systems to have reached this level of evaluation is the system for the diagnosis of abdominal pain developed in Leeds (see Section 3.7.3). A study involving 8 hospitals, 250 clinicians and 16737 patients was designed to assess the impact of the system on clinical practice (Adams *et al*, 1986). The clinicians' performance

Diagnostic Accuracy 1st Investigation	Diagnostic Accuracy After Examination	Bad Diagnostic Error	Unnecessary Laparotomy	Perforated Appendix	Admissions of patients with non-specific abdomonal pain	Average hospital stay (days) non-specific abdomonal pain	Average hospital stay (days) Appendicitis	Mortality	
45.6%	57.9%	6.3%	25.2%	23.7%	47.4%	4.0	6.7	1.2%	Baseline
65.3%	74.2%	2.7%	10.4%	11.5%	26.3%	3.3	5.4	0.92%	Test

Table 6.1 Performance Indices in the Abdominal Pain System

was monitored during a base-line period of one year and a subsequent period of two years during which the computer system was available for use. The indices of performance measured during the study included: the diagnostic accuracy on first examination, the diagnostic accuracy after investigation, the number of unnecessary laparotomies (surgical investigations), the incidence of perforated appendix, the number of bad diagnostic errors (the patient required urgent surgery, but was not diagnosed), the number of admissions from the Accident and Emergency Department and the average length of stay in hospital. A summary of these indices as measured during the base-line and test periods is shown in Table 6.1.

This study measured the performance of clinicians acting with and without the aid of the computer system - it was not designed to measure the performance of the system in isolation. As can be seen from Table 6.1, all the performance indices registered an improvement during the period in which the computer system was available and it was estimated that the reduction in admissions, unnecessary laparotomies and length of hospital stay could lead to an annual saving of £20m for the National Health Service.

Another important observation was made during this study. The accuracy of the initial diagnosis was measured for several different groups of doctors: one using structured forms for data collection, one using the forms and receiving feedback about their diagnostic performance and another using forms, feedback and the computer. It was found that the use of forms to guide data collection increased diagnostic accuracy from 45.7% to 56.7% after one month and that in the same time period the use of forms, feedback and the computer increased accuracy to 64.8%. However, after four months, there was no significant difference between the group using forms and feedback and the group using forms, feedback and the computer (in both cases the diagnostic accuracy was over 70%).

This would seem to indicate that one of the major contributions to be made by computer systems is to encourage clinicians to adopt a more structured and systematic approach to diagnosis and that the provision of feedback about diagnostic performance can significantly increase accuracy. This latter point could also be viewed as an indirect benefit of the use of computer aids since they provide the impetus for the type of study decsribed above.

In contrast to a final evaluation of the clinical efficacy of a knowledge-based system along the lines of the clinical trial of a new drug, it could be argued that such a system should be evaluated in the same way as a human clinician. In this case the ultimate test of the system is to make it available for use at the discretion of the clinical staff - they will only come to depend on the system if it demonstrates a consistently high standard of performance over a long period of time. Although this may seem a somewhat casual approach , it has been used as the basic method of evaluation for several systems, most notably the DXPLAIN project sponsored by the American Medical Association (see Section 2.4.1).

6.3 Evaluation of Prototype Systems

6.3.1 Introduction

The clinical study of the diagnosis of abdominal pain, described above, was the culmination of almost 20 years of work on a computer system that is, at least conceptually, quite simple. It is an indication of the enormous effort required to introduce a system for routine clinical use and could explain why so few systems have achieved this status.

A system must undergo many other evaluations before its impact on clinical practice can be measured. Informal evaluation of an evolving knowledge-based system occurs throughout its development - indeed its plays a fundamental role in the development cycle as depicted in Figure 3.27. In many respects, informal evaluation drives the direction of the research effort, with modifications to the design and implementation of the system being made in response to the evaluation results (Cohen & Howe, 1988). However, at some point in the development cycle, it becomes useful to fix the design, implementation details and knowledge base of the system and to perform a formal evaluation of the prototype. It has been suggested that the first formal evaluation should take place once the developers of the system are satisfied with its performance in the majority of the test cases presented to it (Buchanan & Shortliffe, 1984; 578).

The purposes of the formal evaluation of the prototype system are these (Chandrasekaran, 1983):

> to provide a formal demonstration to those outside the development group that the system achieves a satisfactory level of performance.

to identify the areas in which the system fails to perform satisfactorily so that further research and development can be directed appropriately. Two specific areas to be considered are the richness of the knowledge base and the adequacy of the reasoning mechanism.

to assess the system in different operational roles: as a diagnostician, an explanation facility or an electronic textbook of medicine, for example.

to assess different facets of each operational role, for instance in a diagnostic role these might include accuracy, response time, ease of use, performance with missing data, etc.

The precise method used for evaluation is determined both by consideration of the system itself - its nature and the goals of the evaluation - and of the resources available for the evaluation experiment. A full evaluation involving hundreds of test cases requires a great investment of time by the participating clinicians, many of them, as experts, being extremely busy people. The number of test cases is usually restrained by this second consideration, although it must be ensured that a sufficient number are considered to render the evaluation statistically valid.

Alan Turing proposed that the answer to his question *Can machines think?* could be found by playing the imitation game (Turing, 1950). In this game a computer and a human being are placed in different rooms and are questioned by an observer. The output from each room is reformulated to disguise its origin and the observer must decide, on the basis of the responses, which room contains the computer and which the human. If the human and computer cannot be distinguished, then the computer has passed the imitation test and can be deemed a thinking machine.

It is this basic philosophy that underlies the methods of evaluation of many prototype knowledge-based systems (Chandrasekaran, 1983). Two basic variations can be identified in practice - methods which define a gold standard of diagnosis for each test case and compare the system with that standard and methods which present the system's output for critique by expert clinicians. These two methods are described in Sections 6.3.3 and 6.3.4 after a brief discussion of the way in which test cases should be selected.

6.3.2 Selecting Test Cases

The test cases to be used for the system evaluation can be selected from either retrospective or prospective data. Retrospective cases are readily available in the form of clinical patient records, but may not contain all the data required. Before selecting such cases, a clear specification of the scope of the knowledge-based system should be determined (*ie* the possible diagnoses it can make and the observations on which these are based). These same criteria can then be used to select the test cases - it is important to note that cases selected in this way do not test the completeness of the knowledge base. Retrospective cases can be selected randomly (*eg* the first 50 cases in a batch that meet the scope criteria) or can be selected from particularly interesting cases known to the participating clinicians, which is often the only way to include the rarer diagnoses.

Prospective cases can be gathered over a period of time from the clinical environment in which the knowledge-based system is intended to operate. The advantages of this method are that the precise data required can be recorded and that randomly selected cases reflect the true frequency of occurrence of each category of diagnosis (this may not always be so for retrospective case data). Incompleteness of the knowledge base may be revealed by randomly selecting the cases from the working domain of the system, but equally the entire knowledge base may not be exercised since the rarer diagnoses may not occur during the period of data collection. The best approach appears to be a random selection of cases (either retrospective or prospective) followed by a selection of specific retrospective cases in order to cover the diagnoses that did not occur in the random selection.

6.3.3 Comparison With a Gold Standard

In this method, the role of the human in the imitation game is taken by an expert clinician who specifies a gold standard of diagnosis for each case in the test set. The same cases are then presented to the computer system and the role of the observer is simply to report the number of cases in which the computer agrees the the gold standard.

Where the test cases have been selected from retrospective data, it may be possible to determine the gold standard from an unequivocal, retrospective diagnosis made at the time the patient was discharged or from evidence revealed at a post mortum examination. More often, the gold standard is determined by presenting an expert clinician with the same case data as the knowledge-based system; ideally, this expert clinician should be someone unconnected with the original system development (Chandrasekaran, 1983). One problem associated with this method is that clinical judgements are inevitably subjective and even an expert clinician may disagree with his peers in a significant proportion of cases. Several

DIAGNOSIS	MD1 MD2	MD1 PUFF	MD2 PUFF
Normal	92	95	92
OAD	94	99	94
RLD	92	99	85
DD	90	91	85
Total	92	96	89

OAD Obstrcutive Airways Disease DD Diffusion Defect
RLD Restrictive Lung Disease

Table 6.2 The Evaluation of PUFF

solutions to this problem have been found. The evaluation of the rheumatology consulting system AI/RHEUM (Kingsland, 1985) used a gold standard set by the consensus opinion of a panel of three expert clinicians for each of the 384 retrospective test cases. The validity of the standard was checked using a subset of 48 cases - the three clinicians were asked to make independent diagnoses for each case and it was found that there was agreement between at least two of them in 96% of cases (46/48).

Another approach was taken by the developers of the pulmonary function diagnostic system PUFF (Aikins et al, 1983). Each of 144 cases was diagnosed by the system, the expert who developed the knowledge base and a second expert unconnected with the project. The level of agreement was then compared as shown in Table 6.2. The highest level of agreement was between the system and the expert involved with its development; the level of agreement between the independent expert and the system was similar to that between the two experts. This seems to indicate that the system accurately captured the diagnostic expertise of the clinician on whose knowledge it was based.

6.3.4 Expert Critique

A more faithful interpretation of the original imitation game was used for the evaluation of MYCIN after evaluations based on comparison with a gold standard had shown that it was difficult to achieve accuracy greater than 75% (Yu *et al*, 1979). Ten cases were selected which covered a broad range of diagnoses and treatments. Six expert clinicians, one resident and one student were asked to prescribe therapy for each case; none of the eight participants were connected with the MYCIN project. The eight sets of prescriptions together with those of MYCIN and the attending clinician in each case (making 100 in all) were then presented to eight outside experts who were asked to form their own prescription

for each case and to critique the other prescriptions. For each of the 100 prescriptions they specified whether it was equivalent to their own, an acceptable alternative or unacceptable. It was found that MYCIN's prescription was considered acceptable (*ie* equivalent or acceptable) by a majority of the eight evaluators in 7 out of the 10 cases - the best clinicians scored only 5/10.

A critiquing method was also used to evaluate the system CASNET, which diagnosed the eye disease glaucoma. At a meeting of Opthalmologists, experts were asked to input difficult cases recalled from their past experience and to critique the diagnosis given by the system. In 77% of the 44 cases presented to the system, the experts judged it to perform at a competent level. Although at first sight this evaluation does not seem to be as thorough as the evaluation of MYCIN, the fact that completely random, difficult cases were used meant that CASNET faced a far stiffer test than MYCIN.

Overall, it would seem that the method of critique is a less stringent evaluation than a straight comparison with a gold standard, since evaluators may be more inclined to find the system's conclusions acceptable in the former method.

6.3.5 Measuring Diagnostic Accuracy

The traditional method of assessing the accuracy of a diagnostic test is to determine its diagnostic sensitivity and specificity (see for example Sunderman & Van Soestbergen, 1971). The outcome of a test is *true positive* for the cases in which it correctly indicates the diagnosis and *true negative* for those in which it correctly indicates its absence. Conversely, a *false positive* test indicates the diagnosis when it is in fact absent and a *false negative* test indicates its absence when it is present.

The *sensitivity* of a diagnostic test measures the proportion of cases in which the diagnosis is present that are correctly predicted by the test:

$$\text{Sensitivity} = \frac{\text{Correctly Predicted Diagnoses}}{\text{Total Cases of the Diagnosis}}$$

$$= \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

The *specificity* of a test measures the proportion of cases in which the absence of the diagnosis is correctly predicted:

$$\text{Specificity} = \frac{\text{Correctly Predicted Absence of Diagnosis}}{\text{Total Cases in which Diagnosis is Absent}}$$

$$= \frac{\text{True Negative}}{\text{True Negative + False Positive}}$$

The interplay between sensitivity and specificity can be appreciated by considering that a test which always indicated a single diagnosis, regardless of the true situation, would have a sensitivity of 1 but a specificity of 0. A completely random test for a diagnosis in which there were n possible outcomes with equal *a priori* probability (1/n) would have:

Diagnosis Present 1/n	Diagnosis Absent (n-1)/n
True Positive $(1/n)^2$	True Negative $((n-1)/n)^2$
Sensitivity 1/n	Specificity (n-1)/n

Instead of considering sensitivity and specificity in isolation, a better indication of the adequacy of a test is the diagnostic index:

diagnostic index = sensitivity + specificity

The perfect test has a diagnostic index of 2 and a completely random test a diagnostic index of 1.

Most of the published results of the evaluation of knowledge-based systems have reported accuracy as the ratio of correct diagnoses to total cases (where some gold standard has been used to determine the correct diagnoses):

$$\text{Diagnostic Accuracy} = \frac{\text{Number of Correct Diagnoses}}{\text{Total Number of Cases}}$$

Two performance indices were used in the evaluation of a knowledge-based system for the diagnosis and management of patients with transient ischaemic attacks (Reggia, 1985). The first of these, the Kappa Statistic (Spitzer *et al*, 1967) takes account of that part of the accuracy of the system that could be due to a purely random agreement:

$$k = \frac{\text{Accuracy} - P_c}{1 - P_c}$$

where P_c is the accuracy of a completely random system. If there are n mutually exclusive and exhaustive diagnostic hypotheses $P_c = 1/n$.

The second index of performance used by Reggia measures the degree of belief assigned by the system to the true diagnosis in each of the evaluation cases:

$$\text{Accuracy Coefficient } Q = (2/n).\Sigma_i(Bel_i-0.5)$$

where the summation is over the set of n evaluation cases and Bel_i is the degree of belief assigned by the system to the true diagnosis in the ith case. A random system would have Q=0, one that was always incorrect would have Q=-1 and a perfect system would have Q=1.

6.3.6 Discussion

Table 6.3 gives a summary of the evaluation results from a number of medical consultation systems. It is extremely difficult to compare the performance of these systems, since they cover a wide range of medical domains and were evaluated in experiments of varying degrees of rigour. The table shows the number of evaluation cases, the general method used (gold standard comparison or expert critique) and the highest reported accuracy. However, there are many other factors to be considered when analyzing these data.

INTERNIST-1 appears at first sight to perform at a much lower accuracy than the other systems (40% accuracy compared with over 68% for the least accurate of the other systems). This can be explained in part by the domains in which the systems operate. The INTERNIST-1 knowledge base covered some 75% of internal medicine whereas the other systems cover only very restricted domains; it is much easier to achieve high accuracy in a limited as opposed to a wide domain. Another factor is that the data used for evaluating INTERNIST-1 were drawn from a collection of particularly challenging cases published in the New England Journal of Medicine whereas the other systems were generally evaluated using routine cases. This touches on something of a paradox in the evaluation and use of knowledge-based systems. A system that performs well with difficult cases will not necessarily show the same level of success with the routine ones. However, a system that does not perform well in routine use will not be readily accepted - one that is correct in the majority of cases would appear to be more favourable, even though the small proportion of cases it gets wrong are probably the difficult ones with which clinicians most need its assistance.

It is important, therefore, to investigate the cases in which the system made an incorrect judgement, to make sure that useful advice is given in the cases where it is most needed. This is all the more important when a prototype is evaluated since usually one of the purposes of the evaluation is to identify the areas in which further development effort should be concentrated.

SYSTEM	NO OF CASES	METHOD OF EVALUATION	HIGHEST REPORTED ACCURACY	REFERENCE
MYCIN	10	critique	70%	Yu et al, 1979
AI/RHEUM	384	gold standard	94%	Kingsland et al, 1983
INTERNIST-1	43	gold standard	40%	Miller et al, 1982
CASNET	44	critique	77%	Weiss et al, 1978
PUFF (EMYCIN)	144	gold standard	96%	Aikins et al, 1983
ANEMIA (EXPERT)	30	critique	87%	Quaglini et al, 1988
CADIAG-2 (Rheumatology)	327	gold standard	68.2%	Adlassnig et al, 1985
TIA	103	gold standard	70.9%	Reggia, 1985

Table 6.3 Evaluation of Some Knowledge-Based Systems

Another point to consider in the evaluation of a knowledge-based system is the level of performance of the experts in the same domain. In certain domains, a system which nominally has a low level of accuracy may still be performing better than the relevant experts. Some attempt must be made in any reasonable evaluation to assess the level of performance of both the system and the human experts.

6.4 Bench Tests

6.4.1 Introduction

A number of bench tests of the knowledge-based system described in Chapters 4 and 5 were performed in order to evaluate the translation of the theoretical design into a working prototype. The results of these tests are presented in the next four subsections. A small knowledge base of three disorders was used to investigate the handling of laboratory data using the method introduced in Section 4.3.4. The performance of the system when presented with an incomplete set of laboratory data will be illustrated and an example of the dialogue interaction will be given. Finally, a comparison will made between the three methods of system control that were described in Section 5.2

6.4.2 Performance With Laboratory Data

A small knowledge base was constructed to test the system's performance in discriminating between possible diagnoses when presented with laboratory data only. The knowledge base consisted of a single laboratory data variable, V, which was defined as having a mean value 0.0 and standard deviation 1.0 in a reference population of healthy individuals. Three disorders were defined:

Disorder 1 was characterized by a low or normal level of V
Disorder 2 was characterized by a normal level of V
Disorder 3 was characterized by a normal or high level of V

Values of V in the range [-5,5] were input to the system and the resultant belief in the three disorders was plotted. Details of the knowledge base and the results obtained in the test are shown in Figure 6.1.

It can be seen from Figure 6.1 that when V=0.0 the belief in Disorder 2 (0.494) is about twice that of Disorders 1 and 3 (0.253). As V tends towards large positive values the belief in Disorder 3 tends towards unity and belief in the other disorders tends towards zero, with Disorder 2 always having a higher belief than Disorder 1. Similarly, as V tends towards large negative values the belief in Disorder 1 tends towards unity and belief in Disorders 2 and 3 tends towards zero.

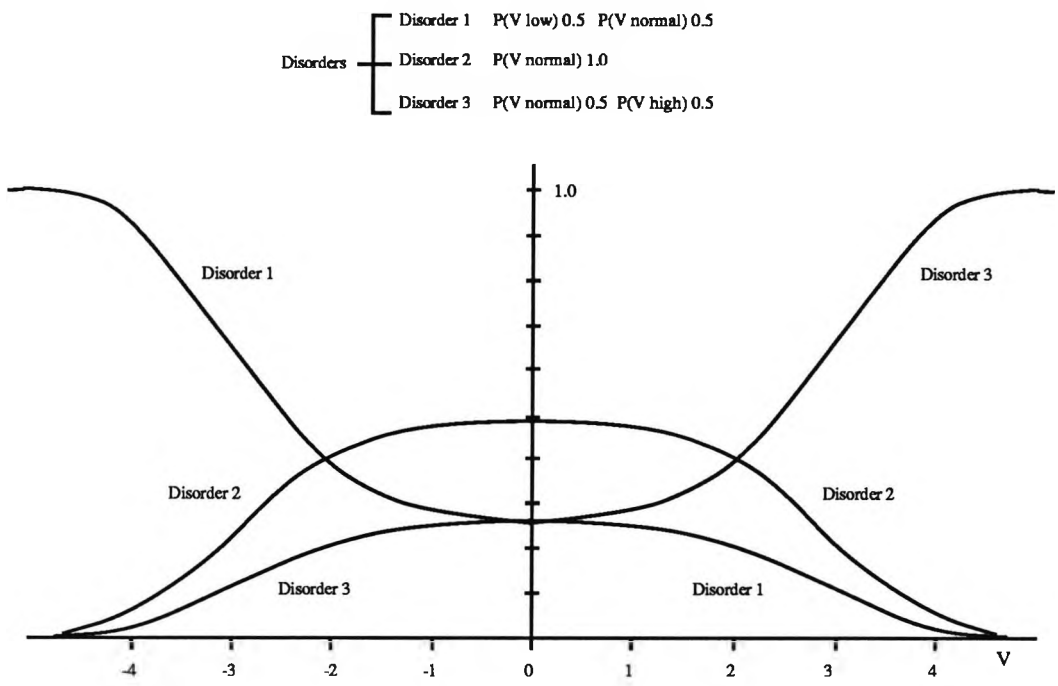


Figure 6.1 Diagnosis Based on a Single Laboratory Data Variable

When V=2.0 belief in Disorders 2 and 3 is equal (0.4) and twice that of Disorder 1 (0.2). Once V has risen to 4.0 belief in Disorder 3 (0.94) is much greater than belief in either Disorder 2 (0.04) or Disorder 1 (0.02).

The behaviour displayed by the system agrees with what is expected intuitively. From the knowledge base it is seen that a normal value of V is expected in all cases of Disorder 2, with cases of Disorder 3 it is equally likely that V will be normal or high and with Disorder 1 V is equally likely to be normal or low. So when V has its mean value (is at its most normal level) it is expected that Disorder 2 is twice as likely as the other Disorders which could equally exhibit low or high V. Belief in Disorder 2 is not exactly twice that of Disorder 1 or 3 because even when a patient presents with a mean value of V there is still a small possibility that it is high or low for that particular individual.

In Section 4.3.4 the value of a laboratory data variable was defined as high if it falls further than 2 standard deviations above the mean and normal if it falls within 2 standard deviations of the mean. So it would be expected that when V=2.0 (ie 2 standard deviations above its mean value) it is equally likely to be high or low. Hence Disorder 2, which always exhibits normal V, and Disorder 3 which could have normal or high V are expected to be equally likely at this point.

This test has shown that the method of data classification used in the system creates a smooth and gradual variation of belief in the diagnostic hypotheses as the data values vary and that the degree of belief assigned to each hypothesis is in accordance with what is intuitively expected.

6.4.3 Performance With Incomplete Data

The performance of the system when presented with an incomplete set of laboratory data was tested using one of the evaluation test cases in Appendix IV (Case 22). The full set of data for this case is:

pH 7.53
pCO_2 5.06 kPa
HCO_3^- 32.70 mmol/L
Base Excess 10.10 mmol/L

and the diagnosis based on these data is:

Uncompensated metabolic alkalosis 0.7823
Respiratory alkalosis & metabolic alkalosis 0.1966

When presented with the pH value only, the system returns a diagnosis of dominant alkalosis. If PCO2 is now input the result is:

Input	pH 7.53 pCO_2 5.06 kPa
Calculated	HCO_3^- 31.324 mmol/L Base Excess 8.074 mmol/L
Diagnosis	Uncompensated metabolic alkalosis 0.782
	Respiratory alkalosis & metabolic alkalosis 0.197

The system has calculated HCO_3^- and Base Excess from pH and pCO_2. The algorithm used by the system calculates HCO_3^- to within 5% of the value obtained from the blood gas analyser; the algorithm for Base Excess is not quite so accurate, giving a discrepency of about 20%. Despite these differences from the real data, the final diagnosis is almost identical since both Base Excess and HCO_3^- are raised well above their normal levels.

The system exhibits different behaviour when the data are presented in a different order. If pCO_2 is input alone the diagnosis is:

pCO_2 5.06 kPa

Uncompensated metabolic alkalosis 0.212
Uncompensated metabolic acidosis 0.212
Measurement error 0.188
Normal blood gases 0.129

None of the hypotheses are particularly strong. A simple uncompensated metabolic disturbance is most likely although it is impossible to distinguish between acidosis and alkalosis. With only one measurement, there is still a possibility that a measurement error has occurred and since the value of pCO_2 is only fairly high the patient may not have an acid-base disturbance. If HCO_3^- is now input:

pCO_2 5.06 kPa
HCO_3^- 32.7 mmol/L

Uncompensated Metabolic Alkalosis 0.623
Measurement error 0.166
Respiratory alkalosis & metabolic alkalosis 0.157

Already, uncompensated metabolic alkalosis appears to be the most likely diagnosis. The hypotheses of normal blood gases and uncompensated metabolic acidosis have been eliminated although the the probability of a measurement error is only slightly reduced. Respiratory alkalosis & metabolic alkalosis has emerged as another hypothesis. Base Excess is now input:

pCO_2 5.06 kPa
HCO_3^- 32.7 mmol/L
Base Excess 10.1 mmol/L

Uncompensated metabolic alkalosis 0.734
Respiratory alkalosis & metabolic alkalosis 0.184

The possibility of a measurement error has now been eliminated and the two final hypotheses are well established.

This experiment has illustrated the difference between the behaviour of the system and the behaviour of a simple decision tree. In the latter case, the supply of a limited set of data will at best result in the top branches of the tree being explored; it is more likely that no conclusions will be reached at all. With the current system, hypotheses at any point of the hierarchy can be activated by a limited data set and as more data are added the focus of attention can shift to any hypothesis. The hierarchy is not searched in the way that a decision tree would be searched; evidence impacts on nodes at any level and its effect is then distributed throughout.

6.4.4 An Example Dialogue

An example of the type of dialogue that can be conducted is set out below using data from the evaluation Case 39. In the transcript the user's input is shown in italic script and the system's output in bold script. The initial diagnosis has been made using the data:

pH 7.01
pCO_2 9.99 kPa
HCO_3^- 19.3 mmol/L
Base Excess -12.6 mmol/L

and the user selects *dialogue* from the main menu of the diagnostic system:

My diagnosis for this patient is respiratory acidosis & metabolic acidosis with severe hypoxaemia.
Respiratory acidosis & metabolic acidosis is consistent with cardiac arrest.

Why?

Respiratory acidosis & Metabolic acidosis was indicated by:
low base excess
low hco3

high pco2

low ph

Severe hypoxaemia was indicated by:

FIO2=21

PO2<5.32

Why didn't you diagnose respiratory acidosis?

Respiratory acidosis was contra-indicated by:

low base excess

low hco3

What is the pCO2?

The pCO2 is 9.9 kPa by measurement.

How did pCO2 affect the diagnosis?

The effect of pCO2 on my belief in possible diagnoses was as follows:

High hco3 with pco2 increased

Low hco3 with pco2 ruled out

Measurement error decreased

Metabolic acidosis & metabolic alkalosis decreased

Mixed metabolic acidosis ruled out

Normal blood gases ruled out

Partially compensated metabolic acidosis ruled out

Partially compensated metabolic alkalosis increased

Respiratory acidosis & metabolic acidosis ruled out

Respiratory alkalosis & metabolic alkalosis ruled out

Respiratory alkalosis ruled out

Respiratory acidosis increased

Uncompensated metabolic acidosis ruled out

Uncompensated metabolic alkalosis ruled out

What if pCO2 was 3.7?

My diagnosis for this patient would be partially compensated metabolic acidosis with severe hypoxaemia.

The dialogue above has illustrated the manner in which the system can be interrrogated in order to obtain explanations for its conclusions. The explanations are in terms of the effects of evidence on the hypotheses considered - there is no deeper knowledge involved.

6.4.5 Evaluation of System Speed

The speed of the diagnostic system using the three different control strategies presented in Section 5.2 was evaluated using ten evaluation cases (15 - 24) - the results are shown in Table 6.4. The system takes 40 - 50 seconds to diagnose a case depending on its complexity. At the time of the evaluation the system was running as interpreted code and speed could be increased by a factor of 5 - 10 if the code was compiled. An automatic blood gas analyzer takes about 30 - 60 seconds to process a sample and in this context the overall speed of the system is quite acceptable.

Interestingly, the first trigger control strategy (which executes the first knowledge source triggered) is slightly faster than the mixed schedule (which executes on each cycle all pending knowledge source activations from the first triggered knowledge source). This is because additional knowledge source activations in the latter case are found by backtracking which takes longer than repeating the cycle the necessary number of times.

The times taken for the full schedule strategy are not more than 15% greater than the first trigger times which suggests that the overhead in implementing a more sophisticated scheduler would not be excessive.

CASE	FIRST TRIGGER	MIXED SCHEDULE	FULL SCHEDULE
15	41.36	41.63	46.46
16	43.55	43.88	49.54
17	41.47	41.79	47.90
18	40.64	40.81	45.76
19	43.66	44.00	52.51
20	41.20	41.47	47.35
21	43.50	43.66	46.47
22	41.36	41.53	44.99
23	52.46	53.00	59.43
24	45.80	46.08	51.47

Table 6.4 Time for Diagnosis on Ten Cases Using Different Control Strategies. (Timings are in seconds)

6.5 Evaluation of a Knowledge-Based System for Blood Gas Analysis

6.5.1 Introduction

The blood gas analysis system was evaluated using a random set of 51 retrospective cases gathered from records of patients in the ICU of the West Middlesex Hospital and 9 cases selected from the literature (so that some of the more unusual diagnoses featured in the evaluation). These 60 cases were transcribed to the standard format shown in Figure 6.2 and presented to the expert responsible for the knowledge base, a group of three senior clinicians (senior registrars) and a group of three junior clinicians (senior house officers).

CASE 1

CLINICAL DIAGNOSIS Diabetic

pH 7.05
pCO2 ..... 1.596 kPa
HCO3-..... 5 mmol/L
BE ..... -30 mmol/L
pO2 .....14.36 kPa

CLINICAL DATA:
age 17
sex female
Kussmaul breathing

Please check ONE interpretation for acid-base balance and ONE for hypoxaemic state:

Acid-Base Disorder	**Hypoxaemic State**
[ ] Normal blood gases	[ ] Adequate O2 tension
[ ] Uncompensated metabolic acidosis	[ ] Mild hypoxaemia
[ ] Partially compensated metabolic acidosis	[ ] Moderate hypoxaemia
[ ] Compensated metabolic acidosis	[ ] Severe hypoxaemia
[ ] Uncompensated metabolic alkalosis	[ ] Corrected hypoxaemia
[ ] Partially compensated metabolic alkalosis	[ ] Uncorrected hypox
[ ] Compensated metabolic alkalosis	[ ] Hyperoxaemia
[ ] Uncompensated respiratory acidosis	
[ ] Partially compensated respiratory acidosis	
[ ] Compensated respiratory acidosis	
[ ] Uncompensated respiratory alkalosis	
[ ] Partially compensated respiratory alkalosis	
[ ] Compensated respiratory alkalosis	
[ ] Respiratory acidosis & metabolic acidosis	
[ ] Respiratory acidosis & metabolic alkalosis	
[ ] Respiratory alkalosis & metabolic acidosis	
[ ] Respiratory alkalosis & metabolic alkalosis	

Figure 6.2 Form for Gathering Evaluation Results

For the groups of senior and junior clinicians the evaluation was split into three sets of 20 cases and one set was given to each clinician. In this way a representative diagnosis was obtained for each case from a senior and a junior clinician but each participant was only required to diagnose 20 cases. The details of each test case and a summary of the diagnoses given is included as Appendix IV. The next two sections present an analysis and discussion of the evaluation results.

6.5.2 Analysis of Results

The system was evaluated only for its diagnosis of acid-base disorders because the hypoxaemic state was decided using simple ranges of pO_2 - an evaluation would only reveal whether the clinicians used the same classification ranges as the system. A summary of the diagnoses made by the system and the clinicians is shown in Table 6.5.

DIAGNOSIS	SYSTEM	EXPERT	SENIOR	JUNIOR
Uncomp met acid	1	5	2	3
Part comp met acid	6	5	9	12
Comp met acid	0	4	2	2
Met acid	3	-	-	-
Uncomp met alk	4	5	5	2
Part comp met alk	2	2	2	1
Comp met alk	0	0	0	1
Met alk	0	-	-	-
Uncomp resp alk	6	5	5	10
Part comp resp alk	3	4	6	2
Comp resp alk	4	4	3	4
Resp alk	4	-	-	-
Uncomp resp acid	4	5	6	4
Part comp resp acid	1	1	1	2
Comp resp acid	0	1	1	0
Resp acid	1	-	-	-
Resp acid & met alk	1	0	0	2
Resp alk & met alk	3	0	0	3
Resp alk & met acid	0	3	2	3
Resp acid & met acid	11	10	10	7
Normal blood gases	4	6	6	2
Low HCO3 with pCO2	2	-	-	-
Total	60	60	60	60

Table 6.5 Summary of Diagnoses of Acid-Base Disorders

	EXPERT	SENIOR	JUNIOR	SYSTEM
EXPERT		33	24	33 (39)
SENIOR	55%		23	32 (38)
JUNIOR	40%	37%		26 (35)
SYSTEM	55% (65%)	53% (63%)	43% (58%)	

Table 6.6 Agreement on Diagnosis

The system diagnosed 3 cases of respiratory alkalosis & metabolic alkalosis and 1 case of respiratory acidosis & metabolic alkalosis; neither the expert nor the senior clinicians made any diagnoses of these two disorders. In 10 cases the system made a diagnosis at an intermediate level of the hypothesis hierarchy (metabolic acidosis, respiratory acidosis, respiratory alkalosis or low HCO3 with pCO2).

The relative accuracy of the system, the development expert, the senior clinicians and the junior clinicians is shown in Table 6.6. The set of possible diagnoses made available to the clinicians (Figure 6.2) comprised only the leaf nodes in the hypothesis hierarchy of the system which could (and did) make diagnoses at intermediate levels of the hierarchy in cases where it could not distinguish between a set of leaf nodes. There were ten cases in which the system behaved in this way. The main figures for the accuracy of the system in Table 6.6 include only those cases in which the system's top ranking diagnosis was identical to the clinician's; the figures in parentheses include cases in which the clinician's diagnosis was subsumed by the less specific diagnosis of the system (these will be referred to as cases of partial agreement). The system showed a similar level of agreement with both the expert and senior clinicians - 55% (65%) agreement with the expert, 53% (63%) agreement with the senior clinicians. The agreement between the expert and senior clinicians themselves was also 55%. Thus it appears, in terms of a simple accuracy test, that the system operates at the same level as the expert and senior clinicians.

The diagnosis of the junior clinicians agreed with the expert and senior clinicians in 40% and 37% of cases respectively. The junior clinicians agreed with the system in 43% (58%) of cases. As was to be expected, there was a lower level of agreement between the junior clinicians and either the expert or senior clinicians than there was between the expert and senior clinicians themselves. There was a corresponding drop in the agreement between junior clinicians and the system and although this level of agreement was a little higher than

the agreement with the other clinicians, the difference was not large enough to be significant.

Of the 33 cases in which the expert and senior clinicians' diagnoses concurred, the system agreed completely in 25 (75.8%) and partially in another 3. There were thus only 5 cases (15.2%) in which the system disagreed. These 5 cases are analyzed below:

Case 6 The clinicians' diagnosis was respiratory alkalosis & metabolic acidosis, the system's was partially compensated metabolic acidosis which was the diagnosis of the junior clinician.

Case 11 The clinicians' diagnosis was normal blood gases, the system's was uncompensated metabolic alkalosis which is characterized by normal pCO_2 and high pH, HCO_3^- and Base Excess. In fact the latter three were all raised to just over 2 standard deviations from their normal values whilst pCO2 was slightly low (within 1 standard deviation of the mean) which was enough to make the diagnosis for the system.

Case 21 The diagnosis made by the clinicians was compensated respiratory acidosis, the system had respiratory acidosis & metabolic alkalosis which was the diagnosis of the junior clinician. The system ruled out compensated respiratory acidosis because the relationship between ΔpH and ΔpCO_2 did not hold.

Case 37 The clinicians' diagnosis was uncompensated metabolic alkalosis whilst the system had respiratory alkalosis & metabolic alkalosis (0.5377) as slightly more likely than uncompensated metabolic alkalosis (0.4478). The junior clinician had respiratory alkalosis & metabolic alkalosis. Overall, the system's conclusion would seem to agree well with the clinicians.

Case 43 The clinicians had compensated respiratory alkalosis, the system had respiratory alkalosis (0.397) preferred to compensated respiratory alkalosis (0.2793) or respiratory alkalosis & metabolic acidosis (0.2147). The junior clinician had partially compensated respiratory alkalosis. Once again, although the system's top ranking diagnosis did not agree with the two experienced clinicians, the overall performance of the system seems quite reasonable.

Of the 27 cases in which the expert and senior clinicians did not give the same diagnosis, the system agreed completely with the expert in 8 cases and partially in another and also in 8 completely and 1 partially with the senior clinicians. There were also 2 cases (17 and 58) in which the system was in partial agreement with both the diagnoses of the clinicians. There were thus only 7 cases in which the expert , senior clinicians and system all had

different diagnoses (8, 9, 19, 25, 31, 45 and 52). In two of these (8 and 45) the system agreed with the junior clinician, in a third (52) the closely ranked alternative diagnosis agreed with both the expert and junior clinicians and in another (19) the alternative diagnosis agreed with the senior and junior clinicians. The remaining 3 cases are analyzed below:

Case 9 The system seems fairly confused on this case, and so do the clinicians! The system gives 4 possible diagnoses, all having fairly low probability (0.1046 - 0.3158). The second ranked diagnosis, normal blood gases (0.1817) agreed with the expert. The system ruled out the diagnoses of the senior and junior clinicians because of the relationship between ΔpH and ΔpCO_2.

Case 25 The system's preferred diagnosis of respiratory alkalosis (0.7731) was in partial agreement with the junior clinician. The diagnoses of the expert and senior clinicians were ruled out by the system because of the relationship between ΔpH and ΔpCO_2.

Case 31 The system gave three possible diagnoses of which the third ranked agreed with the senior and junior clinicians. The expert's diagnosis of compensated metabolic acidosis was ruled out by the relationship between ΔpH and ΔpCO_2.

From the analysis above it can be seen that the system disagreed with the clinician's diagnoses of compensated disorders in cases 9, 21, 25 and 31 because of the relationship between ΔpH and ΔpCO_2. There are three ways in which the system's performance could be improved in this area. Firstly, the knowledge base could be altered to make the limits on ΔpH and ΔpCO_2 less stringent. Secondly, as was mentioned in Section 5.4.4, the term ΔV is interpreted by the system as *the change in variable V from its mean value* (in a reference population) whereas it may be more correct to interpret it as *the change in V from its last measured value* (in the same patient). Thirdly, the relationships are evaluated as either true or false and in the latter case the hypothesis is then completely ruled out. It might be better to interpret the set {=,<,>,=<,>=) as fuzzy conditions so that the weight of evidence contributed by the relationship depends on the *degree* to which the left hand side is less than, greater than or equal to the right hand side.

Tables 6.7 and 6.8 show the performance of the system broken down by diagnosis. In cases where the system made a diagnosis at an intermediate level of the hypothesis hierarchy which subsumed the clinician's diagnosis, a true +ve was recorded. When it did not subsume the clinician's diagnsosis a false +ve was recorded for each of the descendent leaf node diagnoses.

DISORDER	CASES	TRUE +ve	FALSE -ve	TRUE -ve	FALSE +ve	SENSITIVITY	SPECIFICITY	DIAGNOSTIC INDEX
Uncomp met acid	5	2	3	54	1	0.40	0.98	1.38
Part comp met acid	5	4	1	52	3	0.80	0.95	1.75
Comp met acid	4	0	4	54	2	0.00	0.96	0.96
Uncomp met alk	5	3	2	54	1	0.60	0.98	1.58
Part comp met alk	2	2	0	58	0	1.00	1.00	2.00
Uncomp resp alk	5	5	0	51	4	1.00	0.93	1.93
Part comp resp alk	4	2	2	51	5	0.50	0.91	1.41
Comp resp alk	4	2	2	52	4	0.50	0.93	1.43
Uncomp resp acid	5	5	0	55	0	1.00	1.00	2.00
Part comp resp acid	1	0	1	58	1	0.00	0.98	0.98
Comp resp acid	1	0	1	59	0	0.00	1.00	1.00
Resp alk & Met acid	3	0	1	555	2	0.00	0.96	0.96
Resp acid & Met acid	10	9	1	48	2	0.90	0.96	1.86
Normal Blood Gases	6	4	2	54	0	0.66	1.00	1.66

Table 6.7 Development Expert as Gold Standard

DISORDER	CASES	TRUE +ve	FALSE -ve	TRUE -ve	FALSE +ve	SENSITIVITY	SPECIFICITY	DIAGNOSTIC INDEX
Uncomp met acid	2	2	1	57	1	1.00	0.98	1.98
Part comp met acid	9	5	5	48	3	0.55	0.94	1.49
Comp met acid	2	1	2	57	1	0.50	0.98	1.48
Uncomp met alk	5	2	3	53	2	0.40	0.96	1.36
Part comp met alk	2	1	1	57	1	0.50	0.98	1.48
Uncomp resp alk	5	3	4	50	5	0.60	0.91	1.51
Part comp resp alk	6	3	3	52	2	0.50	0.96	1.46
Comp resp alk	3	1	2	54	3	0.33	0.95	1.28
Uncomp resp acid	6	5	2	54	0	0.83	1.00	1.83
Part comp resp acid	1	1	0	59	0	1.00	1.00	2.00
Comp resp acid	1	0	1	59	0	0.00	1.00	1.00
Resp alk & Met acid	2	0	2	57	1	0.00	0.98	0.98
Resp acid & Met acid	10	10	0	49	1	1.00	0.98	1.98
Normal Blood Gases	6	4	2	54	0	0.66	1.00	1.66

Table 6.8 Senior Clinician as Gold Standard

The only significant difference between the pattern of diagnoses by the expert and senior clinicians is that the senior clinicians tended to make more diagnoses of partially compensated metabolic acidosis than the expert whose diagnoses of metabolic acidosis were split fairly evenly between compensated, uncompensated and partially compensated. The system performed best on cases of respiratory acidosis & metabolic acidosis and worst on compensated disorders (discussed above). The system did not make any diagnoses of respiratory alkalosis & metabolic acidosis which is reflected in a low diagnostic index. There was in fact only one case where the clinicians agreed on a diagnosis of respiratory alkalosis & metabolic acidosis and this has been discussed above (Case 6). Overall, it is

difficult to draw firm conclusions in diagnostic categories where only one or two diagnoses were made and further investigation of these would be fruitful.

6.6 Summary

This Chapter has discussed the problems associated with the evaluation of knowledge-based systems in medicine, in particular the method of evaluation, the gathering of test cases and the analysis of results. The knowledge-based system described in Chapters 4 and 5 was evaluated in a series of bench tests and by a clinical study in the domain of acid-base balance which showed that the system performs well in comparison with expert clinicians. In the next Chapter the utility of the diagnostic knowledge-based system and the FRAMEBUILDER knowledge editing environment as a set of domain independent tools is evaluated through the development of a system for the diagnosis of hyperlipidaemia.

CHAPTER SEVEN

EVALUATION IN A SECOND APPLICATION DOMAIN

7.1 Introduction

The knowledge-based diagnostic system and the FRAMEBUILDER knowledge editing environment were originally designed to be used for the interpretation of blood gas analysis results. It was recognized during the design of these systems that they could be applied for other, similar problems of interpretation and they were therefore developed to be domain independent. In order to evaluate the applicability of the systems in other domains, a knowledge-based system was developed for the diagnosis of hyperlipidaemia. The diagnosis is based on an evaluation of the family history of heart disease, peripheral vascular disease and hyperlipidaemia, measurement of cholesterol, triglyceride, LDL (low density lipoproteins) and HDL (high density lipoproteins) and the patient's history and examination. An overview of lipid physiology and hyperlipidaemia disorders is given in Appendix V.

The remaining sections in this chapter describe the knowledge acquisition process using the FRAMEBUILDER tool and the performance of the diagnostic system for some example cases using real clinical data.

7.2 Knowledge Acquisition

7.2.1 Introduction

The knowledge required to build a knowledge-based system for the diagnosis of hyperlipidaemia and the assessment of the risk of coronary heart disease was acquired from an expert clinical biochemist during three knowledge elicitation sessions, each lasting 2-3 hours. From the outset, the knowledge acquisition process was fundamentally different from the application in blood gas analysis since in the latter case the knowledge engineer had acquired considerable background knowledge of the domain before knowledge elicitation sessions commenced. At the start of the first session for hyperlipidaemia diagnosis, the knowledge engineer was completely ignorant of the domain.

The following three sections describe the three knowledge acquisition sessions that took place at the West Middlesex Hospital.

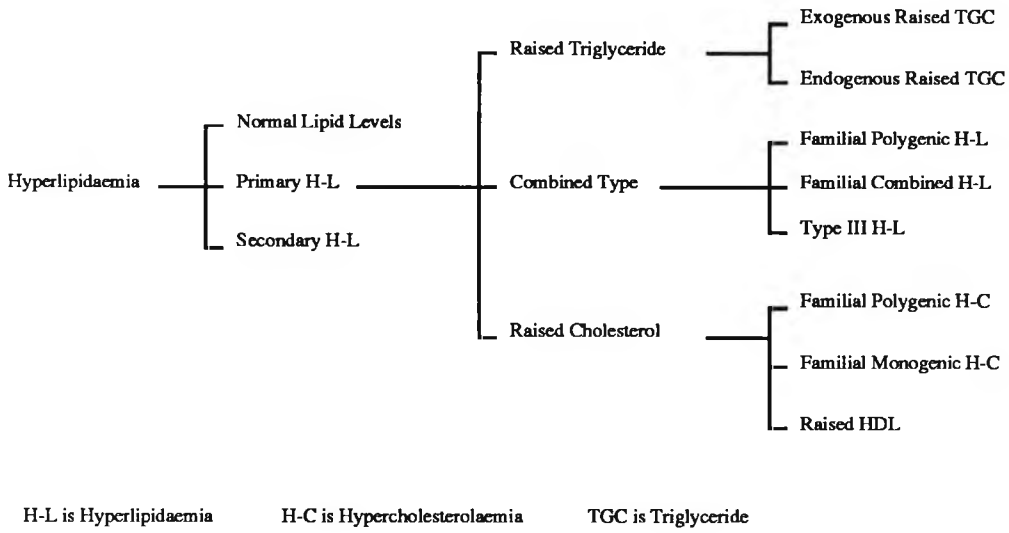


Figure 7.1 Hypothesis Hierarchy for Hyperlipidaemia

7.2.2 First Session

The first task in the first session was for the expert to give a broad outline of the domain: the principal diagnoses to be made and the relevant patient observations. As in the case of acid-base disorders, the initial structuring of the domain was best achieved using a paper and pen, not the computer knowledge acquisition environment.

Once an initial hypothesis hierarchy had been determined, it was entered into the knowledge base using FRAMEBUILDER. The final hierarchy for Hyperlipidaemia is shown in Figure 7.1 (in fact the nodes for normal lipid levels and raised HDL were added in the second session, when some changes to the names of the nodes were also made). At the first level of the hierarchy, hyperlipidaemia is divided into primary hyperlipidaemia (cases in which it is the primary physiological disturbance), secondary hyperlipidaemia (where it is caused by an underlying disease) and normal lipid levels (*ie* no hyperlipidaemia present - this node is included to ensure the condition of exhaustiveness on the nodes of the hierarchy).

Thyroid Disease

Liver Disease

Alcoholism

Renal Disease

Obesity

Diabetes

Table 7.1 Disease States for Secondary Hyperlipidaemia

The diagnosis of secondary hyperlipidaemia is made by the observation of one of the diseases shown in Table 7.1. These were entered in the knowledge base as diseases; their presence was specified as evidence for secondary hyperlipidaemia and their absence as evidence for primary hyperlipidaemia. Hence if one of the diseases is present, the only possible diagnoses are normal lipid levels or secondary hyperlipidaemia. The system could not be used for critiquing because diseases were specified as evidence for or against disorder hypotheses. An alternative approach would have been to specify hyperlipidaemia as a disorder in the frames of the primary disease states, whereby the determination of primary or secondary hyperlipidaemia would come about as a result of the critiquing process.

The second level of the hierarchy in Figure 7.1 represents the different categories of primary hyperlipidaemia. The three hypotheses at this level are distinguished by levels of plasma cholesterol and triglyceride. It was found that a three interval classification was not adequate in this instance and so the method presented in Section 4.3.3 could not be used. Instead two disorder classes were defined - Cholesterol Level and Triglyceride Level - with the simple hierarchies shown in Figure 7.2. The appropriate levels of cholesterol and triglyceride were entered as relationship evidence for each node. The nodes such as *safe cholesterol* were then added in the history slots of hyperlipdaemia disorders.

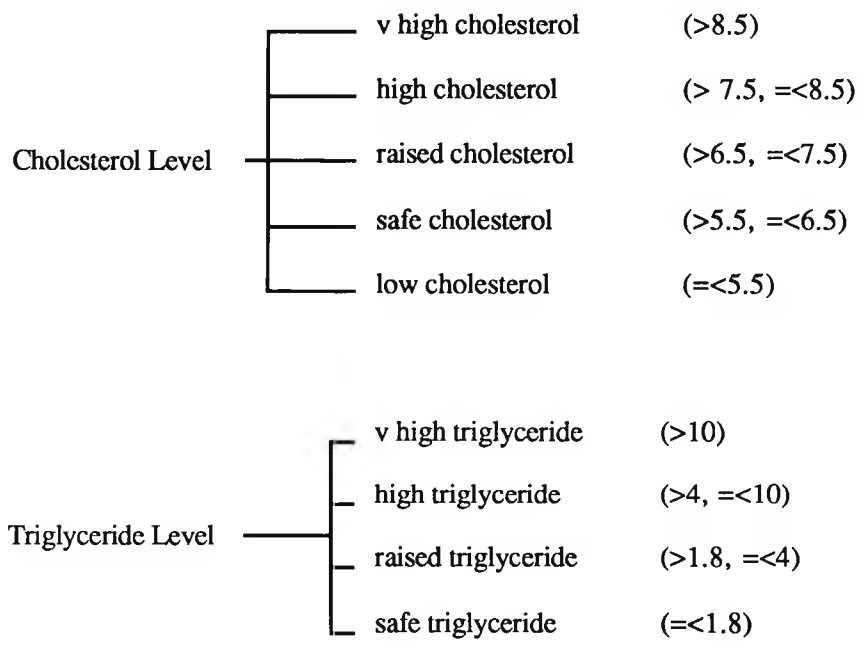


Figure 7.2 Hierarchies for Cholesterol and Triglyceride Levels

Abdominal Pain present absent	Lipid Electrophoresis shows broad β band shows no β band
Stored Plasma Appearance clear with creamy layer cloudy with creamy layer cloudy clear	Xanthoma tuberous linear palmar erruptive tendon absent
Pancreatitis present absent	Peripheral Vascular Disease present absent

Table 7.2 Summary of Signs and Symptoms for Hyperlipidaemia

The remainder of the first session was spent in determining the useful observations for diagnosis of hyperlipidaemia. A summary of these is given in Table 7.2 and an explanation can be found in Appendix V.

7.2.3 Second Session

It was decided at the first session to assess each patient for the risk of Coronary Heart Disease (CHD). In the second session a CHD risk factor was defined, based on the three main contributors to CHD risk - hypertension, elevated plasma cholesterol and cigarette smoking. These three factors were used by the Framingham Heart Study Group (they also used sex, age, ECG findings and HDL level) to produce an algorithm for calculation of CHD risk in terms of the number of deaths predicted over a ten year period. Such an algorithm was not applicable for use in this system and so a heuristic risk factor was defined.

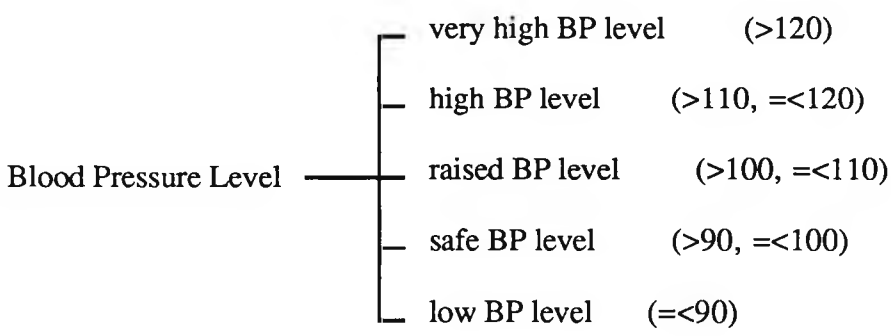


Figure 7.3 Blood Pressure Level

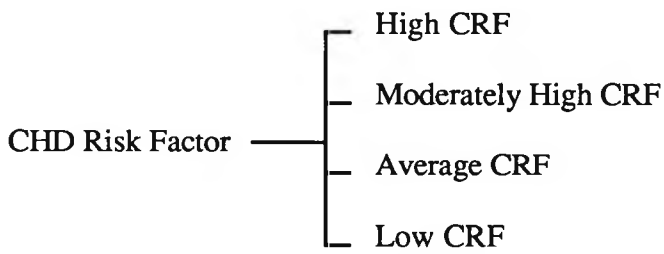


Figure 7.4 Coronary Heart Disease Risk Factor

Smoking habits were added as an item of patient history and blood pressure level was specified as a disorder as shown in Figure 7.3 (the category *safe BP level* is not well named and would be better named as *slightly raised,* or even *unsafe BP level*). The CHD risk factor itself was defined as low, average, moderate or high (Figure 7.4) and evidence of smoking habits, blood pressure and cholesterol level were specified for each definition.

Also in the second session, the observations defined in the first session were assigned to hypotheses in the hyperlipidaemia hierarchy. An important aspect of the diagnosis that was addressed in the second session is the consideration of the patient's family history of hyperlipidaemia, ischaemic heart disease and peripheral vascular disease. This knowledge was elicited from the expert using the grids shown in Figures 7.5 and 7.6 and was then transfered to the knowledge base using initial probability assignments of 0.0, 0.5 or 1.0 as appropriate.

7.2.4 Third Session

In the third session some hypothetical cases were run through the system by the domain expert and adjustments were made to the knowledge base. Most of these involved the probability assignments for the family history that had been assigned roughly in the previous session.

	FM H-C	FP H-C	FC H-L	FP H-L
Cholesterol	√	√	√	-
Triglyceride	-	-	√	-
Both	-	-	√	√

FM-HC is Familial Monogenic Hypercholesterolaemia
FP H-C is Familial Polygenic Hypercholesterolaemia
FP H-L is Familial Polygenic Hyperlipidaemia
FC H-L is Familial Combined Hyperlipidaemia

Figure 7.5 Elicitation Grid for Family History

	FM H-C	FP H-C	FC H-L	FP H-L
Ischaemic Heart Disease	++ / +	+ / -	++	+
Peripheral Vascular Disease	-	-	++	+
Both	-	-	++	+

+ is for relatives aged 50-60 ++ is for relatives aged below 50

FM-HC is Familial Monogenic Hypercholesterolaemia
FP H-C is Familial Polygenic Hypercholesterolaemia
FP H-L is Familial Polygenic Hyperlipidaemia
FC H-L is Familial Combined Hyperlipidaemia

Figure 7.6 Elicitation Grid for IHD/PVD

Several errors in the knowledge base were corrected, the attribute *tuberous* was added to the symptom *xanthoma* and *peripheral vascular disease* was added as an item of patient history (distinct from peripheral vascular disease in the patient's family history). At the end of the third session the knowledge base was fixed (Appendix VI) and the system was ready to undergo the evaluation described in the next section.

7.3 Evaluation

The system was evaluated using 38 randomly selected cases from the West Middlesex Hospital and two hypothetical cases that were included to cover the rather unusual diagnoses of exogenous and endogenous raised triglyceride. Details of the cases are given in Appendix VII. The 40 cases were presented to the system and to the expert who developed the knowledge base; a summary of the diagnoses made is shown in Table 7.3.

In 6 cases the system made a diagnosis at an intermediate level of the hierarchy (Raised Cholesterol or Combined Hyperlipidaemia) and 5 of these subsumed the expert's diagnosis. There was complete agreement between the system and the expert in 25 cases (62.5%) and disagreement in 10 cases (25%). In 7 of these cases the system gave the expert's diagnosis as its second ranked alternative. The remaining 3 cases are analyzed below.

DIAGNOSIS	EXPERT	SYSTEM	EXPERT & SYSTEM
Endogenous Raised Triglyceride	1	1	1
Exogenous Raised Triglyceride	1	1	1
Raised Cholesterol	-	1	-
Familial Polygenic Hypercholesterolaemia	6	8	5
Familial Monogenic Hypercholesterolaemia	12	8	7
Raised HDL	3	2	2
Combined Type Hyperlipidaemia	-	5	-
Familial Polygenic Hyperlipidaemia	4	6	1
Familial Combined Hyperlipidaemia	11	6	6
Type III Hyperlipidaemia	1	1	1
Normal	1	1	1
Total	40	40	25

Table 7.3 Summary of Diagnoses of Hyperlipidaemia

Case 7 The system had familial polygenic or familial combined hyperlipidaemia as equally ranked alternative diagnoses, the expert had familial monogenic hypercholesterolaemia. The Triglyceride level (1.9 mmol/L) is only slightly raised above the safe level (1.8 mmol/L) which is why the expert made a diagnosis of hypercholesterolaemia. Because the system uses hard limits to decide Triglyceride levels, the possibility that a level of 1.9mmol/L was in the safe range was completely ruled out.

Case 19 The system made its diagnosis of Familial Polygenic Hyperlipidaemia because of the family history of heart disease in the age range 50 - 60 years.The expert's diagnosis was Familial Combined Hyperlipidaemia which would be indicted if the family history of heart disease was for ages below 50.

Case 27 The system had Familial Mongenic Hypercholesterolaemia because of the familily history of heart disease below the age of 50. The expert had Familial Polygenic Hypercholesterolaemia.

DIAGNOSIS	CASES	TRUE +ve	FALSE -ve	TRUE -ve	FALSE +ve	SENSITIVITY	SPECIFICITY	DIAGNOSTIC INDEX
Endogenous Raised Triglyceride	1	1	0	39	0	1.00	1.00	2.00
Exogenous Raised Triglyceride	1	1	0	39	0	1.00	1.00	2.00
Familial Polygenic Hypercholesterolaemia	6	5	1	31	3	0.83	0.91	1.74
Familial Monogenic Hypercholesterolaemia	12	8	4	28	0	0.67	1.00	1.67
Raised HDL	3	2	1	37	0	0.67	1.00	1.67
Familial Polygenic Hyperlipidaemia	4	4	0	31	5	1.0	0.86	1.86
Familial Combined Hyperlipidaemia	11	7	4	28	1	0.64	0.97	1.61
Type III Hyperlipidaemia	1	1	0	38	1	1.0	0.97	1.97
Normal	1	1	0	39	0	1.0	1.00	2.00

Table 7.4 Breakdown of Results by Diagnosis

The diagnoses of the system can also be analyzed by using the expert's diagnosis as a gold standard and calculating the sensitivity and specificity of the system for each diagnostic category. Such an analysis is shown in Table 7.4 where the diagnoses made by the system at intermediate levels of the hypothesis hierarchy were treated in the same way as in Section 6.5.2 (true +ve if they subsumed the expert's diagnosis and false negative for each subsumed diagnosis if none of these agreed with the expert). It can be seen that the system had a tendency to under diagnose the monogenic disorders (Familial Monogenic Hypercholesterolaemia and Familial Combined Hyperlipidaemia) and to over diagnose the polygenic disorders (Familial Polygenic Hypercholesterolaemia and Familial Polygenic Hyperlipidaemia).

7.4 Discussion

The purpose of developing the hyperlipidaemia system was to evaluate the FRAMEBUILDER and diagnostic systems as a set of tools. This evaluation can be considered successful in that a system was developed in a very short time and was seen to perform well on a set of 40 tests cases. The evaluation in this second domain also served to highlight several areas in which the two systems can be extended or improved.

The hyperlipidaemia system is slow (taking over two minutes to diagnose a typical case) because it has to make two complete bottom-to-top traversals of the physiological blackboard. On the first traversal the pseudo disorders of cholesterol, triglyceride and blood pressure levels are diagnosed and on the second traversal they are asserted as evidence on the raw data level of the blackboard so that diagnoses of hyperlipidaemia and cardiac risk factor can be made. Moreover, the diagnoses of cholesterol, triglyceride and blood pressure levels are not entirely satisfactory because they are made by hard limits on the relevant variables which are set as relationship evidence. Most of the inaccuracies of the system in the evaluation cases can be attributed to this method of classification.

Both the problems outlined above would be solved if the method of classification introduced in Section 4.3.3 was extended so that more than three intervals could be defined for the classification. FRAMEBUILDER could then be extended so that for any laboratory data variable an arbitrary number of intervals could be defined and given appropriate labels. Another extension would allow distributions other than the Gaussian distribution to be used.

For classification into an arbitrary number of intervals, if class N is defined for values of a data variable V that are altered in the range $[\alpha,\beta]$ from the healthy measurement of an individual, the probability that an observed value X is classified as N is $\phi(X-\alpha) - \phi(X-\beta)$ for a particular patient, where ϕ is the probability distribution of the reference population. Thus the classifiaction in Section 4.3.3 is a special case in which three intervals are defined:

low	$\alpha = -\infty, \beta = -2\sigma$
normal	$\alpha = -2\sigma, \beta = 2\sigma$
high	$\alpha = 2\sigma, \beta = \infty$

where σ is the standard deviation of the reference population.

7.5 Summary

The diagnostic system and the FRAMEBUILDER knowledge editing environment were evaluated as a set of domain independent tools by applying them in the domain of Hyperlipidaemia. A diagnostic system was developed in a short time and was shown to perform well on a set of test cases. The evaluation identified a number of areas in which the tools could be extended or improved.

CHAPTER EIGHT

CONCLUSIONS

In the opening chapter of this thesis, it was argued that the rapid increase in the amount of data available to clinicians in intensive care medicine will lead them to seek the assistance of computers for data analysis and decision making. Although conventional computing techniques have been applied to medical diagnosis and patient management, the ability of knowledge-based systems to handle uncertain or incomplete data and to explain the basis of their conclusions makes them a more attractive means of achieving these goals.

The applicability of knowledge-based techniques to the interpretation of data in intensive care medicine has been investigated through the design, implementation and evaluation of a system for the diagnosis of disorders of acid-base balance and hypoxaemic state, based on the interpretation of data available from an automatic blood gas analyser. The original objectives for the system, as stated in the first chapter, were:

(1) to make it easy to use and to keep interaction between computer and clinician to a minimum.
(2) to combine different methods of knowledge representation and reasoning within it.
(3) to provide detailed explanation of its conclusions and of the domain of acid-base balance in general.
(4) to make its knowledge easily accessible to clinicians for review and update.

A review of existing knowledge-based systems in medicine was made in Chapter 2, where the progression in methods of implementation was traced through three generations of systems. The early systems used *ad hoc* methods of reasoning with surface level knowledge and in the second generation systems the representation of deeper level knowledge of the physiological causes of disease was explored. A new generation of systems is emerging which seek to use more rigorous methods of reasoning with surface knowledge so that deep knowledge need only be used to provide explanations or to clarify difficult cases.

Methods of knowledge representation, manipulation and control were examined in Chapter 3 and in Chapters 4 and 5 a number of these were developed further and applied in the implementation of a knowledge-based system. The first of the objectives stated above was achieved by devoting a large part of the development time to creating an attractive user interface and by ensuring that an initial diagnosis could be made using only the data

available from a blood gas analyser, so that these could in future be passed on-line directly to the system.

A frame-based representation of knowledge, a method of data classification and a scheme for impacting the effect of evidence on a hierarchy of hypotheses were combined within a blackboard control architecture to achieve the second of the objectives. The dual panelled blackboard architecture has provided an excellent control structure for the data classification and evidence handling procedures and has proved to be a useful model of the distinction made in clinical practice between the diagnosis of physiological disorders and underlying disease states.

The third objective is the only one of the four that has not been fully realized. The explanations provided by the system are in terms of the effects of the observed evidence on the various diagnostic hypotheses and there is no deeper knowledge involved. The incorporation of such knowledge would be one way in which the system could usefully be extended.

In order to achieve the final objective, a knowledge editing tool, FRAMEBUILDER, was developed. This tool allows all the knowledge used in making decisions to be inspected by clinicians and has also proved invaluable as an aid to knowledge acquisition. Because the diagnostic system assumes nothing about the knowledge base constructed using FRAMEBUILDER, the two systems can be used together as a toolset for the constuction of knowledge-based other fields of medicine in which diagnosis is based on the interpretation of laboratory data.

Chapters 6 and 7 described the evaluation of the system, assessed its strengths and weaknesses and suggested ways in which it could be developed further. In Chapter 6 an evaluation of the knowledge-based system in the domain of acid-base balance showed that it performs at a level comparable with that of an expert clinician. In Chapter 7 the set of tools was applied to the construction of a knowledge-based system for the diagnosis of Hyperlipidaemia , demonstrating that it was possible to produce, in a short space of time, a system that performed well in an evaluation test. More importantly, the application of the tools to a second domain identified several areas in which further development could be made.

A fully operational system in the domain of acid-base balance would be connected directly to an automatic blood gas analyser and incorporate a graded evaluation of data relationships considered as evidence. More generally, important improvements would be the expansion of the data classification scheme to accommodate more than three intervals and re-implementation in a faster language than Prolog (C for instance).

This work has contributed to the field of Systems Science...

by developing methods of data classification and evidence handling and implementing them using a blackboard control structure.

by using a dual panelled blackboard architecture to model the diagnosis of physiological disorders and underlying disease states.

and to the field of Medicine...

by applying knowledge-based techniques to produce systems that display expert-level performance in the domains of acid-base balance and hyperlipidaemia.

by developing a knowledge editing tool that can be used with the systems already created or for the construction of new systems in other fields of medicine.

Overall, this work has indicated that by incorporating knowledge-based techniques in a set of easily used tools, clinicians will be able to develop their own consultation systems and that in this way such systems will enter into widespread use throughout many fields of medicine.

REFERENCES

Adams ID, Chan M, Clifford PC, Cooke WM, Dallos V, De Dombal FT, Edwards MH, Hancock DM, Hewett DJ, McIntyre N, Somerville PG, Spiegelhalter DJ, Wellwood J, Wilson DH. Computer aided diagnosis of acute abdominal pain: a multicentre study. British Medical Journal, 1986; 293: 800-804.

Adlassnig K-P. A fuzzy logical model of computer-assisted medical diagnosis. Meth Inform Med, 1980; 19: 141-148.

Adlassnig K-P, Kolarz G, Scheithauer W. Present state of the medical expert system CADIAG-2. Meth Inform Med, 1985; 24: 13-20.

Aikins JS, Kunz JC, Shortliffe EH, Fallat RJ. PUFF: an expert system for interpretation of pulmonary function data. Comput Biomed Res, 1983; 16: 199-208.

Andreassen S, Woldbye M, Falck B, Andersen SK. MUNIN - a causal probabilistic network for interpretation of electromyographic findings. Proc IJCAI, 1987; 366-72.

Barnett GO, Cimino JJ, Hupp JA, Hoffer EP. DXplain: an evolving diagnostic decision support system. Journal American Medical Association, 1987; 258: 67-74.

Barnett JA. Computational methods for a mathematical theory of evidence. Proc IJCAI, 1981; 868-75.

Bleich HL. Computer evaluation of acid-base disorders. J Clin Invest, 1969; 48: 1689-96.

Bleich HL. Computer-based consultation. Electrolyte and acid-base disorders. Amer J Med, 1972; 53: 285-91.

Blum RL. Discovery and representation of causal relationships from a large time-oriented clinical database: the RX project. Comput Biomed Res, 1982; 15: 164-187.

Bobrow DG, Winograd T. An overview of KRL, a knowledge representation language. Cognitive Science, 1977; 1: 3-46.

Boose JH. Uses of repertory grid-centred knowledge acquisition tools for knowledge-based systems. Int J Man-Machine Studies, 1988; 29: 287-310.

Brachman RJ. What IS-A is and isn't: an analysis of taxonomic links in semantic networks. IEEE Computer, 1983; 30-7.

Brachman RJ, Schmolze JG. An overview of the KL-ONE knowledge representation system. Cognitive Science, 1985; 9: 171-216.

Broughton JO, Kennedy TC. Interpretation of arterial blood gases by computer. Chest, 1984; 85: 148-9.

Buchanan BG, Shortliffe EH (eds). Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project. Reading, Ma: Addison-Wesley, 1984.

Chandrasekaran B. On evaluating AI systems for medical diagnosis. AI Magazine, Summer, 1983; 34-48.

Charniak E. The Bayesian basis of common sense medical diagnosis. Proc AAAI, 1983; 70-3.

Cheeseman PC. In defense of probability. Proc IJCAI, 1985; 1002-9.

Cheeseman P. Probabilistic verus fuzzy reasoning. In: LN Kanal & JF Lemmer (eds) Uncertainty in Artificial Intelligence. North-Holland, 1986; 85-102.

Chelsom JJ, Ellis TJ, Cramp DG, Carson ER. A review of blood gas and pH analysis: theory, techniques and equipment. London: City University, Centre for Measurement and Information in Medicine, 1987a. (Research Memorandum No. MIM/JJC-TJE-DGC-ERC/6).

Chelsom JJ, Ellis TJ, Cramp DG, Carson ER. Computer-aided interpretation of blood gas data. London: City University, Centre for Measurement and Information in Medicine, 1987b. (Research Memorandum No. MIM/JJC-TJE-DGC-ERC/7).

Clancey WJ. Tutoring rules for guiding a case method dialogue. Int J Man-Machine Studies, 1979; 11: 25-49.

Clancey WJ. From GUIDON to NEOMYCIN and HERACLES in twenty short lessons: ORN final report 1979-1985. AI Magazine, August, 1986; 40-60.

Clancey WJ, Letsinger R. NEOMYCIN: reconfiguring a rule-based expert system for application to teaching. Proc IJCAI, 1981; 829-36.

Clark JS, Gardner R. On-line computerized acid-base analysis. Proc 21st ACEMB, Houston , Texas, 1968.

Clocksin WF, Mellish CS. Programming in Prolog, 2nd ed. Berlin: Springer-Verlag, 1984.

Cohen ML. A computer program for the interpretation of blood-gas analysis. Comput Biomed Res, 1969; 2: 549-57.

Cohen PR, Howe AE. How evaluation guides AI research. AI Magazine, Winter, 1988; 35-43.

Collinson PO, Jones RG, Howes M, Nicholls J, Sheehy N, Boran GR, Cramp DG. Data capture in the clinical environment - factors limiting acceptability and methods of data validation. In: Medical Informatics 88: Computers in Clinical Medicine. British Medical Informatics Society, London, 1988; 205-9.

Colmerauer A. Prolog in 10 figures. IJCAI, 1987; 487-99.

Coomans D, Broeckaert I, Derde MP, Tassin A, Massart DL, Wold S. Use of a microcomputer for the definition of multivariate confidence regions in medical diagnosis based on clinical laboratory profiles. Comput Biomed Res, 1984; 17:1-14.

Corkill DD, Lesser VR, Hudlicka E. Unifying data-directed and goal-directed control. Proc 2nd Annual National Conference On Artificial Intelligence, 1982; 143-7.

Cravetto C, Lesmo L, Massa Rolandino R, Molino G, Torasso P. An expert system for liver disease diagnosis (LITO2). In: Proc 9th SCAMC, 1985; 330-4.

Croft DJ. Is computerized diagnosis possible? Comput Biomed Res, 1972; 5: 351-67.

Davis R. Interactive transfer of expertise: acquisition of new inference rules. Artif Intell, 1979; 12: 121-158.

Davis R. Meta-rules: reasoning about control. Artif Intell, 1980; 15: 179-222.

Davis R, King J. An overview of production systems. In: E Elcock & D Michie (eds) Machine Intelligence 8: Machine Representations of Knowledge. New York, John Wiley, 1977; 300-34.

De Dombal FT. Computer-aided decision support - the obstacles to progress. Meth Inform Med, 1987; 26: 183-4.

De Dombal FT, Leaper DJ, Staniland JR, McCann AP, Horrocks JC. Computer-aided diagnosis of acute abdominal pain. British Medical Journal, 1972; 9-13.

Dempster A. Upper and lower probabilities induced by multi-valued mapping. Ann Math Statistics, 1967; 38: 325-39.

Diederich J, Ruhmann I, May M. KRITON: a knowledge-acquisition tool for expert systems. Int J Man-Machine Studies, 1987; 26: 29-40.

Duda R, Gaschnig J, Hart P. Model design in the PROSPECTOR consultant system for mineral exploration. In: D. Michie (ed) Expert Systems in the Microelectronic Age. Edinburgh: Edinburgh University Press, 1979; 153-67.

Dybkaer R. Observed value related to reference values. In: R. Grasbeck, T Alstrom (eds). Reference Values in Laboratory Medicine. The Current State of the Art. Chichester: John Wiley, 1981; 263-78.

Easterby-Smith M. The design, analysis and interpretation of repertory grids. Int J Man-Machine Studies, 1980; 13: 3-24.

Elstein AS, Shulman LS, Sprafka SA. Medical Problem Solving: An Analysis of Clinical Reasoning. Cambridge, MA: Harvard University Press, 1978.

Englemore RS, Terry A. Structure and function of the CRYSALIS system. Proc 6th IJCAI, Tokyo, Japan, 1979: 250-6.

Ericsson KA, Simon HA. Verbal reports as data. Psychological Review, 1980; 87: 215-51.

Erman LD, Lesser VR. A multi-level organisation for problem solving using many diverse cooperating sources of knowledge. Proc. 4th IJCAI, Tbilisi, USSR, 1975; 483-90.

Erman LD, Hayes-Roth F, Lesser VR, Reddy DR. The Hearsay-II speech understanding system: integrating knowledge to resolve uncertainty. ACM Computing Surveys, 1980; 12: 213-53.

Erman LD, London PE, Fickas SF. The design and an example use of Hearsay-III. Proc 7th IJCAI, Los Altos, California, 1981; 409-15.

Feinstein AR. Clinical biostatics XXXIX. The haze of Bayes, the aerial palaces of decision analysis, and the computerized Ouiji board. Clinical Pharmacology Therapeutics, 1977; 21: 482-96.

Fillmore C. The case for case. In: Bach & Harms (eds), Universals in Linguistic Theory. Chicago: Holt, 1968.

First MB, Soffer LJ, Miller RA. QUICK (QUick Index to Caduceus knowledge): Using the Internist-1/Caduceus knowledge base as an electronic textbook of medicine. Comput Biomed Res, 1985; 18: 137-65.

Flanagan JC. The critical incident technique. Psychological Bulletin, 1954; 51: 327-58.

Forbus KD. Qualitative process theory. Artif Intell, 1984; 24: 85-168.

Forgy CL. Rete: a fast algorithm for the many pattern / many object pattern match problem. Artif Intell, 1982; 19: 17-37.

Fox J , Glowinski A, O'Neil M. The Oxford System of Medicine: a prototype system for primary care. Proc AIME 87. New York: Springer-Verlag, 1987; 213-6.

Fox MS, Lowenfield S, Kleinosky P. Techniques for sensor-based diagnosis. IJCAI, 1983; 158-63.

Gardner RM, Cannon GH, Morris AH, Olsen KR, Price WG. Computerized blood gas interpretation and reporting system. Computer, 1975; 39-45.

Gheorghe AV, Bali H, Carson E. A Markovian decision model for clinical diagnosis and treatment applied to the respiratory system. IEEE Trans Sys Man Cyber, 1976; SMC-6: 595.

Glymour C. Independence assumptions and Bayesian updating. Artif Intell, 1985; 25: 95-9.

Goldberg M, Green SB, Moss ML, Marbach CB, Garfinkel D. Computer based instruction and diagnosis of acid-base disorders: a systematic approach. JAMA, 1973; 223: 269-75.

Gordon J, Shortliffe EH. A method for managing evidential reasoning in a hierarchical hypothesis space. Artif Intell, 1985; 26: 323-57.

Grasbeck R, Saris N-E. Establishment and use of normal values. Scand J Clin Lab Invest, 1969; 26 (110); 62-3.

Greiner R, Lenat DB. A representation language language. Proc AAAI, 1980; 283-5.

Grover MD. A pragmatic knowledge acquisition methodology. Proc IJCAI, 1983; 436-8.

Gruber TR. Acquiring strategic knowledge from experts. Int J Man-Machine Studies, 1988; 29: 579-97.

Hayes-Roth B, Hayes Roth F, Rosenschein S, Cammarata S. Modelling planning as an incremental opportunistic process. Proc 6th IJCAI, Tokyo, Japan, 1979; 375-83.

Hayes-Roth B. A blackboard architecture for control. Artif Intell, 1985; 26: 251-321.

Hayes-Roth F. Rule-based systems. Comms ACM, 1985; 28: 921-32.

Hayes-Roth F. Knowledge-based expert systems - the state of the art in the US. In: J. Fox (ed) Expert Systems: State of The Art Report. Pergamon Infotech, 1984.

Hendrix GG. Expanding the utility of semantic networks through partitioning. IJCAI, 1975; 115-21.

Hendrix GG. Encoding knowledge in partitioned networks. In: NV Findler (ed). Associative Networks. Representation and Use of Knowledge by Computers. New York: Academic Press, 1979; 51-92.

Hingston DM, Irwin RS, Pratter MR, Dalen JE. A computerized interpretation of arterial pH and blood gas data: do physicians need it? Resp Care, 1982; 27: 809-15.

Horrocks JC, McCann AP, Staniland JR, Leaper DJ, De Dombal FT. Computer-aided diagnosis: description of an adaptable system, and an operational experience with 2,034 cases. British Medical Journal, 1972; 5-9.

IFCC Scientific Committee, Clinical Section, Expert Panel on the Theory of Reference Values. The theory of reference values. Part 6. Presentation of observed values related to reference values. Clin Chem Acta, 1982; 127: 441F-48F.

Jalawayski A, Lauterbach R, Smith BE, Modell JH. A computer method for determination of acid-base and oxygenation variables in adult and infant blood samples. J Lab & Clin Med, 1968; 71: 328.

Johnson RW. Independence and Bayesian updating methods. Artif Intell, 1986; 29: 217-22.

Kaldor G, Rada R. Computerised evaluation of acid-base disorders based on a nine-cell decision matrix. Med & Biol Eng & Comput, 1985; 23: 269-73.

Kari A, Saijonmaa J, Ruoknonen E, Takala J. The assessment of a data management system for critical care. In: Medical Informatics 88: Computers in Clinical Medicine. British Medical Informatics Society, London, 1988.

Kehler TP, Clemenson GD. An application development system for expert systems. Syst Softw, 1984; 3: 212-24.

Kelly GA. The Psychology of Personal Constructs. New York: Norton, 1955.

Kim JH, Pearl J. A computational model for causal and diagnostic reasoning in inference systems. Proc IJCAI, 1983; 190-3.

Kingsland LC III. The evaluation of medical expert systems: experience with the AI/RHEUM knowledge-based consultant system in rheumatology. Proc 9th SCAMC, 1985; 292-5.

Kingsland L, Sharp G, Capps R, Benge J, Kay D, Reese G, Hazelwood S, Lindberg D. Testing of a criteria-based consultant system in rheumatology. Proc MEDINFO, 1983; 514-7.

Kuipers BJ. Qualitative simulation. Artif Intell, 1986; 29: 289-338.

Kuipers B, Kassirer JP. Causal reasoning in medicine: analysis of a protocol. Cognitive Science, 1984; 8: 362-85.

Kulikowski CA. Artificial intelligence in medical consultation systems: a review. IEEE Eng Med Biol, 1988; x: 34-9.

Kunz J. Analysis of physiological behaviour using a causal model based on first principles. Proc AAAI, 1984; 225-9.

Leaper DJ, Horrocks JC, Staniland JR, De Dombal FT. Computer-assisted diagnosis of abdominal pain using "estimates" provided by clinicians. British Medical Journal, 1972; 350-4.

Ledley RS, Lusted LB. Reasoning foundations of medical diagnosis. Science, 1959; 130: 9-21.

Lehnert WG. The Process of Question Answering. A Computer Simulation of Cognition. Hillsdale, NJ: Lawrence Erlbaum Associates, 1978.

Lenat DB. The nature of heuristics. Artif Intell, 1982; 19: 189-249.

Lesmo L, Marzuqli M, Molino G, Torasso P. An expert system for the evaluation of liver functional assessment. J Medical Systems, 1984; 8: 87-101.

Lesser VR, Erman LD. A retrospective view of the Hearsay-II architecture. Proc. 5th IJCAI, 1977; 790-800.

Lindberg DAB, Sharp GC, Kingsland LC III, Weiss SM, Hayes SP, Ueno H, Hazelwood SE. Computer based Rheumatology consultant. In: DAB Lindberg, Kaihara (eds). Proc MEDINFO 80. North Holland, 1980; 1311-5.

Lowerre BT, Reddy DR. The HARPY speech understanding system. In: WA Lea (ed). Trends in Speech Recognition. Englewood Cliffs, NJ: Prentice-Hall, 1980.

Mainland D. Normal values in medicine. Ann NY Acad Science, 1969; 161: 527-37.

Mann J, Ball M. Hyperlipidaemia. Medicine International, 1985; 580-4.

Mars NJI, Miller PL. Knowledge acquisition and verification tools for medical expert systems. Medical Decision Making, 1987; 7: 6-11.

Martin M, Jeffreys B. Use of a minicomputer for storing, reporting and interpreting arterial blood gases/pH and pleural fluid pH. Resp Care, 1983; 28: 301-8.

Masarie FE, Miller RA. INTERNIST-1 to quick medical reference (QMR): the transition from a mainframe to a microcomputer. Proc IEEE 9th Ann Conf Eng Med Biol Society, Boston, 1987; 1521-2.

Masarie FE, Miller RA, Myers JD. INTERNIST-1 properties: representing common sense and good medical practice in a computerised medical knowledge base. Comput Biomed Res, 1985; 18: 458-79.

Michalski RS, Chilausky RL. Knowledge acquisition by encoding expert rules versus computer induction from examples - a case study involving soybean pathology. Int J Man-Machine Studies, 1980; 12: 63-87.

Michie D. High-road and low-road programs. AI Magazine, Winter, 1981; 21-2.

Miller PL. Attending: critiquing a physician's management plan. IEEE Trans PAMI, 1983; PAMI-5: 449-61.

Miller PL. Building an expert critiquing system: ESSENTIAL-ATTENDING. Meth Inform Med, 1986; 25: 71-8.

Miller PL, Rennels GD. Prose generation from expert systems. An applied linguistics approach. AI Magazine, Fall, 1988; 37-44.

Miller RA. INTERNIST-1/CADUCEUS: Problems facing expert consultant programs. Meth Inform Med, 1984; 23: 9-14.

Miller RA, Pople HE, Myers JD. INTERNIST-1: an experimental computer-based diagnostic consultant for general internal medicine. New Eng J Med, 1982; 307: 468-76.

Minsky ML. A framework for representing knowledge. In: PH Winston (ed) The Psychology of Computer Vision. New York: McGraw-Hill, 1975; 211-77.

Miranker DP. TREAT: a better match algorithm for AI production systems. Proc AAAI, 1987; 42-7.

Molino G, Cravetto G, Torasso P, Console L. CHECK: a diagnostic expert system Combining HEuristic and Causal Knowledge. Int J Biomedical Measurement Informatics Control, 1986; 1:182-93.

Musen MA, Fagan LM, Combs DM, Shortliffe EH. Use of a domain model to drive an interactive knowledge-editing tool. Int J Man-Nachine Studies, 1987; 26: 105-21.

Myers JD. The computer as a diagnostic consultant, with emphasis on use of laboratory data. Clin Chem, 1986; 32: 1714-8.

Narins RG, Emmett M. Simple and mixed acid-base disorders: a practical approach. Medicine, 1980; 59: 161-87.

Newell A. Some problems of basic organization in problem-solving programs. In: M.C. Youvits, G.T. Jacobi & G.D. Goldstein (eds). Conference on Self-Organizing Systems. Washington DC: Spartan Books, 1962; 393-423.

Newell A, Simon H. Human Problem Solving. Englewood Cliffs, NJ: Prentice-Hall, 1972.

Nii HP. Blackboard systems: the blackboard model of problem solving and the evolution of blackboard architectures. AI Magazine, Summer 1986a; 38-53.

Nii HP. Blackboard systems. Blackboard application systems, blackboard systems from a knowledge engineering perspective. AI Magazine, August, 1986b; 82-106.

Nii HP, Feigenbaum EA, Anton JJ, Rockmore AJ. Signal-to-symbol transformation: HASP/SIAP case study. AI Magazine, Spring, 1982; 23-35.

Patil RS. Causal representation of patient illness for electrolyte and acid-base diagnosis. Massachusetts Institute of Technology, 1981; MIT/LCS/TR-267.

Patil RS, Senyk O. Efficient structuring of composite causal hypotheses in medical diagnosis. Proc 11th SCAMC, 1987; 23-9.

Patil RS, Szolovits P, Schwartz WB. Causal understanding of patient illness in medical diagnosis. Proc IJCAI, 1981; 893-9.

Patil RS, Szolovits P, Schwartz WB. Information acquisition in diagnosis. Proc AAAI, 1982; 345-8

Patten T. Systemic Text Generation as Problem Solving. Cambridge: Cambridge University Press, 1988.

Pauker SG, Gorry GA, Kassirer JP, Schwartz WB. Towards the simulation of clinical cognition: taking a present illness by computer. Amer J Med, 1976; 60: 981-96.

Paycha F. Medical diagnosis and cybernetics. Proc. Symposium on Mechanization of Thought Processes. London: HM Stationery Office, 1959; Vol 2: 635-59.

Pearl J. On evidential reasoning in a hierarchy of hypotheses. Artif Intell, 1986a; 28: 9-15.

Pearl J. Fusion, propagation, and structuring in belief networks. Artif Intell, 1986b; 29: 241-88.

Pednault EPD, Zucker SW, Muresan LV. On the independence assumption underlying subjective Bayesian updating. Artif Intell, 1981; 16: 213-22.

Pereira FCN, Warren DHD. Definite clause grammars for language analysis - a survey of the formalism and a comparison with augmented transition networks. Artif Intell, 1980; 13: 231-78.

Pople HE. Heuristics methods for imposing structure on ill-structured problems: the structuring of medical diagnostics. In: P. Szolovits (ed) Artificial Intelligence in Medicine. AAAS Symposium Series, Boulder Co: Westview Press, 1982; 119-85.

Pople HE. The formation of composite hypotheses in diagnostic problem solving: an exercise in synthetic reasoning. Proc IJCAI-77; 1030-7.

Pople HE. DIALOG: a model of diagnostic logic for internal medicine. Proc IJCAI-75; 848-55.

Post E. Formal reductions of the general combinatorial problem.
Amer J Math, 1943; 65: 197-268.

Prerau DS. Knowledge acquisition in the development of a large expert system. AI Magazine, Summer, 1987; 43-51.

Price DJ, Mason J. Resolving the numerical chaos at the bedside. In J. Bryant, J. Roberts, P. Windsor (eds), Current Perspectives in Health Computing. London: British Computer Society, 1986; 147-57.

Quaglini S, Stefanelli M, Barosi G, Berzuini A. ANEMIA: an expert consultation system. Comput Biomed Res, 1986; 19: 13-27.

Quaglini S, Stefanelli M, Barosi G, Berzuini A. A performance evaluation of the expert system ANEMIA. Comput Biomed Res, 1988; 21: 307-23.

Quillian MR. Word concepts: a theory and simulation of some basic semantic capabilities. Behavioural Science, 1967; 12: 410-30.

Reddy DR, Erman LD, Neely RB. A model and a system for machine recognition of speech. IEEE Trans Audio & Electroacoustics, 1973; AU-21 (3): 229-38.

Reggia JA. Evaluation of medical expert systems. A case study in performance assessment. Proc 9th SCAMC, 1985; 287-91.

Reggia JA, Nau DS, Wang PY. Diagnostic expert systems based on a set covering model. Int J Man-Machine Studies, 1983; 19: 437-60.

Richards B, Goh AES. Computer assistance in the treatment of patients with acid-base and electrolyte disturbances. Proc MEDINFO, 1977; 407-10.

Robinson JA. A machine-oriented logic based on the resolution principle. J ACM, 1965; 12: 23-41.

Rosner SW, Palmer A, Caceras CA. A computer program for computation and interpretation of pulmonary function data. Comput Biomed Res, 1971; 4: 141-56.

Screck DM, Zacharias D, Grunau CF. Diagnosis of complex acid-base disorders: physician preformance versus the microcomputer. Ann Emerg Med, 1986; 15: 164-70.

Schubert LK. Extending the expressive power of semantic networks. Artif Intell, 1976; 7: 163-98.

Schwartz WB. Medicine and the computer. The promise and problems of change. New Eng J Med, 1970; 283: 1257-64.

Schweickert R, Burton AM, Taylor NK, Corlett EN, Shadbolt NR, Hedgecock AP. Comparing knowledge elicitation techniques: a case study. Artif Intell Review, 1987; 1:245-53.

Scott AC, Clancey WJ, Davis R, Shortliffe EH. Explanation capabilities of knowledge-based production systems. In: BG Buchanan & EH Shortliffe (eds). Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project. Reading, Ma: Addison-Wesley, 1984; 338-62.

Selfridge OG. Pandemonium : a paradigm for learning. Proc. Symposium on Mechanization of Thought Processes. London: HM Stationery Office, 1959; Vol 2: 511-31.

Severinghaus JW. Interpreting acid-base balance. Resp Care, 1982; 27: 1414-5.

Shadbolt N, Burton M. Knowledge elicitation. In: J. Wilson, N. Corlett (eds) Evaluation of Human Work: Practical Ergonomics Methodology. Taylor and France, 1989.

Shafer G. A Mathematical Theory of Evidence. Princeton: Princeton University Press, 1976.

Shafer G, Logan R. Implementing Dempster's rule for hierarchical evidence. Artif Intell, 1987; 33: 271-98.

Shaw MLG, Gaines BR. KITTEN: knowledge initiation and transfer tools for experts and novices. Int J Man-Machine Studies, 1987; 27: 251-80.

Shortliffe EH, Scott AC, Bischoff MB, van Melle W, Jacobs CD. ONCOCIN: an expert system for oncology protocol management. Proc IJCAI, 1981; 876-81.

Shortliffe EH, Davis R, Axline SG, Buchanan BG, Green CC, Cohen SN. Computer-based consultations in clinical therapeutics: explanation and rule acquisition capabilities of the MYCIN system. Comput Biomed Res, 1975; 8: 303-20.

Shortliffe EH, Axline SG, Buchanan BG, Merigan TC, Cohen SN. An artificial intelligence program to advise physicians regarding anitmicrobial therapy. Comput Biomed Res, 1973; 6: 544-60.

Sittig DF. Computerized management of patient care in a complex, controlled clinical trial in the intensive care unit. Proc 11th SCAMC, 1987; 225-32.

Smith J. Design and implementation of a natural language interface which allows interrogation of a medical database. BSc Project, City University, 1988.

Smith RG, Baker JD. The Dipmeter Advisor System. A case study in commercial expert system development. Proc IJCAI, 1983; 122-9.

Stefik M, Bobrow DG, Mittal S, Conway L. Knowledge programming in LOOPS: Report on an experimental course. AI Magazine, Fall 1983; 3-13.

Summers R. Carson ER, Cramp DG, Leaning MS. AIRS - an artificial intelligent respirator system. In: C Cobelli, L Mariani (eds). Proceedings of the IFAC Symposium on Modelling & Control in Biomedical Systems. Oxford: Pergamon Press, 1988; 199-203.

Sunderman FW, Van Soestbergen AA. Probability computations for clinical interpretations of screening tests. Amer J Clin Pathol, 1971; 55: 105-11.

Swartout WR. XPLAIN: a system for creating and explaining expert consulting systems. Artif Intell, 1983; 21: 285-325.

Swartout WR, Smoliar SW. Explaining the link bewteen causal reasoning and expert behaviour. Proc SCAMC, 1987; 37-42.

Szolovits P, Pauker SG. Categorical and probabilistic reasoning in medical diagnosis. Artif Intell, 1978; 11: 115-44.

Tango T. An interpretation of normal ranges based on a new concept 'individual difference quotient' of clinical laboratory data. Medical Informatics, 1981; 6: 161-74.

Teach RL, Shortliffe EH. An analysis of physician attitudes regarding computer-based clinical consultation systems. Comput Biomed Res, 1981; 14: 542-58.

Turing AM. Computing machinery and intelligence. Mind,1950; 59. Also appears in: E. Feigenbaum, J. Feldman (eds), Computers and Thought. New York: McGraw-Hill, 1963; 1-35.

Vallbona C, Pevny E, McMath F. Computer analysis of blood gases and acid-base states. Comput Biomed Res, 1971; 4: 623-33.

van Lente F, Castellani W, Chou D, Matzen RN, Galen RS. Application of the EXPERT consultation system to accelerated laboratory testing and interpretation. Clin Chem, 1986; 32: 1719-25.

van Melle W. A domain-independent production-rule system for consultation programs. Proc IJCAI, 1979; 923-5.

van Melle W, Scott AC, Bennett JS, Peairs M. The Emycin Manual. Stanford: Stanford University, Department of Computer Science, 1981; Report no STAN-CS-81-885.

Warner HR, Olmsted CM, Rutherford BD. HELP - a program for medical decision-making. Comput Biomed Res, 1972; 5: 65-74.

Weiner F, Fayman M, Teitelman U, Bursztein S. Computerized medical reasoning in diagnosis and treatment of acid-base disorders. Crit Care Med, 1983; 11: 470-5.

Weiner F, Weil MH, Carlson RW. Computer systems for facilitating management of the critically ill. Comput Biol Med, 1982; 12: 1-15.

Weizenbaum J. ELIZA - a computer program for the study of natural language communication between man and machine. Comms ACM, 1966; 9: 36-44.

Weiss SM, Kulikowski CA. EXPERT: a system for developing consultation models. Proc IJCAI, 1979; 942-7.

Weiss SM, Kulikowski CA, Amarel S, Safir A. A model-based method for computer-aided medical decision making. Artif Intell, 1978; 11: 145-72.

Weiss SM, Kulikowski CA, Galen RS. Developing microprocessor-based expert models for instrument interpretation. Proc IJCAI, 1981; 853-55.

Weiss SM, Politakis P, Ginsberg A. Empirical analysis and refinement of expert system knowledge bases. Proc 10th SCAMC, 1986; 53-60.

Welbank M. A review of knowledge acquisition techniques for expert systems. Ipswich: Martlesham Consultancy Services, British Telecom Laboratories, 1983.

Woods WA. Transition network grammars for natural language analysis. Comms ACM, 1970; 13: 591-606.

Yu VL, Fagan LM, Wraith SM, Clancey WJ, Scott AC, Hannigan J, Blum RL, Buchanan BG, Cohen SN. Antimicrobial selection by a computer: A blinded evaluation by infectious disease experts. JAMA, 1979; 242: 1279-82.

Zadeh LA. Fuzzy sets. Information and Control, 1965; 8: 338-53.

Zadeh LA. A fuzzy-set-theoretic interpretation of linguistic hedges. J Cybernetics, 1972; 2: 4-34.

Zadeh LA. Outline of a new approach to the analysis of complex systems and decision processes. IEEE Trans Systems Man Cybernetics, 1973; SMC-3; 28-44.

Zadeh LA. Fuzzy sets as a basis for a theory of possibility. Fuzzy Sets and Systems, 1978; 1: 3-28.

Zadeh LA. A theory of approximate reasoning. In: J. Hayes, D. Michie, L.I. Mikulich (eds) Machine Intelligence 9. New York: Halstead Press, 1979; 149-94.

APPENDIX I

BLOOD GAS ANALYSIS AND ACID-BASE BALANCE

A1.1 Introduction

This appendix covers the background physiology and details of clinical practice that are necessary for an full understanding of the material presented in Chapters 4 and 6. Further details can be found in Chelsom, *et al* (1987a;b) or in any of the texts listed in the bibiography.

The next section gives a brief outline of the physiology of cardiopulmonary homeostasis and acid-base balance. This is followed by a description of the main measurements made by automatic blood gas analysers and the information that each conveys. Finally, the major disorders of acid-base balance and hypoxaemic state are presented.

A1.2 Physiology

The continuing function of the human body depends upon the maintenance of a suitable internal environment. There are many regulatory systems at work in the body and the process by which these interact in order maintain that internal environment is called physiological homeostasis.

Cardiopulmonary homeostasis describes the interaction between the cardiovascular and pulmonary systems in order to achieve a balance between ventilation (exchange of oxygen and carbon dioxide in the lungs) and internal respiration (consumption of oxygen and production of carbon dioxide in the tissues). When this balance is achieved, there will be adequate oxygenation of body tissues, elimination of the carbon dioxide produced in tissue respiration and maintenance of pH in intercellular and extracellular fluids. Somewhat confusingly, ventilation is more normally referred to as respiration.

The lungs function as simple gas exchangers: oxygen and carbon dioxide are exchanged between blood and air by a process of passive diffusion, which is brought about by differences in the partial pressures (or tensions) of the gases in blood and alveolar air. The operation of the lungs can be quantified by the respiratory exchange ratio:

$$\text{Respiratory Exchange Ratio} = \frac{\text{Carbon Dioxide Eliminated}}{\text{Oxygen Taken Up}}$$

$$= 0.8 \text{ for a normal, healthy lung}$$

The body derives the energy needed to sustain life by metabolising fats, carbohydrates and proteins (collectively called the substrate). The final stages of metabolism occur in the tissue cells themselves, where chemical reactions consume oxygen and produce carbon dioxide, water and energy. This process is quantified by the respiratory quotient:

$$\text{Respiratory Quotient} = \frac{\text{Carbon Dioxide Produced}}{\text{Oxygen Consumed}}$$

$$= 0.8 \qquad \text{under normal metabolic conditions}$$

It can be seen that under normal conditions there is a balance between the respiratory exchange ratio and the respiratory quotient and that cardiopulmonary homeostasis is achieved.

Most of the oxygen transported in the blood between the lungs and body tissues does so by combination with haemoglobin; a small amount is actually disolved in the blood and accounts for the partial pressure of oxygen in blood (pO_2). Similarly, a small amount of the carbon dioxide carried in the blood dissolves and creates a partial pressure of carbon dioxide in blood (pCO_2). A further small amount combines with haemoglobin, but over 90% reacts with water to form carbonic acid which dissociates to produce hydrogen and bicarbonate ions. The pH of blood is a measure of the level of hydrogen ions (actually the negative logarithm of the hydrogen ion concentration) and must be maintained within a restricted range in order for many vital physiological processes to continue. The pH can be regulated by ventilation (changes in carbon dioxide levels causing changes in the concentrations of hydrogen and bicarbonate ions) or by various metabolic actions. The main metabolic control of pH is exercised by the kidneys which secrete hydrogen ions into the blood and absorb bicarbonate.

A substance that gives up hydrogen ions in solution is an acid; one that combines with hydrogen ions is a base. Carbonic acid gives up a hydrogen ions in a reversible reaction with water ie the bicarbonate produced is a base (called the conjugate base of carbonic acid). A solution containing carbonic acid and bicarbonate ions in equilibrium will tend to resist changes in pH since an increase in hydrogen ions will be offset by combination with bicarbonate and a decrease in hydrogen ions will be compensated by dissociation of carbonic acid. For this reason any such acid/conjugate base pair is called a buffer (ie it buffers changes in hydrogen ion concentration). As well as carbonic acid/bicarbonate, other important buffer systems in the body are proteins (chiefly haemoglobin) and phosphates (operating in the kidneys).

There are three main sources of acid in the body. Dietary acids are produced during the digestion and breakdown of foodstuffs; lactic acid is produced during anaerobic metabolism (*ie* production of energy in cells in the absence of oxygen) which occurs normally in some body cells (eg blood, eye, brain) or during sustained exercise; keto acid is produced during metabolic processes in the liver.

In summary, it can be said that three main interacting mechanisms operate to maintain acid-base balance in the body:

respiration

buffer systems

renal mechanisms

A1.3 Data Measurements in Blood Gas Analysis

The three main measurements made by automatic blood gas analysers are pH, pCO_2 and pO_2, normally using an arterial blood sample.

The pH indicates the degree of acidity (hydrogen ion concentration) of the blood and must be maintained within a fairly narrow range, in order for the electrical function of the heart and nervous system, and various biochemical processes of cellular metabolism, to continue. The cause of fluctuations in pH can be either metabolic or respiratory in origin and other data must be assessed in order to determine this cause. The carbon dioxide tension of arterial blood (pCO_2) is a direct indication of how well the lungs are functioning in their role as gas exchangers; it also indicates the respiratory component of acid-base balance.

The plasma bicarbonate (HCO_3^-) is one indicator of the metabolic component and is calculated from PCO2 using the Henderson-Hasselbalch equation:

$$HCO_3^- = 0.226.pCO_2.\exp(pH-6.1)$$

for HCO_3^- measured in mmol/L and pCO_2 in kPa.

Buffer base (BB) is the total level of all buffer anions in the blood (*ie* bicarbonate, phosphate, proteins, haemoglobin) and gives a true reflection of the buffering power of the blood. Normal buffer base (NBB) is the value of buffer base at normal levels of pH and pCO_2 (7.4 and 5.32 kPa). Base excess is the difference between normal buffer base and the buffer base calculated by automatic blood gas analysers from values of pH, pCO_2 and

haemoglobin concentration. Base excess gives an indication of the metabolic component of acid-base balance.

The third main data measurement made by automatic blood gas analysers is oxygen tension (pO_2). It is used to assess the oxygenation of body tissues; they cannot live without an adequate oxygen supply. A low oxygen tension is called hypoxaemia but its presence does not necessarily imply hypoxia (under oxygenation) of body tissues (and conversely, hypoxia can exist in the presence of an adequate oxygen tension). This is because hypoxia also depends on the concentration of haemoglobin in the blood and its oxygen saturation (*ie* the amount of oxygen it is carrying expressed as a precentage of its maximum oxygen capacity). Oxygen saturation depends not only on oxygen tension but also on pH, pCO_2 and temperature.

A1.4 Acid-Base Disorders

There are four simple acid-base disorders: respiratory acidosis, respiratory alkalosis, metabolic acidosis and metabolic alkalosis. In respiratory disorders a primary change in pCO_2 tension is responsible for the change in pH; in metabolic disorders the change in pH is caused by a primary change in bicarbonate ion concentration.

Under normal conditions, the mechanisms of acid-base regulation described in Section A1.2 act to keep pH at its normal value; hence compensatory responses to the primary influence on pH act to return it to its normal value. Metabolic disturbances are compensated by an increase or decrease in ventilation, respiratory disturbances are compensated by appropriate metabolic action. Depending on the length of time over which compensation has occurred, the simple disorders can be described as uncompensated, partially compensated or fully compensated.

Mixed acid-base disorders exist when two or more of the simple acid-base disturbances are present together. They can be detected because the compensation expected in the case of a simple disorder does not occur. Since carbon dioxide depletion and retention cannot occur simultaneously, it is impossible to have a mixed disorder comprising respiratory acidosis and alkalosis.A respiratory disorder and a metabolic disorder can occur together and two metabolic disorders can be present simultaneously since there are several different causes of simple metabolic acidosis or alkalosis. In theory, a respiratory disorder could also occur with a mixed metabolic disorder to produce a *triple disorder*.

A1.5 Bibliography

Adams AP, Hahn CE. Principles and Practice of Blood-Gas Analysis, (2nd ed). Edinburgh: Churchill Livingston, 1982.

Beetham R. A review of blood pH and blood-gas analysis. Annals Clinical Biochemistry, 1982; 19: 198-213

Davenport HW. The ABC of Acid-Base Chemistry, (6th ed). Chicago: University of Chicago Press, 1974.

Eastham RD. A guide to Water, Electrolyte and Acid-base Metabolism. Bristol: John Wright, 1983.

Gardner MLG. Medical Acid-Base Balance: The Basic Principles. London: Bailliere Tindall, 1978.

Narins RG, Emmett M. Simple and mixed acid-base disorders: A practical approach. Medicine, 1980; 59: 161-87.

Shapiro BA, Harrison RA, Walton JR. Clinical Application of Blood Gases, (3rd ed). Chicago: Year Book Medical Publishers, 1982.

Walmsley RN, Guerin MD. Disorders of Fluid and Electrolyte Balance. Bristol: John Wright, 1984.

APPENDIX II

COMPUTER-AIDED INTERPRETATION OF BLOOD GAS DATA

A2.1 Introduction

Computers have been used in the area of blood gas analysis since the 1960s. Initially, they were employed in the calculation of various derived parameters and to evaluate the nomograms used in interpretation (see for example Jalawayski *et al*, 1968). These calculations are now performed as an integral part of an automatic blood gas analyser.

By the late 1960s the first programs began to appear that performed the interpretation of blood gas data. A program written in assembler for a time-shared mainframe computer (Clark & Gardner, 1968) combined data acquisition and correction with a limited interpretation of results. The first interpretation programs written in higher level languages were those of Cohen (1969) and Bleich (1969). These systems are described in Section A2.2.

Blood gas analysis was one of the first specific areas of medicine in which computers were successfully applied as interpretative aids, which is an indication of their suitability for the task. This is a domain which experts find fairly straightforward to understand but which is notoriously confusing for non-experts - hence a well designed computer program has the potential to dramatically improve the performance of non-expert clinicians.

A survey at the Mercy Medical Centre, Denver showed a drop in *untimely or inappropriate therapeutic responses* from 33% to 9% when a computerized interpretation system assisted clinicians in the treatment of patients with life-threatening acid-base disorders (Broughton & Kennedy, 1984). A separate study revealed that a cross-section of clinicians from emergency medicine, internal medicine, paediatrics, surgery and family practice had success rates of 86%, 49% and 17% for the diagnosis of single, double and triple acid-base disorders (Schreck *et al*, 1986).

Another study found that 71% of clinicians participating in a trial of diagnostic accuracy did not regard computer assistance necessary, though the overall success rate for the group in diagnosing acid-base disturbances was only 39% (Hingston *et al*, 1982). The circumstances of this trial have been called into question in an argument over the different procedures that clinicians use to make a diagnosis (Severinghaus, 1982). The basic objection to the trial was that a computer is likely to perform better than a clinician when an over-complicated approach to diagnosis is proscribed. This

argument serves to highlight another feature of blood gas interpretation that can be confusing for the non-expert clinician, namely the difference in nomenclature and diagnostic procedure between different centres (particularly between America and Europe). The use of computer aids to interpretation at least ensures that a common approach is taken amongst its users.

A2.2 Early Systems

Two systems, developed in the late 1960s, provided the basic models for many subsequent programs that interpreted blood gas data. At St Joseph's Hospital in Phoenix, Arizona at program was developed in FORTRAN running on an IBM mainframe computer (Cohen, 1969). Data were input using punched cards, the adequacy of arterial oxygenation was checked using O2 saturation and pH, pCO_2 and HCO_3^- were checked with the Henderson-Hasselbalch equation. The main interpretation was made by identifying the relevant area on a nomogram of pH, pCO_2 and HCO_3^-. There were 28 different areas defined on the nomogram and an interpretation was provided for each one which depended on whether the patient was mechanically ventilated or not (ie there were 56 possible interpretations).

The second of the early systems was developed at the Beth Israel Hospital, Boston and was written in a high level language called Stringcomp for a PDP-1D mainframe computer (Bleich, 1969). It was later translated into BASIC to run on a GE-635. The main part of the program was based on a comprehensive algorithm which determined the acid-base disorder and a differential diagnosis of the underlying disease. The user input the available data which were checked for physiological consistency. The program then stepped through the algorithm, requesting further data as necessary. Finally, an evaluation note was printed which indicated the acid-base disorder, a differential diagnosis of diseases, suggestions for therapy and references to the medical literature. The program was later extended to cover electrolyte disorders (Bleich, 1972) using the same algorithmic approach. A number of interesting observations were made by Bleich about the clinical use of his program:

> formulation of a computer program forced the researcher to explicitly state all the medical knowledge involved
>
> the program encouraged clinicians to experiment with hypothetical patients and data and thus served as an educational tool
>
> clinicians tended to become more disciplined in their evaluation of laboratory data

A2.3 Algorithm-Based Systems

A system for storing, reporting and interpreting blood gas data (and pleural fluid pH) was introduced at the Mt Sinai Medical Centre, Cleveland in 1979 (Martin & Jeffreys, 1983). An automatic blood gas analyser and a co-oximeter were connected to a DEC minicomputer through an RS232 interface. Data from the analysers passed on-line to the computer where they were stored on floppy disk; reports based on these data could be printed as required. A branched algorithm provided interpretation based on the numerical data only (no clinical information was used) by selecting one of 16 acid-base statements, 5 oxygenation statements and 5 ventilaation statements. The management and interpretation programs were written in BASIC. In clinical use it was found that the staff, who had little previous experience with computers, encountered only minor difficulties in using the system. Handwritten reports, with their inevitable transcription errors, were eliminated and the volume of paperwork was greatly reduced.

A microcomputer program written in BASIC has been evaluated at the Veterans Administration Medical Centre in Michigan (Kaldor & Rada, 1985). The program used a branched algorithm to classify pH, pCO_2 and HCO_3^- into five intervals and to check compensation with published equations (Narins & Emmett, 1980). The program was evaluated in tests on 50 patients. Each patient was diagnosed using clinical data only and this was compared with the system's diagnosis based on blood gas data. The diagnoses were deemed positive (acid-base disorder diagnosed), negative (normal acid-base status) or equivocal (no firm decision possible) and were compared using a nine-element matrix.

A2.4 Nomogram-Based Systems

An acid-base map of PCO2, pH and HCO3, constructed using the medical literature and the expertise of clinicians at the Hospital of the University of Pennsylvania formed the basis of a program written in FORTRAN for a PDP-6 computer (Goldberg *et al*, 1973). The acid-base map was divided into more than 30 areas, each with an associated differential diagnosis. When enough information had been input to focus on one area of the map, an algorithm specific to that area was used to narrow down the differential diagnosis. A report could be generated in various degrees of detail in order to accommodate different users, ranging from students to experienced clinicians.

The same acid-base map was used in a modified form for a system at the Latter Day Saints Hospital in Salt Lake City (Gardner *et al*, 1975). The logic for determining the region of the acid-base map corresponding to the input data was implemented using the

HELP system, a general program for making decisions based on input data and previous decisions (Warner *et al*, 1971). The system was appraised in clinical trials lasting 17 days by asking clinicians to complete an evaluation note after each computer interpretation. The conclusions drawn from this survey were:

the interpretation was accurate and was accepted by clinical staff

patient care was improved and reporting time was reduced, especially in the intensive care unit

procedures and nomenclature were standardized

clinical staff gained educational benefit from the computer interpretation

A2.5 Intelligent Systems

The knowledge-based system ABEL (Patil, 1982) also used the acid-base nomogram for its initial diagnosis of disorders. It was then capable of deep level causal reasoning as has been described in Section 2.3.2. Another system described by its creators as a *cognitive model* approach to the interpretation of acid-base disorders was developed at the Rambam University Hospital in Israel (Weiner *et al*, 1983). The system made use of a series of domain independent programs, written in FORTRAN, that had previously been used to contruct other medical systems (Weiner *et al*, 1982).

A knowledge-base was created as a tree-like pathway of inferences, using information from local experts and the medical literature. Each inference in the pathway was grouped with criteria for its confirmation or rejection to form a *medical logic module*. Each of the criteria could be present or absent (Y or N) and carried a points score for the confirmation, implication or rejection of the module. Each logic module could appear as one of the criteria in another module.

The program operated by accepting data input from the user and working through the logic modules, evaluating their criteria. Numerical data were analysed by numerical modules which classified them into discrete categories so that they could be treated in the same way as the other (Y or N) criteria. The inference represented by a logic module became instantiated when the total points contributed by its criteria exceeded a pre-set threshold value. At the end of a consultation text associated with each confirmed inference was printed out.

In a trial of 54 cases, the system agreed in 50 with the clinicians involved in the specification of its knowledge base (93% agreement). The developers claimed that the system *can capture the medical reasoning of a group of experts, can express the nuances of their approach and make such knowledge available to the less experienced practitioner* (Weiner *et al*, 1983). Such claims seem slightly ambitious: the decision evaluation process employed was very simple, the user interaction was unfriendly (all data had to be input at the start of a session) and the explanation consisted of printing the stored text for confirmed inferences. Despite these criticisms, the system dispalyed a high degree of accuracy and its modular design and tree structured knowledge base resemble the system that forms the subject of this thesis.

A2.6 Summary

A number of systems have been described which perform the interpretation of blood gas data. Generally speaking, these systems adopt an algorithmic approach or an approach based on the identification of regions on the acid-base nomogram. In either case, they exhibit purely categorical reasoning and output results in the form of textual descriptions stored with each possible interpetation. Five such systems were compared in a study at the University of Manchester (the systems of Cohen, Bleich and Goldberg described above and two developed by Vallbona, *et al* (1971) and Rosner, *et al* (1971)). The results of this study (Richards & Goh, 1977) showed that there was total agreement amongst the systems in just over 20% of cases, that 3 or more agreed on 88% of cases and that there was total disagreement in 2% of cases.

APPENDIX III

KNOWLEDGE BASE FOR BLOOD GAS INTERPRETATION

DATA VARIABLES

pH
upper limit 9.000
lower limit 6.000
mean value 7.390
std dev 0.025
default none

PCO2 kPa
upper limit 50.000
lower limit 0.000
mean value 5.320
std dev 0.333
default none

HCO3 mmol/L
upper limit 50.000
lower limit 0.000
mean value 24.000
std dev 1.000
default none

Base Excess mmol/L
upper limit 30.000
lower limit -30.000
mean value 0.000
std dev 1.150
default none

PO2 kPa
upper limit 100.000
lower limit 0.000
mean value 12.640
std dev none
default none

FIO2 %
upper limit 100.000
lower limit 0.000
mean value none
std dev none
default 21.000

Hb g/100ml g/100ml
upper limit 20.000
lower limit 0.000
mean value none
std dev none
default 16.000

Na mmol/L
upper limit 200.000
lower limit 0.000
mean value 140.000
std dev 2.500
default none

K mmol/L
upper limit 20.000
lower limit 0.000
mean value 4.500
std dev 0.500
default none

Cl mmol/L
upper limit 150.000
lower limit 0.000
mean value 100.000
std dev 2.500
default none

Ca
upper limit none
lower limit none
mean value none
std dev none
default none

O2 sat %
upper limit 100.000
lower limit 0.000
mean value none
std dev none
default 97.500

Anion Gap mmol/L
upper limit 50.000
lower limit 0.000
mean value 12.000
std dev 2.000
default none

P50
upper limit none
lower limit none
mean value none
std dev none
default none

Temperature C
upper limit 50.000
lower limit 20.000
mean value 37.200
std dev 0.600
default none

Respiratory rate /min
upper limit 100.000
lower limit 0.000
mean value 18.000
std dev 3.000
default none

MAP mmHg
upper limit 200.000
lower limit 0.000
mean value 90.000
std dev 10.000
default none

Creatinine umol/L
upper limit 200.000
lower limit 0.000
mean value none
std dev none
default none

Glucose
upper limit none
lower limit none
mean value none
std dev none
default none

Buffer Base mmol/L
upper limit 100.000
lower limit 10.000
mean value 42.000
std dev 5.000
default none

age years
upper limit 120
lower limit 0
mean value none
std dev none
default none

RELATIONSHIPS FOR DATA DERIVATION
1. Anion Gap=Na+K-Cl-HCO3
2. Base Excess=(1-Hb/43)*((HCO3-24.25)+(2.3*Hb+7.7)*(pH-7.4))
3. HCO3=0.23*PCO2*10^(pH-6.1)
4. Buffer Base=Na+K-Cl

SIGNS & SYMPTOMS

coma
absent
present
unknown

tetany
absent
present
unknown

tremor
absent
present
unknown

anxiety
absent
present
unknown

somnolence/headaches
absent
present
unknown

Kussmaul breathing
absent
present
unknown

shock
absent
present
unknown

vasodilatation
normal
vasodilated
vasoconstricted
unknown

acid ingestion
absent
present
unknown

diuretics
absent
present
unknown

GI fluid loss
absent
present
unknown

PATIENT HISTORY

sex

female
male
unknown

occupation
unknown

disorder
unknown

previous disorder
unknown

clinical diagnosis
unknown

FRAMES FOR DISORDERS
CLASS: hypoxaemic state

Frame for hypoxaemic state

Frame for O2 therapy Type of hypoxaemic state (0.500)
relation 1. FIO2>21

Frame for no O2 therapy Type of hypoxaemic state (0.500)
relation 1. FIO2=21

Frame for uncorrected hypox Type of O2 therapy (0.250)
relation 1. PO2<10.64

Frame for corrected hypoxaemia Type of O2 therapy (0.250)
relation 1. PO2>=10.64 relation 2. PO2<13.3

Frame for excessive O2 therapy Type of O2 therapy (0.250)
relation 1. PO2>=13.3

Frame for adequate O2 tension Type of no O2 therapy (0.250)
relation 1. PO2>=10.64

Frame for mild hypoxaemia Type of no O2 therapy (0.250)
relation 1. PO2>=7.98 relation 2. PO2<10.64

Frame for moderate hypoxaemia Type of no O2 therapy (0.250)
relation 1. PO2>=5.32 relation 2. PO2<7.98

Frame for severe hypoxaemia Type of no O2 therapy (0.250)
relation 1. PO2<5.32

CLASS: acid-base disorder

Frame for acid-base disorder

Frame for measurement error Type of acid-base disorder (0.100)
variable Anion Gap usual 0.300
variable Anion Gap high 0.300
variable Anion Gap low 0.400
variable Cl usual 0.400
variable Cl high 0.300
variable Cl low 0.300
variable K usual 0.400
variable K high 0.300
variable K low 0.300
variable Na usual 0.400
variable Na high 0.300
variable Na low 0.300
variable Base Excess low 0.300
variable Base Excess high 0.300
variable Base Excess usual 0.400
variable HCO3 low 0.300
variable HCO3 high 0.300
variable HCO3 usual 0.400
variable PCO2 low 0.300
variable PCO2 high 0.300
variable PCO2 usual 0.400
variable pH low 0.300
variable pH high 0.300
variable pH usual 0.400

Frame for neutral ph Type of acid-base disorder (0.300) variable pH usual 1.000

Frame for dominant acidosis Type of acid-base disorder (0.300) variable pH low 1.000

Frame for dominant alkalosis Type of acid-base disorder (0.300) variable pH high 1.000

Frame for resp acid & met acid Type of dominant acidosis (0.330)
variable Anion Gap usual 0.500
variable Anion Gap high 0.500
variable Base Excess low 1.000
variable HCO3 low 1.000
variable PCO2 high 1.000

Frame for respiratory acidosis Type of dominant acidosis (0.330)
variable Anion Gap usual 1.000
variable PCO2 high 1.000

Frame for metabolic acidosis Type of dominant acidosis (0.330)
variable Anion Gap usual 0.500
variable Anion Gap high 0.500
variable Base Excess low 1.000
variable HCO3 low 1.000

Frame for resp alk & met alk Type of dominant alkalosis (0.330)
variable Anion Gap high 1.000
variable Base Excess high 1.000
variable HCO3 high 1.000
variable PCO2 low 1.000

Frame for resp alkalosis Type of dominant alkalosis (0.330)
variable Anion Gap usual 0.500
variable Anion Gap high 0.500
variable PCO2 low 1.000

Frame for metabolic alkalosis Type of dominant alkalosis (0.330)
variable Anion Gap usual 0.500
variable Anion Gap high 0.500
variable Base Excess high 1.000
variable HCO3 high 1.000

Frame for uncomp resp acid Type of respiratory acidosis (0.500)
variable Base Excess usual 1.000
variable HCO3 usual 1.000

Frame for part comp resp acid Type of respiratory acidosis (0.500)
variable Base Excess high 1.000
variable HCO3 high 1.000

Frame for uncomp met acid Type of metabolic acidosis (0.500)
variable PCO2 usual 1.000

Frame for part comp met acid Type of metabolic acidosis (0.500)
variable PCO2 low 1.000

Frame for uncomp resp alk Type of resp alkalosis (0.500)
variable Base Excess usual 1.000
variable HCO3 usual 1.000

Frame for uncomp met alk Type of metabolic alkalosis (0.500)
variable PCO2 usual 1.000

Frame for part comp met alk Type of metabolic alkalosis (0.500)
variable PCO2 high 1.000

Frame for part comp resp alk Type of resp alkalosis (0.500)
variable Base Excess low 1.000
variable HCO3 low 1.000

Frame for comp resp alk Type of low HCO3 with pCO2 (0.330)
variable Anion Gap usual 1.000
variable Base Excess low 1.000
symptom GI fluid loss absent 0.800
symptom GI fluid loss present 0.200
symptom diuretics absent 0.800
symptom diuretics present 0.200
symptom acid ingestion absent 0.800
symptom acid ingestion present 0.200
symptom vasodilatation normal 0.500
symptom vasodilatation vasodilated 0.100
symptom vasodilatation vasoconstricted 0.400
symptom shock absent 0.800
symptom shock present 0.200
symptom Kusmaul breathing absent 0.800
symptom Kusmaul breathing present 0.200
symptom anxiety absent 0.200
symptom anxiety present 0.800
symptom tremor absent 0.800
symptom tremor present 0.200
symptom tetany absent 0.200
symptom tetany present 0.800
symptom coma absent 0.800
symptom coma present 0.200
symptom somnolence/headaches absent 0.800
symptom somnolence/headaches present 0.200
relation 1. ΔHCO3>=3.75*ΔPCO2-4 relation 2. ΔHCO3=<3.75*ΔPCO2+4

Frame for resp alk & met acid Type of low HCO3 with pCO2 (0.330)
variable Anion Gap usual 0.500
variable Anion Gap high 0.500
variable Base Excess low 1.000
symptom GI fluid loss absent 0.200
symptom GI fluid loss present 0.800
symptom diuretics absent 0.800
symptom diuretics present 0.200
symptom acid ingestion absent 0.200
symptom acid ingestion present 0.800
symptom vasodilatation normal 0.500
symptom vasodilatation vasodilated 0.100
symptom vasodilatation vasoconstricted 0.400
symptom shock absent 0.200
symptom shock present 0.800
symptom Kusmaul breathing absent 0.200
symptom Kusmaul breathing present 0.800
symptom somnolence/headaches absent 0.800
symptom somnolence/headaches present 0.200
symptom anxiety absent 0.200
symptom anxiety present 0.800
symptom tremor absent 0.200
symptom tremor present 0.800
symptom tetany absent 0.200
symptom tetany present 0.800
symptom coma absent 0.200
symptom coma present 0.800

Frame for comp met acid Type of low HCO3 with pCO2 (0.330)
variable Anion Gap usual 0.500
variable Anion Gap high 0.500
variable Base Excess low 1.000
symptom GI fluid loss absent 0.200
symptom GI fluid loss present 0.800
symptom diuretics absent 0.800
symptom diuretics present 0.200
symptom acid ingestion absent 0.200
symptom acid ingestion present 0.800
symptom vasodilatation normal 0.500
symptom vasodilatation vasodilated 0.100
symptom vasodilatation vasoconstricted 0.400
symptom somnolence/headaches absent 0.800
symptom somnolence/headaches present 0.200
symptom anxiety absent 0.800
symptom anxiety present 0.200
symptom tetany absent 0.800
symptom tetany present 0.200
symptom coma absent 0.200
symptom coma present 0.800
symptom Kusmaul breathing present 0.800
symptom Kusmaul breathing absent 0.200
symptom shock present 0.800
symptom shock absent 0.200
symptom tremor present 0.800
symptom tremor absent 0.200
relation 1. ΔPCO2=<0.2*ΔHCO3 relation 2. ΔPCO2>=0.133*ΔHCO3

Frame for comp met alk Type of high HCO3 with pCO2 (0.010)
variable K low 0.900
variable Base Excess high 1.000
variable K usual 0.100
symptom acid ingestion absent 0.800
symptom acid ingestion present 0.200
symptom vasodilatation normal 0.500
symptom vasodilatation vasodilated 0.100
symptom vasodilatation vasoconstricted 0.400
symptom shock absent 0.800
symptom shock present 0.200
symptom Kusmaul breathing absent 0.800
symptom Kusmaul breathing present 0.200
symptom somnolence/headaches absent 0.800
symptom somnolence/headaches present 0.200
symptom anxiety absent 0.800
symptom anxiety present 0.200
symptom tremor absent 0.800
symptom tremor present 0.200
symptom tetany absent 0.200
symptom tetany present 0.800
symptom coma absent 0.800
symptom coma present 0.200
symptom GI fluid loss absent 0.800
symptom GI fluid loss present 0.200
symptom diuretics absent 0.200
symptom diuretics present 0.800
relation 1. ΔPCO2=<0.133*ΔHCO3 relation 2. ΔPCO2>=0.033*ΔHCO3

Frame for resp acid & met alk Type of high HCO3 with pCO2 (0.190)
variable K low 0.900
variable Base Excess high 1.000
variable K usual 0.100
symptom acid ingestion absent 0.800
symptom acid ingestion present 0.200
symptom vasodilatation normal 0.500
symptom vasodilatation vasodilated 0.400
symptom vasodilatation vasoconstricted 0.100
symptom shock absent 0.800
symptom shock present 0.200
symptom Kusmaul breathing absent 0.800
symptom Kusmaul breathing present 0.200
symptom somnolence/headaches absent 0.200
symptom somnolence/headaches present 0.800
symptom anxiety absent 0.800
symptom anxiety present 0.200
symptom tremor absent 0.200
symptom tremor present 0.800
symptom tetany absent 0.800
symptom tetany present 0.200
symptom coma absent 0.200
symptom coma present 0.800
symptom diuretics present 0.800
symptom diuretics absent 0.200
symptom GI fluid loss present 0.200
symptom GI fluid loss absent 0.800

Frame for comp resp acid Type of high HCO3 with pCO2 (0.800)
variable K usual 1.000
variable Base Excess high 1.000
symptom GI fluid loss absent 0.800
symptom GI fluid loss present 0.200
symptom acid ingestion absent 0.800
symptom acid ingestion present 0.200
symptom vasodilatation normal 0.500
symptom vasodilatation vasodilated 0.400
symptom vasodilatation vasoconstricted 0.100
symptom shock absent 0.800
symptom shock present 0.200
symptom Kusmaul breathing absent 0.800
symptom Kusmaul breathing present 0.200
symptom somnolence/headaches absent 0.200
symptom somnolence/headaches present 0.800
symptom anxiety absent 0.800
symptom anxiety present 0.200
symptom tremor absent 0.200
symptom tremor present 0.800
symptom tetany absent 0.800
symptom tetany present 0.200
symptom coma absent 0.200
symptom coma present 0.800
symptom diuretics absent 0.800
symptom diuretics present 0.200
relation 1. ΔHCO3>=3*ΔPCO2-4 relation 2. ΔHCO3=<3*ΔPCO2+4

Frame for mixed met acid Type of dominant acidosis (0.000)
variable Base Excess low 1.000
variable HCO3 low 1.000
variable PCO2 low 1.000
variable Anion Gap usual 0.500
variable Anion Gap high 0.500
variable Cl high 1.000

Frame for met acid & met alk Type of neutral ph (0.100)
variable Anion Gap usual 0.500
variable Anion Gap high 0.500
variable Buffer Base high 1.000
variable Base Excess low 0.300
variable Base Excess high 0.300
variable Base Excess usual 0.400
variable HCO3 low 0.300
variable HCO3 high 0.300
variable HCO3 usual 0.400
variable PCO2 low 0.300
variable PCO2 high 0.300
variable PCO2 usual 0.400

Frame for normal blood gases Type of neutral ph (0.100)
variable Base Excess usual 1.000
variable Buffer Base usual 1.000
variable Anion Gap usual 1.000
variable HCO3 usual 1.000
variable PCO2 usual 1.000

Frame for high HCO3 with pCO2 Type of neutral ph (0.400)
variable Anion Gap usual 1.000
variable HCO3 high 1.000
variable PCO2 high 1.000

Frame for low HCO3 with pCO2 Type of neutral ph (0.400)
variable HCO3 low 1.000
variable PCO2 low 1.000

FRAMES FOR DISEASES
CLASS: Causes of resp acid

Frame for Causes of resp acid
history resp acid & met acid present 0.200
history resp acid & met alk present 0.200
history respiratory acidosis present 0.300
history comp resp acid present 0.300

Frame for Ventilator Breakdown Type of Causes of resp acid (0.000)

Frame for Lung Disease Type of Causes of resp acid (0.000)

Frame for Impaired Lung Motion Type of Causes of resp acid (0.000)

Frame for Thoracic Cage Limit Type of Causes of resp acid (0.000)

Frame for Neuromusc Disorders Type of Causes of resp acid (0.000)

Frame for CNS Depression Type of Causes of resp acid (0.000)

Frame for cardiac arrest Type of Causes of resp acid (0.000)

Frame for Sedatives Type of CNS Depression (0.000)

Frame for Resp centre lesion Type of CNS Depression (0.000)

Frame for trauma Type of Resp centre lesion (0.000)

Frame for isthaemia Type of Resp centre lesion (0.000)

Frame for Myopathies Type of Neuromusc Disorders (0.000)

Frame for Neuropathies Type of Neuromusc Disorders (0.000)

Frame for muscular dystrophy Type of Myopathies (0.000)

Frame for potassium depletion Type of Myopathies (0.000)

Frame for Guillain-Barre Type of Neuropathies (0.000)

Frame for Polio Type of Neuropathies (0.000)

Frame for Kyphoscoliosis Type of Thoracic Cage Limit (0.000)

Frame for Scleroderma Type of Thoracic Cage Limit (0.000)

Frame for Crush injury Type of Thoracic Cage Limit (0.000)

Frame for Pleural effusion Type of Impaired Lung Motion (0.000)

Frame for Pneumothorax Type of Impaired Lung Motion (0.000)

Frame for Acute Obstruction Type of Lung Disease (0.000)

Frame for Chronic Obst Disease Type of Lung Disease (0.000)

Frame for Severe Pneumonia Type of Lung Disease (0.000)

Frame for Pulmonary Oedema Type of Lung Disease (0.000)

Frame for Aspiration Type of Acute Obstruction (0.000)

Frame for Tumor Type of Acute Obstruction (0.000)

Frame for Spasm Type of Acute Obstruction (0.000)

Frame for Laryngospasm Type of Spasm (0.000)

Frame for Bronchospasm Type of Spasm (0.000)

CLASS: Causes of met acid

Frame for Causes of met acid
history metabolic acidosis present 0.200
history resp acid & met acid present 0.200
history resp alk & met acid present 0.200
history comp met acid present 0.200
history mixed met acid present 0.200

Frame for Normal/Hyperkalaemic Type of Causes of met acid (0.000)

Frame for Hypokalaemic Type of Causes of met acid (0.000)

Frame for Lactic Acidosis Type of Causes of met acid (0.000)

Frame for Ketoacidosis Type of Causes of met acid (0.000)

Frame for Toxins Type of Causes of met acid (0.000)

Frame for Renal Failure Type of Causes of met acid (0.000)

Frame for Paraldehyde Type of Toxins (0.000)

Frame for Salicyclates Type of Toxins (0.000)

Frame for Ethylene Glycol Type of Toxins (0.000)

Frame for Methanol Type of Toxins (0.000)

Frame for Glycogenesis Defect Type of Ketoacidosis (0.000)

Frame for Glycogenosis I Type of Ketoacidosis (0.000)

Frame for Diabetes Mellitus Type of Ketoacidosis (0.000)

Frame for Starvation Type of Ketoacidosis (0.000)

Frame for Alcoholism Type of Ketoacidosis (0.000)

Frame for Ureteral Diversions Type of Hypokalaemic (0.000)

Frame for Carb Anhyd Inhibitor Type of Hypokalaemic (0.000)

Frame for Post-Hypercapnic Type of Hypokalaemic (0.000)

Frame for Renal Tubular Acid Type of Hypokalaemic (0.000)

Frame for Diarrhoea Type of Hypokalaemic (0.000)

Frame for Sulphur Toxicity Type of Normal/Hyperkalaemic (0.000)

Frame for Hydronephrosis Type of Normal/Hyperkalaemic (0.000)

Frame for Early Renal Failure Type of Normal/Hyperkalaemic (0.000)

CLASS: Causes of resp alk

Frame for Causes of resp alk
history resp alkalosis present 0.300
history resp alk & met alk present 0.200
history resp alk & met acid present 0.200
history comp resp alk present 0.300

Frame for CNS Disorders Type of Causes of resp alk (0.000)

Frame for Fever Type of Causes of resp alk (0.000)

Frame for Endotoxaemia Type of Causes of resp alk (0.000)

Frame for Hyperthyroidism Type of Causes of resp alk (0.000)

Frame for Pregnancy Type of Causes of resp alk (0.000)

Frame for Hormones/Drugs Type of Causes of resp alk (0.000)

Frame for Liver Insufficiency Type of Causes of resp alk (0.000)

Frame for Mild Pul Oedema Type of Causes of resp alk (0.000)

Frame for Pulmonary Disease Type of Causes of resp alk (0.000)

Frame for Post Metabolic Acid Type of Causes of resp alk (0.000)

Frame for Pneumonia Type of Pulmonary Disease (0.000)

Frame for Pulmonary Embolus Type of Pulmonary Disease (0.000)

Frame for Restrictive Disorder Type of Pulmonary Disease (0.000)

Frame for Analeptic Overdose Type of Hormones/Drugs (0.000)

Frame for Progesterone Type of Hormones/Drugs (0.000)

Frame for Catecholomines Type of Hormones/Drugs (0.000)

Frame for Salicylates Type of Hormones/Drugs (0.000)

Frame for CNS Infection Type of CNS Disorders (0.000)

Frame for CNS Tumor Type of CNS Disorders (0.000)

Frame for Cerebrovasc Accident Type of CNS Disorders (0.000)

CLASS: Causes of met alk

Frame for Causes of met alk
history metabolic alkalosis present 0.300
history resp alk & met alk present 0.200
history resp acid & met alk present 0.200
history comp met alk present 0.300

Frame for Bartler Syndrome Type of Causes of met alk (0.000)

Frame for Exogenous Steroid Type of Causes of met alk (0.000)

Frame for Adrenal Disorder Type of Causes of met alk (0.000)

Frame for Carbenicillin Type of Causes of met alk (0.000)

Frame for Penicillin Type of Causes of met alk (0.000)

Frame for Post Fasting Glucose Type of Causes of met alk (0.000)

Frame for Hyperparathyroidism Type of Causes of met alk (0.000)

Frame for Cushing Syndrome Type of Adrenal Disorder (0.000)

Frame for Hyperaldosteronism Type of Adrenal Disorder (0.000)

Frame for Carbenoxalone Type of Exogenous Steroid (0.000)

Frame for Licorice Ingestion Type of Exogenous Steroid (0.000)

APPENDIX IV

EVALUATION OF THE BLOOD GAS SYSTEM: CASE DATA

CASE 1

CLINICAL DIAGNOSIS Diabetic

pH 7.05

pCO2 ..... 1.596 kPa

HCO3-..... 5 mmol/L

BE ..... -30 mmol/L

pO2 .....14.36 kPa

CLINICAL DATA:
age 17
sex female
Kussmaul breathing

CASE 2

CLINICAL DIAGNOSIS Chronic obstuctive airway disease

pH 7.10

pCO2 ..... 3.325 kPa

HCO3-..... 8 mmol/L

BE ..... -20 mmol/L

pO2 ..... 5.32 kPa

O2 sat ..... 52 %

CLINICAL DATA:
age 66
sex female

CASE 3

CLINICAL DIAGNOSIS Fractured arm

pH 7.42

pCO2 ..... 3.857 kPa

HCO3-..... 19 mmol/L

BE ..... -4 mmol/L

pO2 ..... 10.24 kPa

O2 sat ..... 96 %

CLINICAL DATA:
age 21 RR 16 /min
sex female
pregnant

CASE 4

CLINICAL DIAGNOSIS Fractured ankle

pH 7.55

pCO2 ..... 3.591 kPa

HCO3-..... 23 mmol/L

BE ..... 0 mmol/L

pO2 ..... 13.96 kPa

O2 sat ..... 100 %

CLINICAL DATA:
age 24
sex female
disorientated
confused

CASE 5

CLINICAL DIAGNOSIS Drug Overdose

pH 7.15

pCO2 ..... 10.640 kPa

HCO3-..... 28 mmol/L

BE ..... 0 mmol/L

pO2 ..... 5.58 kPa

O2 sat ..... 80 %

CLINICAL DATA:
age 34
sex female
comatose

CASE 6

CLINICAL DIAGNOSIS Salicylate intoxication

pH 7.15

pCO2 ..... 1.596 kPa

HCO3-..... 4.4 mmol/L

CLINICAL DATA:
age 3
sex male

CASE 7

CLINICAL DIAGNOSIS Respiratory failure

pH 7.257

pCO2 ..... 6.30 kPa

HCO3-..... 21.7 mmol/L

BE .....-5.2 mmol/L

pO2 ..... 7.3 kPa

FIO2 ..... 50 %

CLINICAL DATA:
Lymphoma sex male
E. Coli infection age 72
Chemotherapy + pancytopaemia

CASE 8

FOLLOWS CASE (7) AFTER 7 hours

CLINICAL DIAGNOSIS Pneumonia

pH 7.388

pCO2 ..... 4.06 kPa

HCO3-..... 18.6 mmol/L

BE .....-4.7 mmol/L

pO2 ..... 13.8 kPa

FIO2 ..... 80 %

CLINICAL DATA:
Lymphoma RR 14 /min
E. Coli infection Temp 38.2 C
Chemotherapy + pancytopaemia MAP 50 mmHg
Dopamine sex male
Dobutamine age 72

CASE 9

CLINICAL DIAGNOSIS Pneumonia

FOLLOWS CASE (8) AFTER 4 hours

pH 7.426

pCO2 ..... 4.53 kPa

HCO3-..... 22.6 mmol/L

BE .....-0.6 mmol/L

pO2 ..... 8.7 kPa

FIO2 ..... 60 %

CLINICAL DATA:
Lymphoma RR 14 /min
E. Coli infection Temp 36.6 C
Chemotherapy + pancytopaemia MAP 60 mmHg
Dopamine sex male
Dobutamine age 72

CASE 10

CLINICAL DIAGNOSIS Pneumonia

FOLLOWS CASE (9) AFTER 12 hours

pH 7.421

pCO2 ..... 4.85 kPa

CO3-..... 24.0 mmol/L

pO2 ..... 20.2 kPa

FIO2 ..... 80 %

CLINICAL DATA:
Lymphoma RR 14 /min
E. Coli infection Temp 36.0 C
Chemotherapy + pancytopaemia MAP 85 mmHg
Dopamine sex male
Dobutamine age 72

CASE 11

CLINICAL DIAGNOSIS Pneumonia

FOLLOWS CASE (10) AFTER 1 day

pH 7.445

pCO2 ..... 5.11 kPa

HCO3-..... 26.7 mmol/L

BE ..... 3.2 mmol/L

pO2 ..... 15.7 kPa

FIO2 ..... 80 %

CLINICAL DATA:
Lymphoma RR 14 /min
E. Coli infection Temp 36.0 C
Chemotherapy + pancytopaemia MAP 100 mmHg
Dopamine sex male
Dobutamine age 72

CASE 12

CLINICAL DIAGNOSIS Lymphoma

FOLLOWS CASE (11) AFTER 19 days

pH 7.340

pCO2 ..... 4.75 kPa

HCO3-..... 19.7 mmol/L

BE ..... -4.7 mmol/L

pO2 ..... 15.5 kPa

FIO2 ..... 35 %

CLINICAL DATA:
Lymphoma RR 15 /min
E. Coli infection Temp 35.8 C
Chemotherapy + pancytopaemia MAP 80 mmHg
Dopamine sex male
Dobutamine age 72

CASE 13

CLINICAL DIAGNOSIS Lymphoma

FOLLOWS CASE (12) AFTER 1 day

pH 7.240

pCO2 ..... 6.02 kPa

HCO3-..... 19.7 mmol/L

BE ..... -6.8 mmol/L

pO2 ..... 12.6 kPa

FIO2 ..... 35 %

CLINICAL DATA:
sex male
Lymphoma
E. Coli infection
Extubated
age 72
RR 24 /min
Temp 36.6 C
MAP 110 mmHg

CASE 14

CLINICAL DIAGNOSIS Septic shock, DIC

pH 7.218

pCO2 ..... 3.50 kPa

HCO3-..... 10.8 mmol/L

BE ..... -14.7 mmol/L

pO2 ..... 8.5 kPa

FIO2 ..... 60 %

CLINICAL DATA:
sex female
age 65
respiratory distress
collapse
cold & clammy
Temp 37 C
MAP 60 mmHg
septic shock

CASE 15

CLINICAL DIAGNOSIS Post op

FOLLOWS CASE (14) AFTER 5 hours

pH 7.513

pCO2 ..... 4.04 kPa

HCO3-..... 24.7 mmol/L

BE ..... 2.9 mmol/L

pO2 ..... 6.5 kPa

FIO2 ..... 40 %

CLINICAL DATA:
sex female
age 65
resuscitated post op
RR 14 /min
Temp 37 C
MAP 60 mmHg

CASE 16

CLINICAL DIAGNOSIS Post op

FOLLOWS CASE (15) AFTER 2 hours

pH 7.457

pCO2 ..... 4.70 kPa

HCO3-..... 22.5 mmol/L

pO2 ..... 8.4 kPa

FIO2 ..... 60 %

CLINICAL DATA:
Resuscitated with fluids & dopamine
vasodilatation
RR 14 /min
MAP 60 mmHg
Temp 37 C

CASE 17

CLINICAL DIAGNOSIS Post op

FOLLOWS CASE (16) AFTER 1 day

pH 7.5

pCO2 ..... 3.28 kPa

HCO3-..... 19.8 mmol/L

pO2 ..... 16.9 kPa

FIO2 ..... 30 %

CLINICAL DATA:
Receiving 50% dextrose infusion
Temp 36.8 C
RR 18 /min
MAP 100 mmHg

CASE 18

CLINICAL DIAGNOSIS Post op

FOLLOWS CASE (17) AFTER 1 day

pH 7.540

pCO2 ..... 5.15 kPa

HCO3-..... 34.1 mmol/L

BE ..... 11.5 mmol/L

pO2 ..... 8.1 kPa

FIO2 ..... 30 %

CLINICAL DATA:
Receiving dextrose & fat infusion
Temp 37.3 C
RR 14 /min
MAP 70 mmHg

CASE 19

CLINICAL DIAGNOSIS Left ventricular failure

pH 7.321

pCO2 ..... 3.98 kPa

HCO3-..... 15.6 mmol/L

BE ..... -8.4 mmol/L

pO2 ..... 18.1 kPa

FIO2 ..... 28 %

CLINICAL DATA:
sex female
age 73
Hypovolaemia
Diuretics
RR 20 /min
MAP 95 mmHg
Temp 37.0 C
vasoconstriction

CASE 20

CLINICAL DIAGNOSIS Congestive cardiac failure & pulmonary embolism

FOLLOWS CASE (19) AFTER 3 days

pH 7.343

pCO2 ..... 6.06 kPa

HCO3-..... 25.0 mmol/L

BE ..... -0.5 mmol/L

pO2 ..... 5.6 kPa

CLINICAL DATA:
sex female
age 73
Cyanosis
RR 20 /min
MAP 110 mmHg
Temp 35.0 C

CASE 21

CLINICAL DIAGNOSIS Respiratory failure

FOLLOWS CASE (20) AFTER 2 days

pH 7.410

pCO2 ..... 7.02 kPa

HCO3-..... 34.2 mmol/L

BE ..... 8.7 mmol/L

pO2 ..... 9.2 kPa

FIO2 ..... 60 %

CLINICAL DATA:
sex female
age 73
atrial fibrilation
IPPV
RR 12 /min
MAP 90 mmHg
Temp 36.7 C

CASE 22

CLINICAL DIAGNOSIS Respiratory failure

FOLLOWS CASE (21) AFTER 6 hours

pH 7.530

pCO2 ..... 5.06 kPa

HCO3-..... 32.7 mmol/L

BE ..... 10.1 mmol/L

pO2 ..... 8.6 kPa

FIO2 ..... 60 %

CLINICAL DATA:
sex female
age 73
IPPV
RR 13 /min
MAP 50 mmHg
Temp 36.7 C

CASE 23

CLINICAL DIAGNOSIS Respiratory failure

pH 7.240

pCO2 ..... 2.92 kPa

HCO3-..... 12.6 mmol/L

BE ..... -15.7 mmol/L

pO2 ..... 6.94 kPa

FIO2 ..... 60 %

CLINICAL DATA:
sex male
age 22
sulphuric acid ingestion
RR 56 /min
MAP 100 mmHg
Temp 37.3 C

CASE 24

CLINICAL DIAGNOSIS Sulpuric acid ingestion

FOLLOWS CASE (23) AFTER 2 hours

pH 7.289

pCO2 ..... 6.38 kPa

HCO3-..... 21.3 mmol/L

BE ..... -3.8 mmol/L

pO2 ..... 14.90 kPa

FIO2 ..... 60 %

CLINICAL DATA:
sex male
age 22
300 ml 8.4% HCO3 over one hour
RR 16 /min
MAP 130 mmHg
Temp 37.3 C

CASE 25

CLINICAL DIAGNOSIS Sulpuric acid ingestion

FOLLOWS CASE (24) AFTER 1 hour

pH 7.410

pCO2 ..... 3.80 kPa

HCO3-..... 21.1 mmol/L

pO2 ..... 20.50 kPa

FIO2 ..... 60 %

CLINICAL DATA:
sex male
age 22
Temp 37.3
RR 26 /min
MAP 80 mmHg

CASE 26

CLINICAL DIAGNOSIS Sulpuric acid ingestion

FOLLOWS CASE (25) AFTER 7 hours

pH 7.210

pCO2 ..... 5.31 kPa

HCO3-..... 15.7 mmol/L

BE ..... -11.0 mmol/L

pO2 ..... 7.9 kPa

FIO2 ..... 60 %

CLINICAL DATA:
sex male
age 22
Temp 37.8
RR 20 /min
MAP 50 mmHg

CASE 27

CLINICAL DIAGNOSIS Sulpuric acid ingestion

FOLLOWS CASE (26) AFTER 1 hour

pH 7.173

pCO2 ..... 6.85 kPa

HCO3-..... 16.2 mmol/L

BE ..... -10.1 mmol/L

pO2 ..... 7.63 kPa

FIO2 ..... 60 %

CLINICAL DATA:
sex male
age 22
Temp 37.8
RR 22 /min
MAP 60 mmHg

CASE 28

CLINICAL DIAGNOSIS Sulpuric acid ingestion

FOLLOWS CASE (27) AFTER 1 day

pH 7.380

pCO2 ..... 2.61 kPa

HCO3-..... 16.5 mmol/L

BE ..... -10.0 mmol/L

pO2 ..... 12.0 kPa

FIO2 ..... 40 %

CLINICAL DATA:
sex male
age 22
Temp 37.9
RR 18 /min
MAP 80 mmHg

CASE 29

CLINICAL DIAGNOSIS Unknown

pH 7.550

pCO2 ..... 7.70 kPa

HCO3-..... 49.0 mmol/L

BE ..... 20.0 mmol/L

pO2 ..... 12.0 kPa

FIO2 ..... 21 %

CLINICAL DATA:
sex male
age 56

CASE 30

CLINICAL DIAGNOSIS Immediately post op

pH 7.400

pCO2 ..... 4.80 kPa

HCO3-..... 23.9 mmol/L

BE ..... -0.9 mmol/L

pO2 ..... 29.50 kPa

FIO2 ..... 40 %

CLINICAL DATA:
sex male
age 70
Temp 34.1
RR 14 /min
MAP 80 mmHg
IPPV

CASE 31

CLINICAL DIAGNOSIS Post op

FOLLOWS CASE (30) AFTER 2 days

pH 7.370

pCO2 ..... 3.50 kPa

HCO3-..... 18.2 mmol/L

BE ..... -7.8 mmol/L

pO2 ..... 9.88 kPa

FIO2 ..... 36 %

CLINICAL DATA:
sex male
age 70
Temp 38.2
Atrial fibrillation
vasoconsriction
RR 14 /min
MAP 50 mmHg
IPPV
septic shock

CASE 32

CLINICAL DIAGNOSIS Post op

FOLLOWS CASE (31) AFTER 12 hours

pH 7.416

pCO2 ..... 4.66 kPa

HCO3-..... 24.0 mmol/L

BE ..... -0.7 mmol/L

pO2 ..... 23.96 kPa

FIO2 ..... 60 %

CLINICAL DATA:
sex male
age 63
Temp 38.2
RR 16 /min
MAP 95 mmHg
IPPV

CASE 33

CLINICAL DIAGNOSIS Post op

FOLLOWS CASE (32) AFTER 3 days

pH 7.302

pCO2 ..... 12.08 kPa

HCO3-..... 24.5 mmol/L

BE ..... 0.0 mmol/L

pO2 ..... 12.08 kPa

FIO2 ..... 40 %

CLINICAL DATA:
sex male
age 63
Temp 37.0
atrial fibrilation
RR 30 /min
MAP 120 mmHg
IPPV
extubated

CASE 34

CLINICAL DIAGNOSIS Ischaemic farct

pH 7.48

pCO2 ..... 2.79 kPa

HCO3-..... 20.2 mmol/L

BE ..... -4.3 mmol/L

pO2 ..... 4.23 kPa

CLINICAL DATA:
sex male
age 68
sepsis from gangrenous toe
slight dehydration
RR 20 /min
Temp 36.8 C
MAP 100 mmHg

CASE 35

CLINICAL DIAGNOSIS Ischaemic farct

FOLLOWS CASE (34) AFTER 1 hour

pH 7.500
pCO2 ..... 2.05 kPa
HCO3-..... 17.8 mmol/L
BE ..... -4.8 mmol/L
pO2 ..... 3.38 kPa
Na ..... 128 mmol/L
K ..... 5.6 mmol/L
WBC 21.2
Urea 8.5

CLINICAL DATA:
sex male
age 68
sepsis from gangrenous toe
cyanosis
RR 20 /min
Temp 36.8 C
MAP 100 mmHg
tachycardia

CASE 36

CLINICAL DIAGNOSIS Ischaemic farct

FOLLOWS CASE (35) AFTER 1 hour

pH 7.46

pCO2 ..... 2.44 kPa

HCO3-..... 18.9 mmol/L

BE ..... -6.8 mmol/L

pO2 ..... 7.03 kPa

FIO2 ..... 100 %

CLINICAL DATA:
sex male
age 68
sepsis from gangrenous toe
RR 20 /min
Temp 38.0 C
MAP 100 mmHg

CASE 37

CLINICAL DIAGNOSIS Ischaemic farct / respiratory failure

FOLLOWS CASE (36) AFTER 3 hours

pH 7.500

pCO2 ..... 4.76 kPa

HCO3-..... 29.8 mmol/L

BE ..... 6.3 mmol/L

pO2 ..... 6.00 kPa

FIO2 ..... 60 %

CLINICAL DATA:
sex male
age 68
sepsis from gangrenous toe

RR 20 /min
Temp 38.5 C
MAP 100 mmHg

CASE 38

CLINICAL DIAGNOSIS MI ARDS

POLLOWS CASE (37) AFTER 3 days

pH 7.413

pCO2 ..... 4.85 kPa

HCO3-..... 23.5 mmol/L

BE ..... -0.1 mmol/L

pO2 ..... 4.80 kPa

FIO2 ..... 60 %

CLINICAL DATA:
sex male
age 68
sepsis from gangrenous toe

RR 20 /min
Temp 38.5 C
MAP 90 mmHg

CASE 39

CLINICAL DIAGNOSIS MI ARDS

POLLOWS CASE (38) AFTER 1 day

pH 7.01

pCO2 ..... 9.99 kPa

HCO3-..... 19.3 mmol/L

BE ..... -12.6 mmol/L

pO2 ..... 4.60 kPa

FIO2 ..... 20 %

CLINICAL DATA:
sex male
age 68
cardiac arrest

RR 0 /min
MAP 0 mmHg

CASE 40

CLINICAL DIAGNOSIS Collapse / Cardiac arrest

pH 7.46

pCO2 ..... 3.36 kPa

HCO3-..... 22.2 mmol/L

BE ..... -2.7 mmol/L

pO2 ..... 10.50 kPa

FIO2 ..... 35 %

CLINICAL DATA:
sex male
age 76
collapse due to internal bleeding

RR 26 /min
MAP 50 mmHg
Temp 35.0 C

CASE 41

CLINICAL DIAGNOSIS Mesenteric embolism, post op

pH 7.259

pCO2 ..... 4.46 kPa

HCO3-..... 15.2 mmol/L

BE ..... -10.2 mmol/L

pO2 ..... 11.70 kPa

FIO2 ..... 60 %

CLINICAL DATA:
sex male
age 76
Laparotomy - no mesenteric embolus

RR 16 /min
MAP 65 mmHg
Temp 37.0 C

CASE 42

CLINICAL DIAGNOSIS Collapse MI (?)

pH 7.100

pCO2 ..... 8.06 kPa

HCO3-..... 15.4 mmol/L

BE ..... -11.0 mmol/L

pO2 ..... 45.40 kPa

FIO2 ..... 100 %

CLINICAL DATA:
sex male
age 72
intubated IPPV

RR 10 /min
MAP 90 mmHg
Temp 37.0 C

CASE 43

CLINICAL DIAGNOSIS Anoxic cerebral damage

POLLOWS CASE (42) AFTER 8 hours

pH 7.430

pCO2 ..... 3.99 kPa

HCO3-..... 22.2 mmol/L

BE ..... -2.1 mmol/L

pO2 ..... 10.11 kPa

CLINICAL DATA:
sex male
age 72
fitting

RR 30 /min
MAP 140 mmHg
Temp 36.5 C

CASE 44

CLINICAL DIAGNOSIS Collapse

pH 7.23

pCO2 ..... 6.50 kPa

HCO3-..... 17.0 mmol/L

BE ..... -1.7 mmol/L

pO2 ..... 4.40 kPa

FIO2 ..... 20 %

CLINICAL DATA:
sex male
age 61
cardiac arrest

RR 0 /min
MAP 0 mmHg
Temp 36.0 C

CASE 45

CLINICAL DIAGNOSIS Anoxic cerebral damage

FOLLOWS CASE (44) AFTER 18 hours

pH 7.618

pCO2 ..... 2.90 kPa

HCO3-..... 28.5 mmol/L

BE ..... 4.5 mmol/L

pO2 ..... 18.35 kPa

FIO2 ..... 40 %

CLINICAL DATA:
sex male
age 61

RR 14 /min
MAP 80 mmHg
Temp 37.0 C

CASE 46

CLINICAL DIAGNOSIS MI, CVA post APSAC

pH 7.26

pCO2 ..... 6.58 kPa

HCO3-..... 24.0 mmol/L

pO2 ..... 11.30 kPa

FIO2 ..... 20 %

CLINICAL DATA:
sex male
age 33
CT scan: large infarct

RR 28 /min
MAP 80 mmHg
Temp 35.6 C

CASE 47

CLINICAL DIAGNOSIS CVA

FOLLOWS CASE (46) AFTER 8 hours

pH 7.51

pCO2 ..... 2.95 kPa

HCO3-..... 23.1 mmol/L

BE ..... 1.6 mmol/L

pO2 ..... 9.22 kPa

FIO2 ..... 35 %

CLINICAL DATA:
sex male
age 33

RR 15 /min
MAP 100 mmHg
Temp 36.2 C

CASE 48

CLINICAL DIAGNOSIS CVA

FOLLOWS CASE (47) AFTER 3 days

pH 7.20

pCO2 ..... 8.44 kPa

HCO3-..... 20.7 mmol/L

BE ..... -4.6 mmol/L

pO2 ..... 15.15 kPa

FIO2 ..... 100 %

CLINICAL DATA:
sex male
age 33
extubated
brain dead

RR 15 /min
MAP 80 mmHg
Temp 35.8 C

CASE 49

CLINICAL DIAGNOSIS Pancreatitis

pH 7.30

pCO2 ..... 5.10 kPa

HCO3-..... 19.6 mmol/L

BE ..... -6.1 mmol/L

pO2 ..... 14.20 kPa

FIO2 ..... 50 %

CLINICAL DATA:
sex female
age 74
vasoconstriction

CASE 50

CLINICAL DIAGNOSIS Unknown

pH 7.55

pCO2 ..... 6.90 kPa

HCO3-..... 44.01/L

BE ..... 17.0 mmol/L

pO2 ..... 9.70 kPa

CLINICAL DATA:
sex male
age 71

CASE 51

CLINICAL DIAGNOSIS Post op aortic aneurism repair

pH 7.451

pCO2 ..... 5.28 kPa

HCO3-..... 28.0 mmol/L

BE ..... 4.2 mmol/L

pO2 ..... 7.68 kPa

FIO2 ..... 60 %

CLINICAL DATA:

sex male
age 68

RR 14 /min
MAP 80 mmHg
Temp 39.5 C

CASE 52

CLINICAL DIAGNOSIS Post op aortic aneurism repair

FOLLOWS CASE (51) AFTER 9 days

pH 7.502

pCO2 ..... 4.79 kPa

HCO3-..... 29.9 mmol/L

BE ..... 6.0 mmol/L

pO2 ..... 10.70 kPa

FIO2 ..... 40 %

CLINICAL DATA:
sex male
age 68

RR 34 /min
MAP 95 mmHg
Temp 37.0 C

CASE 53

CLINICAL DIAGNOSIS Post op

pH 7.54

pCO2 ..... 3.29 kPa

HCO3-..... 21.4 mmol/L

BE ..... 0.8 mmol/L

pO2 ..... 5.90 kPa

FIO2 ..... 40 %

CLINICAL DATA:
sex male
age 82

RR 16 /min
MAP 75 mmHg
Temp 37.5 C

CASE 54

CLINICAL DIAGNOSIS Post op

FOLLOWS CASE (53) AFTER 13 hours

pH 7.49

pCO2 ..... 3.83 kPa

HCO3-..... 23.7 mmol/L

BE ..... 0.7 mmol/L

pO2 ..... 12.10 kPa

FIO2 ..... 40 %

CLINICAL DATA:
sex male
age 82

RR 16 /min
MAP 85 mmHg
Temp 37.0 C

CASE 55

CLINICAL DIAGNOSIS Post op

FOLLOWS CASE (54) AFTER 2 weeks

pH 7.41

pCO2 ..... 2.59 kPa

HCO3-..... 12.5 mmol/L

BE ..... -9.2 mmol/L

pO2 ..... 14.60 kPa

FIO2 ..... 28 %

CLINICAL DATA:
sex male
age 82

RR 15 /min
MAP 100 mmHg
Temp 37.0 C

CASE 56

CLINICAL DIAGNOSIS LVF, Diabetic

pH 7.200

pCO2 ..... 7.35 kPa

HCO3-..... 21.9 mmol/L

BE ..... -5.9 mmol/L

pO2 ..... 7.50 kPa

FIO2 ..... 20 %

CLINICAL DATA:
sex male
age 53
Acute SOB
COAD

CASE 57

CLINICAL DIAGNOSIS LVF, Diabetic

FOLLOWS CASE (56) AFTER 1 hour

pH 7.261

pCO2 ..... 6.07 kPa

HCO3-..... 20.7 mmol/L

BE ..... -5.8 mmol/L

pO2 ..... 8.90 kPa

FIO2 ..... 28 %

CLINICAL DATA:
sex male
age 53

CASE 58

CLINICAL DIAGNOSIS LVF, Pneumonia, Diabetic

FOLLOWS CASE (57) AFTER 4 hours

pH 7.356

pCO2 ..... 4.88 kPa

HCO3-..... 20.7 mmol/L

BE ..... -3.6 mmol/L

pO2 ..... 8.60 kPa

FIO2 ..... 28 %

CLINICAL DATA:
sex male
age 53

CASE 59

CLINICAL DIAGNOSIS LVF, Diabetic, Cardiac arrest

FOLLOWS CASE (58) AFTER 3 days

pH 7.016

pCO2 ..... 9.34 kPa

HCO3-..... 18.2 mmol/L

BE ..... -13.4 mmol/L

pO2 ..... 67.40 kPa

FIO2 ..... 100 %

CLINICAL DATA:
sex male
age 53

CASE 60

CLINICAL DIAGNOSIS LVF

FOLLOWS CASE (59) AFTER 1 hour

pH 7.258

pCO2 ..... 7.46 kPa

HCO3-..... 25.3 mmol/L

BE ..... -2.2 mmol/L

pO2 ..... 8.70 kPa

FIO2 ..... 24 %

CLINICAL DATA:
sex male
age 53
Chest X-ray: pulmonary oedema

CASE	SYSTEM	EXPERT	SENIOR	JUNIOR
1	part comp met acid 0.9782	part comp met acid	part comp met acid	part comp met acid
2	part comp met acid 0.9782	part comp met acid	resp alk & met acid	part comp met acid
3	comp resp alk 0.4822 resp alk & met acid 0.3707 part comp resp alk 0.1247	comp resp alk	part comp resp alk	comp resp alk
4	uncomp resp alk 0.9624	uncomp resp alk	uncomp resp alk	uncomp resp alk
5	respiratory acidosis 0.6691	uncomp resp acid	uncomp resp acid	part comp resp acid
6	part comp met acid 0.9782	resp alk & met acid	resp alk & met acid	part comp met acid
7	resp acid & met acid 0.8884	resp acid & met acid	resp acid & met acid	uncomp resp acid
8	comp resp alk 0.5124 resp alk & met acid 0.3939	comp met acid	part comp met acid	comp resp alk
9	uncomp resp alk 0.3158 normal blood gases 0.2602 measurement error 0.1817 met acid & met alk 0.1046	normal blood gases	comp met acid	comp resp alk
10	normal blood gases 0.7045 uncomp resp alk 0.1290 measurement error 0.1024	normal blood gases	normal blood gases	normal blood gases
11	uncomp met alk 0.7284 resp alk & met alk 0.1367	normal blood gases	normal blood gases	comp met alk
12	metabolic acidosis 0.5629 comp resp alk 0.2245 resp alk & met acid 0.1741	part comp met acid	part comp met acid	part comp met acid
13	resp acid & met acid 0.6932 uncomp met acid 0.2945	resp acid & met acid	resp acid & met acid	resp acid & met alk
14	part comp met acid 0.9782	resp acid & met acid	part comp met acid	uncomp met acid
15	uncomp resp alk 0.6254 resp alk & met alk 0.3126	uncomp resp alk	part comp met alk	uncomp resp alk
16	uncomp resp alk 0.6905 normal blood gases 0.1724 measurement error 0.1082	uncomp resp alk	normal blood gases	comp resp alk
17	resp alkalosis 0.6142 measurement error 0.3769	part comp resp alk	uncomp resp alk	part comp resp alk
18	uncomp met alk 0.8497 resp alk & met alk 0.1251	uncomp met alk	uncomp met alk	part comp met alk

Table A4.1 Summary of Diagnoses for Acid-Base Balance

CASE	SYSTEM	EXPERT	SENIOR	JUNIOR
19	low HCO3 with pCO2 0.5448 part comp met acid 0.4335	uncomp met acid	part comp met acid	part comp met acid
20	uncomp resp acid 0.5615 normal blood gases 0.2889	uncomp resp acid	uncomp resp acid	part comp resp acid
21	resp acid & met alk 0.7259 part comp met alk 0.2056	comp resp acid	comp resp acid	resp acid & met alk
22	uncomp met alk 0.7823 resp alk & met alk 0.1966	uncomp met alk	uncomp met alk	uncomp met alk
23	part comp met acid 0.9782	part comp met acid	part comp met acid	part comp met acid
24	resp acid & met acid 0.9086	resp acid & met acid	uncomp resp acid	part comp met acid
25	resp alkalosis 0.4731 measurement error 0.3068 met acid & met alk 0.2092	comp met acid	comp resp alk	uncomp resp alk
26	uncomp met acid 0.9153	part comp met acid	uncomp met acid	uncomp met acid
27	resp acid & met acid 0.9895	resp acid & met acid	resp acid & met acid	uncomp met acid
28	part comp met acid 0.4842 measurement error 0.2587 met acid & met alk 0.1847	comp met acid	part comp met acid	resp alk & met acid
29	part comp met alk 0.9782	part comp met alk	part comp met alk	resp alk & met alk
30	normal blood gases 0.7734 measurement error 0.1080	normal blood gases	normal blood gases	normal blood gases
31	comp resp alk 0.4522 resp alk & met acid 0.3476 part comp met acid 0.1386	comp met acid	part comp met acid	part comp met acid
32	normal blood gases 0.6083 uncomp resp alk 0.1979 measurement error 0.1148	normal blood gases	normal blood gases	uncomp resp alk
33	uncomp resp acid 0.9624	uncomp resp acid	uncomp resp acid	uncomp resp acid
34	part comp resp alk 0.9358	part comp resp alk	part comp resp alk	resp alk & met acid
35	part comp resp alk 0.9748	resp alk & met acid	part comp resp alk	uncomp resp alk
36	part comp resp alk 0.8053 resp alk & met acid 0.1732	comp resp alk	part comp resp alk	resp alk & met acid
37	resp alk & met alk 0.5377 uncomp met alk 0.4478	uncomp met alk	uncomp met alk	resp alk & met alk
38	normal blood gases 0.7503 measurement error 0.1018	normal blood gases	normal blood gases	uncomp resp alk

Table A4.1 Summary of Diagnoses for Acid-Base Balance (continued)

CASE	SYSTEM	EXPERT	SENIOR	JUNIOR
39	resp acid & met acid 0.9918	resp acid & met acid	resp acid & met acid	resp acid & met acid
40	resp alkalosis 0.8370 resp alk & met acid 0.1007	resp alk & met acid	uncomp resp alk	uncomp resp alk
41	metabolic acidosis 0.9821	uncomp met acid	uncomp met acid	part comp met acid
42	resp acid & met acid 0.9735	resp acid & met acid	resp acid & met acid	resp acid & met acid
43	resp alkalosis 0.3976 comp resp alk 0.2793 resp alk & met acid 0.2147	comp resp alk	comp resp alk	part comp resp alk
44	resp acid & met acid 0.9353	uncomp met acid	resp acid & met acid	uncomp resp acid
45	resp alk & met alk 0.9914	part comp resp alk	uncomp resp alk	resp alk & met alk
46	uncomp resp acid 0.4703 resp acid & met acid 0.3390	uncomp resp acid	uncomp resp acid	uncomp resp acid
47	uncomp resp alk 0.9582	uncomp resp alk	part comp resp alk	uncomp resp alk
48	resp acid & met acid 0.9892	part comp resp acid	resp acid & met acid	resp acid & met acid
49	metabolic acidosis 0.9597	uncomp met acid	uncomp met alk	part comp met acid
50	part comp met alk 0.9809	part comp met alk	uncomp met alk	part comp met acid
51	uncomp met alk 0.8612	uncomp met alk	part comp met acid	part comp met acid
52	resp alk & met alk 0.5015 uncomp met alk 0.4836	uncomp met alk	part comp resp alk	uncomp met alk
53	uncomp resp alk 0.9055	uncomp resp alk	uncomp resp alk	uncomp resp alk
54	uncomp resp alk 0.964	part comp resp alk	part comp resp acid	uncomp resp alk
55	comp resp alk 0.4522 resp alk & met acid 0.3476 part comp resp alk 0.1386	comp resp alk	comp resp alk	comp met acid
56	resp acid & met acid 0.9824	resp acid & met acid	resp acid & met acid	resp acid & met acid
57	resp acid & met acid 0.7398 uncomp met acid 0.2470	resp acid & met acid	resp acid & met acid	resp acid & met acid
58	low HCO3 with pCO2 0.5788 metabolic acidosis 0.3411	uncomp met acid	comp met acid	comp met acid
59	resp acid & met acid 0.9735	resp acid & met acid	resp acid & met acid	resp acid & met acid
60	uncomp resp acid 0.9427	uncomp resp acid	uncomp resp acid	resp acid & met acid

Table A4.1 Summary of Diagnoses for Acid-Base Balance (continued)

APPENDIX V

LIPID PHYSIOLOGY AND HYPERLIPIDAEMIA

A5.1 Physiology

Lipids are compounds containing carbon, hydrogen and oxygen which are an essential component of the body structure and form an important source of energy for tissue cells. They are obtained directly from dietary intake or by synthesis in the body from two carbon units (acetate) derived from carbohydrates and amino acids. As well as being metabolized immediately, lipids form a large part of the body's energy store in the form of adipose tissue. They are an ideal form in which to store energy since they have a higher energy yield than either proteins or carbohydrates.

There are three types of lipids: simple lipids (including cholesterol and triglycerides), compound lipids and derived lipids. Since they themselves are insoluble in water, lipids must be converted into a soluable complex, by binding to apoproteins, before they can be transported to the body tissues in the blood plasma. The complexes that carry lipids in the plasma are called lipoproteins, of which there are several types, depending on the types of apoproteins and the proportion of cholesterol and triglyceride. The different types are: chylomicrons, very low density lipoproteins (VLDL), low density lipoproteins (LDL), intermediate density lipoproteins (IDL) and high density lipoproteins (HDL).

Most of the lipid intake in the diet is in the form of triglycerides which are carried from the intestine to the tissues by chylomicrons. Triglycerides are also synthesized by the body in tthe liver and transported from there to the tissues by VLDL. In the tissues, triglycerides are removed from the chylomicrons and VLDL to leave liproprotein remnants which are then metabolized to form LDL. The LDL, which contains much of the cholesterol, is taken up by tissue cells through LDL receptors.

Some cholesterol is derived directly from the diet (about 25%) but most is synthesized by the tissues themselves, primarily in the liver. Cholesterol is carried from the liver to tissue cells by LDL and from the tissues back to the liver by HDL. The level of lipids in the plasma depends on the rate of formation of chylomicrons in the intestine and VLDL and LDL in the liver and the rate of metabolism of triglycerides and removal of LDL in the tissues.

A5.2 Measurements and Observations

The normal laboratory test for plasma lipids measures cholesterol and triglyceride levels; in some cases, HDL is also measured. Provided that the triglceride level is less than 5 mmol/L, LDL can be calculated from values of cholesterol, HDL and triglceride using:

$$LDL = cholesterol - HDL - triglyceride/2.2$$

where all measurements are in mmol/L.

An indication of the levels of VLDL and chylomicrons can be obtained by observing plasma samples stored in a test tube. Excess VLDL is indicated by a cloudy appearance of the sample and excess chylomicrons appear as a creamy layer on the top.

Different lipoproteins move with different velocities under the influence of an electric field due to differences in the charges carried by the molecules. This phenomenon is the basis of lipid electrophoresis in which a plasma sample is subjected to an electric field and the resulting pattern, formed by the dispersion of the different lipoproteins, is analysed. A classification of hyperlipidaemia produced by the World Health Organization is based on electrophoresis patterns (Types I, IIa, IIb, III, IV, V).

Xanthomas are tumours caused by deposits of lipids. There are several types of xanthoma: tendon xanthomas can appear on tendons of the hands, ankle and knee; tuberous xanthomas appear as a swelling of the affected area; eruptive xanthomas are small, flat nodules on the skin; linear palmar xanthomas are linear lesions occurring on the palms of the hands, in the strain creases.

A5.3 Hyperlipidaemia

The presence of abnormally high plasma lipid levels is called Hyperlipidaemia and is associated with an increased risk of cardiovascular disease. A high level of LDL can cause cholesterol to be deposited on the smooth muscle of arteries (artheroma) leading to restricted blood flow. Plasma lipid levels are affected by diet, smoking habits and exercise as well as genetic and hormonal influences. The diagnosis of the various types of hyperlipidaemia can be made after consideration of the patient's family history, plasma lipid levels (laboratory measurement), stored plasma appearance and cutaneous signs (xanthoma). It is particularly important to identify hereditary hyperlipidaemia since the patient's prognosis in these cases is much worse.

Hyperlipidaemia can be a primary disorder or can be secondary to an underlying disease state such as diabetes mellitus, alcoholism or renal failure. For about 25% of patients with

hyperlipidaemia, it is a secondary disorder (Mann & Ball, 1985) and in these cases the underlying disease is normally treated in the first instance.

Primary hyperlipidaemia can be divided into three broad categories according to the levels of plasma cholesterol and triglyceride: raised triglyceride only, raised cholesterol only and combined raised triglyceride and cholesterol. Raised triglyceride with a normal cholesterol level is an inherited condition which can be classified as endogenous (caused by over synthesis of triglyceride) or exogenous (caused by inadequate handling of ingested lipids).

Raised cholesterol with a normal triglyceride level (hypercholesterolaemia) can be classified into three types. If the raised cholesterol is due to a raised level of HDL, the condition can be left untreated since a raised HDL level is associated with a decreased risk of coronary disease. Familial monogenic hypercholesterolaemia is caused by a genetic defect in the LDL receptors on the surface of tissue cells, leading to a build up of LDL and hence plasma cholesterol. This disorder is indicated by a high or very high level of cholesterol, the presence of tendon xanthomas and a family history of ischaemic heart disease. Drug treatment is preferred to a modified diet alone since this is usually ineffective - without treatment there is a great risk of myocardial infarction and early death (under 60 years of age). Familial polygenic hypercholesterolaemia is quite commonly occurring and is indicated by a high cholesterol level with no strong family history of ischaemic heart disease. This condition usually responds well to dietary modification alone.

There are three types of combined hyperlipidaemia, in which cholesterol and triglyceride levels are both raised. Type III hyperlipidaemia (from the WHO classification) occurs when lipoproteins are not adequately broken down and converted to LDL.

This disorder is associated with an early incidence of peripheral vascular disease and the appearance of tuberous and linear palmar xanthomas. Familial combined hyperlipidaemia is a hereditary disorder that is indicated by high LDL and VLDL, the absence of xanthomas and a strong family history of ischaemic heart disease and/or raised cholesterol and triglyceride. Drug treatment can be used if there is no response to modified diet alone. In cases where there is no strong family history, familial polygenic hyperlipidaemia should be suspected and like familial polygenic hypercholesterolaemia this disorder should respond to a modified diet.

APPENDIX VI

KNOWLEDGE BASE FOR HYPERLIPIDAEMIA

DATA VARIABLES
cholesterol mmol/L
upper limit 20.000
lower limit 0.000
mean value none
std dev none
default none

blood pressure mmHg
upper limit 200.000
lower limit 0.000
mean value none
std dev none
default none

triglyceride mmol/L
upper limit 20.000
lower limit 0.000
mean value none
std dev none
default none

LDL mmol/L
upper limit 20.000
lower limit 0.000
mean value none
std dev none
default none

HDL mmol/L
upper limit 5.000
lower limit 0.000
mean value none
std dev none
default none

age years
upper limit 120
lower limit 0
mean value none
std dev none
default none

RELATIONSHIPS FOR DATA DERIVATION
1. LDL=cholesterol-HDL-triglyceride/2.2

SIGNS & SYMPTOMS
abdominal pain
present
absent
unknown

lipid electrophoresis
no B band
broad B band
unknown

stored plasma
clear/creamy
cloudy/creamy
cloudy
clear
unknown

pancreatitis
present
absent
unknown

xanthoma
tuberous
linear palmar
erruptive
tendon
absent
unknown

PATIENT HISTORY
sex
female
male
unknown

occupation
unknown

disorder
unknown

previous disorder
unknown

clinical diagnosis
unknown

smoking habits
0-5 per day
6-10 per day
11-15 per day
16-20 per day
over 20 per day
unknown

FH raised triglyceride
present
absent
unknown

FH raised cholestorol
present
absent
unknown

FH raised chol & TGC
present
absent
unknown

FH IHD
aged over 60
aged 50 to 60
aged below 50
unknown

FH PVD
aged above 60
aged 50 to 60
aged below 50
unknown

FH IHD & PVD
aged above 60
aged 50 to 60
aged below 50
unknown

PVD
present
absent
unknown

FRAMES FOR DISORDERS
CLASS: Hyperlipidaemia

Frame for Hyperlipidaemia

Frame for Normal Lipid Levels Type of Hyperlipidaemia (0.100)
history safe triglyceride present 1.000
history safe cholesterol present 0.500
history low cholesterol present 0.500

Frame for Primary H-L Type of Hyperlipidaemia (0.800)
history Thyroid Disease present 0.000
history Liver Disease present 0.000
history Alcoholism present 0.000
history Renal Disease present 0.000
history Obesity present 0.000
history Diabetes present 0.000
history Diabetes absent 1.000

Frame for Secondary H-L Type of Hyperlipidaemia (0.100)
history Thyroid Disease present 0.200
history Liver Disease present 0.200
history Alcoholism present 0.200
history Renal Disease present 0.200
history Obesity present 0.100
history Diabetes present 0.100

Frame for Exog Raised TGC Type of Raised Triglyceride (0.500)
symptom stored plasma clear/creamy 1.000

Frame for Endog Raised TGC Type of Raised Triglyceride (0.500)
symptom stored plasma cloudy 0.500
symptom stored plasma cloudy/creamy 0.500

Frame for Raised HDL Type of Raised Cholesterol (0.200)
history safe cholesterol present 0.300
history high cholesterol present 0.300
history raised cholesterol present 0.400
relation 1. HDL>=0.9 relation 2. LDL=<5.0

Frame for Famil Polygenic H-C Type of Raised Cholesterol (0.400)
symptom xanthoma absent 1.000
history high cholesterol present 0.400
history FH IHD aged below 50 0.000
history raised cholesterol present 0.600
history FH IHD aged over 60 0.600
history FH IHD aged 50 to 60 0.400
relation 1. LDL>=5 relation 2. HDL=<1.5

Frame for Famil Monogenic H-C Type of Raised Cholesterol (0.400)
symptom xanthoma absent 0.500
symptom xanthoma tendon 0.500
history high cholesterol present 0.400
history FH IHD aged below 50 0.300
history FH IHD aged over 60 0.100
history FH IHD aged 50 to 60 0.600
history v. high cholesterol present 0.500
history raised cholesterol present 0.100
relation 1. LDL>=5 relation 2. HDL=<1.5

Frame for Raised Triglyceride Type of Primary H-L (0.200)
symptom abdominal pain present 0.900
symptom abdominal pain unknown 0.100
symptom pancreatitis present 0.900
symptom pancreatitis absent 0.100
symptom lipid electrophoresis no B band 1.000
symptom xanthoma erruptive 0.800
symptom xanthoma absent 0.200
history raised cholesterol present 0.100
history safe cholesterol present 0.400
history low cholesterol present 0.500
history v high triglyceride present 0.400
history high triglyceride present 0.300
history safe triglyceride present 0.100
history raised triglyceride present 0.200

relation 1. triglyceride>=cholesterol

Frame for Raised Cholesterol Type of Primary H-L (0.400)
symptom stored plasma clear 1.000
symptom pancreatitis present 0.100
symptom pancreatitis absent 0.900
symptom abdominal pain present 0.100
symptom abdominal pain absent 0.900
symptom lipid electrophoresis no B band 1.000
history raised triglyceride present 0.100
history safe triglyceride present 0.900
history FH raised chol & TGC absent 0.900
history FH raised chol & TGC present 0.100
history FH raised triglyceride absent 0.900
history FH rsised triglyceride present 0.100
history FH raised cholestorol present 0.900
history FH raised cholestorol absent 0.100
history FH PVD aged above 60 0.800
history FH PVD aged 50 to 60 0.100
history FH PVD aged below 50 0.100
history FH IHD & PVD aged above 60 0.800
history FH IHD & PVD aged 50 to 60 0.100
history FH IHD & PVD aged below 50 0.100

Frame for Combined Type H-L Type of Primary H-L (0.400)
symptom pancreatitis present 0.100
symptom pancreatitis absent 0.900
symptom abdominal pain present 0.500
symptom abdominal pain absent 0.500
history v. high cholesterol present 0.400
history high cholesterol present 0.300
history high triglyceride present 0.300
history safe cholesterol present 0.100
history raised cholesterol present 0.200
history v high triglyceride present 0.200
history raised triglyceride present 0.500
relation 1. triglyceride=<cholesterol

Frame for Type III H-L Type of Combined Type H-L (0.100)
symptom lipid electrophoresis broad B band 1.000
symptom stored plasma cloudy/creamy 1.000
symptom xanthoma tuberous 0.400
symptom xanthoma absent 0.100
symptom xanthoma linear palmar 0.500

Frame for Famil Combined H-L Type of Combined Type H-L (0.450)
symptom xanthoma absent 1.000
symptom lipid electrophoresis no B band 1.000
symptom stored plasma cloudy 0.500
symptom stored plasma cloudy/creamy 0.500
history FH IHD & PVD aged below 50 0.900
history FH IHD & PVD aged 50 to 60 0.100
history FH PVD aged 50 to 60 0.100
history FH IHD aged 50 to 60 0.100
history FH IHD aged over 60 0.100
history FH IHD aged below 50 0.800
history FH PVD aged above 60 0.100
history FH PVD aged below 50 0.800
history PVD present 0.900
history PVD absent 0.100
history FH raised triglyceride present 0.900
history FH raised triglyceride absent 0.100
history FH raised cholestorol present 0.900
history FH raised cholestorol absent 0.100
history FH raised chol & TGC present 0.900
history FH raised chol & TGC absent 0.100

Frame for Famil Polygenic H-L Type of Combined Type H-L (0.450)
symptom stored plasma cloudy/creamy 0.500
symptom stored plasma cloudy 0.500
symptom xanthoma absent 1.000
symptom lipid electrophoresis no B band 1.000
history FH IHD aged below 50 0.100
history FH PVD aged above 60 0.100
history FH PVD aged below 50 0.100
history FH PVD aged 50 to 60 0.800
history FH IHD & PVD aged below 50 0.200
history FH IHD & PVD aged 50 to 60 0.800
history FH IHD & PVD aged above 60 0.000
history FH raised chol & TGC present 0.100
history FH raised cholestorol absent 0.100
history FH raised chol & TGC absent 0.900
history FH raised triglyceride absent 0.900
history FH raised cholestorol present 0.900
history FH raised triglyceride present 0.100
history PVD present 0.100
history PVD absent 0.900
history FH IHD aged 50 to 60 0.800
history FH IHD aged over 60 0.100

CLASS: CHD Risk Factor

Frame for CHD Risk Factor

Frame for High CRF Type of CHD Risk Factor (0.250)
history high BP level present 0.400
history very high BP level present 0.500
history raised BP level present 0.100
history low cholesterol present 0.100
history safe cholesterol present 0.100
history raised cholesterol present 0.200
history high cholesterol present 0.300
history v. high cholesterol present 0.300
history smoking habits 11-15 per day 0.100
history smoking habits 0-5 per day 0.100
history smoking habits 6-10 per day 0.100
history smoking habits over 20 per day 0.400
history smoking habits 16-20 per day 0.300

Frame for Moderately High CRF Type of CHD Risk Factor (0.250)
history smoking habits over 20 per day 0.200
history smoking habits 16-20 per day 0.300
history very high BP level present 0.200
history raised BP level present 0.400
history high BP level present 0.400
history v. high cholesterol present 0.100
history raised cholesterol present 0.300
history safe cholesterol present 0.200
history low cholesterol present 0.100
history high cholesterol present 0.300
history smoking habits 0-5 per day 0.100
history smoking habits 6-10 per day 0.100
history smoking habits 11-15 per day 0.300

Frame for Average CRF Type of CHD Risk Factor (0.250)
history smoking habits 11-15 per day 0.200
history smoking habits 6-10 per day 0.400
history smoking habits 0-5 per day 0.400
history raised cholesterol present 0.200
history low cholesterol present 0.300
history raised BP level present 0.200
history safe BP level present 0.500
history low BP level present 0.300
history v. high cholesterol present 0.100
history high cholesterol present 0.100
history safe cholesterol present 0.300

Frame for Low CRF Type of CHD Risk Factor (0.250)
history smoking habits 0-5 per day 1.000
history safe BP level present 0.500
history low BP level present 0.500
history safe cholesterol present 0.500
history low cholesterol present 0.500

CLASS: Blood Pressure Level

Frame for Blood Pressure Level

Frame for very high BP level Type of Blood Pressure Level (0.200)
relation 1. blood pressure>120

Frame for high BP level Type of Blood Pressure Level (0.200)
relation 1. blood pressure>110 relation 2. blood pressure=<120

Frame for raised BP level Type of Blood Pressure Level (0.200)
elation 1. blood pressure>100 relation 2. blood pressure=<110

Frame for safe BP level Type of Blood Pressure Level (0.200)
elation 1. blood pressure>90 relation 2. blood pressure=<100

Frame for low BP level Type of Blood Pressure Level (0.200)
relation 1. blood pressure=<90

CLASS: Cholesterol Level

Frame for Cholesterol Level

Frame for v. high cholesterol Type of Cholesterol Level (0.200)
relation 1. cholesterol>8.5

Frame for high cholesterol Type of Cholesterol Level (0.200)
relation 1. cholesterol>7.5 relation 2. cholesterol=<8.5

Frame for raised cholesterol Type of Cholesterol Level (0.200)
relation 1. cholesterol>6.5 relation 2. cholesterol=<7.5

Frame for safe cholesterol Type of Cholesterol Level (0.200)
relation 1. cholesterol>5.5 relation 2. cholesterol=<6.5

Frame for low cholesterol Type of Cholesterol Level (0.200)
relation 1. cholesterol=<5.5

CLASS: Triglyceride Level

Frame for Triglyceride Level

Frame for v high triglyceride Type of Triglyceride Level (0.250)
relation 1. triglyceride>10

Frame for high triglyceride Type of Triglyceride Level (0.250)
elation 1. triglyceride>4 relation 2. triglyceride=<10 F

rame for raised triglyceride Type of Triglyceride Level (0.250)
relation 1. triglyceride>1.8 relation 2. triglyceride=<4

Frame for safe triglyceride Type of Triglyceride Level (0.250)
relation 1. triglyceride=<1.8

FRAMES FOR DISEASES
CLASS: Primary disease

Frame for Primary disease

Frame for Thyroid Disease Type of Primary disease (0.160)

Frame for Liver Disease Type of Primary disease (0.160)

Frame for Alcoholism Type of Primary disease (0.160)

Frame for Renal Disease Type of Primary disease (0.160)

Frame for Obesity Type of Primary disease (0.160)

Frame for Diabetes Type of Primary disease (0.160)

APPENDIX VII

EVALUATION OF THE HYPERLIPIDAEMIA SYSTEM: CASE DATA

CASE 1

LAB DATA:

Cholesterol	8.8 mmol/L
Triglyceride	1.1 mmol/L
LDL	6.9 mmol/L
HDL	1.4 mmol/L

FAMILY HISTORY:

IHD	aged below 50

CLINICAL DATA:

age	53
sex	female
Smoking Habits	6-10/day
Xanthoma	absent
PVD	no

CASE 2

LAB DATA:

Cholesterol	9.6 mmol/L
Triglyceride	3.8 mmol/L
LDL	6.0 mmol/L
HDL	1.9 mmol/L

FAMILY HISTORY:

IHD	aged above 60

CLINICAL DATA:

age	64
sex	female
Smoking Habits	0-5/day
Xanthoma	absent
PVD	yes

CASE 3

LAB DATA:

Cholesterol	8.7 mmol/L
Triglyceride	2.1 mmol/L
LDL	6.6 mmol/L
HDL	1.1 mmol/L

FAMILY HISTORY:

IHD	aged above 60

CLINICAL DATA:

age	60
sex	male
Smoking Habits	0-5/day
Xanthoma	absent
PVD	no
Stored Plasma	clear

CASE 4

LAB DATA:

Cholesterol	8.9 mmol/L
Triglyceride	2.3 mmol/L
LDL	6.7 mmol/L
HDL	1.2 mmol/L

FAMILY HISTORY:

IHD	aged above 60
Raised Cholesterol	yes

CLINICAL DATA:

age	46
sex	male
Smoking Habits	0-5/day
Xanthoma	absent
PVD	no
Abdominal Pain	no

CASE 5

LAB DATA:

Cholesterol	8.2 mmol/L
Triglyceride	1.3 mmol/L
LDL	6.1 mmol/L
HDL	1.5 mmol/L

FAMILY HISTORY:

IHD	aged 50-60

CLINICAL DATA:

age	66
sex	male
Smoking Habits	0-5/day
Xanthoma	absent
PVD	yes

CASE 6

LAB DATA:

Cholesterol	9.8 mmol/L
Triglyceride	1.8 mmol/L

FAMILY HISTORY:

IHD	aged 50-60

CLINICAL DATA:

age	55
sex	female
Smoking Habits	0-5/day
Xanthoma	absent
Stored Plasma	clear

CASE 7

LAB DATA:

Cholesterol	9.7 mmol/L
Triglyceride	1.9 mmol/L

FAMILY HISTORY:

IHD	aged above 60
Raised Cholesterol	yes

CLINICAL DATA:

age	42
sex	female
Smoking Habits	over 20/day
Xanthoma	absent

CASE 8

LAB DATA:

Cholesterol	12.1 mmol/L
Triglyceride	1.6 mmol/L

FAMILY HISTORY:

IHD	aged below 50
Raised Cholesterol	yes

CLINICAL DATA:

age	33
sex	male
Smoking Habits	0-5/day
Xanthoma	absent
Abdominal Pain	no

CASE 9

LAB DATA:

Cholesterol	7.1 mmol/L
Triglyceride	1.2 mmol/L
LDL	4.4 mmol/L
HDL	2.1 mmol/L

FAMILY HISTORY:

IHD	aged below 50

CLINICAL DATA:

age	50
sex	female
Smoking Habits	0-5/day
Xanthoma	absent
PVD	no
Pancreatitis	no

CASE 10

LAB DATA:

Cholesterol	7.1 mmol/L
Triglyceride	2.1 mmol/L
LDL	4.8 mmol/L
HDL	1.3 mmol/L

CLINICAL DATA:

age	57
sex	female
Smoking Habits	0-5/day

CASE 11

LAB DATA:

Cholesterol	7.7 mmol/L
Triglyceride	2.4 mmol/L
LDL	5.8 mmol/L
HDL	0.8 mmol/L

FAMILY HISTORY:

IHD	aged above 60

CLINICAL DATA:

age	64
sex	female

CASE 12

LAB DATA:

Cholesterol	6.9 mmol/L
Triglyceride	3.9 mmol/L

CLINICAL DATA:

age	37
sex	male
Smoking Habits	16-20/day

CASE 13

LAB DATA:

Cholesterol	7.2 mmol/L
Triglyceride	1.3 mmol/L

FAMILY HISTORY:

IHD	aged below 50

CLINICAL DATA:

age	58
sex	male
Smoking Habits	0-5/day

CASE 14

LAB DATA:

Cholesterol	12.5 mmol/L
Triglyceride	0.6 mmol/L
HDL	1.27 mmol/L

FAMILY HISTORY:

IHD	aged below 50

CLINICAL DATA:

age	36
sex	female
Smoking Habits	0-5/day
Xanthoma	absent
Stored Plasma	clear

CASE 15

LAB DATA:

Cholesterol	7.4 mmol/L
Triglyceride	2.8 mmol/L
LDL	5.2 mmol/L
HDL	0.9 mmol/L

FAMILY HISTORY:

IHD	aged below 50

CLINICAL DATA:

age	44
sex	male
Smoking Habits	0-5/day
Abdominal Pain	yes

CASE 16

LAB DATA:

Cholesterol	6.7 mmol/L
Triglyceride	1.4 mmol/L
LDL	4.9 mmol/L
HDL	1.2 mmol/L

FAMILY HISTORY:

IHD	aged below 50
Raised Cholesterol	yes

CLINICAL DATA:

age	30
sex	male
Smoking Habits	0-5/day

CASE 17

LAB DATA:

Cholesterol	8.1 mmol/L
Triglyceride	3.7 mmol/L
LDL	5.7 mmol/L
HDL	0.7 mmol/L

FAMILY HISTORY:

IHD	aged below 50

CLINICAL DATA:

age	50
sex	male
Smoking Habits	6-10/day

CASE 18

LAB DATA:

Cholesterol	9.2 mmol/L
Triglyceride	4.5 mmol/L
LDL	6.2 mmol/L
HDL	1.0 mmol/L

FAMILY HISTORY:

IHD	aged 50-60

CLINICAL DATA:

age	41
sex	male
Smoking Habits	over 20/day
Stored Plasma	cloudy

CASE 19

LAB DATA:

Cholesterol	8.4 mmol/L
Triglyceride	2.1 mmol/L

FAMILY HISTORY:

IHD	aged 50-60

CLINICAL DATA:

age	53
sex	male
Smoking Habits	6-10/day
Xanthoma	absent

CASE 20

LAB DATA:

Cholesterol	7.7 mmol/L
Triglyceride	2.2 mmol/L
LDL	5.4 mmol/L
HDL	1.3 mmol/L

FAMILY HISTORY:

IHD	aged 50-60

CLINICAL DATA:

age	58
sex	male
Smoking Habits	0-5/day

CASE 21

LAB DATA:

Cholesterol	8.2 mmol/L
Triglyceride	1.0 mmol/L
LDL	6.5 mmol/L
HDL	1.2 mmol/L

FAMILY HISTORY:

IHD	aged below 50

CLINICAL DATA:

age	48
sex	male
Smoking Habits	0-5/day
Xanthoma	absent

CASE 22

LAB DATA:

Cholesterol	7.7 mmol/L
Triglyceride	0.9 mmol/L
LDL	6.1 mmol/L
HDL	1.2 mmol/L

CLINICAL DATA:

age	52
sex	male
Smoking Habits	16-20/day
Xanthoma	absent
PVD	no

CASE 23

LAB DATA:

Cholesterol	8.5 mmol/L
Triglyceride	1.6 mmol/L

FAMILY HISTORY:

IHD	aged above 60

CLINICAL DATA:

age	45
sex	male
Smoking Habits	0-5/day

CASE 24

LAB DATA:

Cholesterol	7.8 mmol/L
Triglyceride	3.9 mmol/L
LDL	6.5 mmol/L
HDL	1.2 mmol/L

CLINICAL DATA:

age	74
sex	female
Smoking Habits	0-5/day
Xanthoma	tendon

CASE 25

LAB DATA:

Cholesterol	7.5 mmol/L
Triglyceride	1.2 mmol/L

FAMILY HISTORY:

IHD	aged 50-60

CLINICAL DATA:

age	35
sex	male
Smoking Habits	0-5/day
Stored Plasma	clear

CASE 26

LAB DATA:

Cholesterol	6.7 mmol/L
Triglyceride	1.1 mmol/L

FAMILY HISTORY:

IHD	aged 50-60

CLINICAL DATA:

age	62
sex	female
Xanthoma	absent
Abdominal Pain	no

CASE 27

LAB DATA:

Cholesterol	6.8 mmol/L
Triglyceride	1.1 mmol/L
LDL	5.2 mmol/L
HDL	1.1 mmol/L

FAMILY HISTORY:

IHD	aged below 50

CLINICAL DATA:

age	60
sex	female
Smoking Habits	0-5/day
Xanthoma	absent

CASE 28

LAB DATA:

Cholesterol	8.7 mmol/L
Triglyceride	3.6 mmol/L

FAMILY HISTORY:

IHD	aged below 50

CLINICAL DATA:

age	42
sex	male
Smoking Habits	6-10/day

CASE 29

LAB DATA:

Cholesterol	7.3 mmol/L
Triglyceride	1.0 mmol/L

FAMILY HISTORY:

IHD	aged above 60

CLINICAL DATA:

age	68
sex	male

CASE 30

LAB DATA:

Cholesterol	7.3 mmol/L
Triglyceride	1.7 mmol/L

CLINICAL DATA:

age	59
sex	female
Smoking Habits	0-5/day

CASE 31

LAB DATA:

Cholesterol	8.5 mmol/L
Triglyceride	3.0 mmol/L

FAMILY HISTORY:

Raised Cholesterol	yes

CLINICAL DATA:

age	39
sex	female
Smoking Habits	0-5/day
Xanthoma	absent
Stored Plasma	cloudy/creamy

CASE 32

LAB DATA:

Cholesterol	6.84 mmol/L
Triglyceride	2.89 mmol/L

FAMILY HISTORY:

IHD	aged 50-60

CLINICAL DATA:

age	30
sex	male
Smoking Habits	6-10/day
Xanthoma	absent
PVD	yes

CASE 33

LAB DATA:

Cholesterol	8.9 mmol/L
Triglyceride	3.7 mmol/L

FAMILY HISTORY:

IHD	aged 50-60

CLINICAL DATA:

age	51
sex	male
Smoking Habits	0-5/day
Xanthoma	absent

CASE 34

LAB DATA:

Cholesterol	8.4 mmol/L
Triglyceride	2.1 mmol/L
LDL	6.5 mmol/L
HDL	0.9 mmol/L

FAMILY HISTORY:

IHD	aged above 60

CLINICAL DATA:

age	54
sex	male
Smoking Habits	0-5/day

CASE 35

LAB DATA:

Cholesterol	5.9 mmol/L
Triglyceride	0.7 mmol/L

FAMILY HISTORY:

IHD	aged below 50
Raised Cholesterol	yes

CLINICAL DATA:

age	52
sex	female

CASE 36

LAB DATA:

Cholesterol	7.5 mmol/L
Triglyceride	2.4 mmol/L

CLINICAL DATA:

age	62
sex	male
Smoking Habits	16-20/day

CASE 37

LAB DATA:

Cholesterol	8.0 mmol/L
Triglyceride	2.9 mmol/L

FAMILY HISTORY:

IHD	aged below 50

CLINICAL DATA:

age	23
sex	female
Smoking Habits	0-5/day

CASE 38

LAB DATA:

Cholesterol	13.1 mmol/L
Triglyceride	7.0 mmol/L

FAMILY HISTORY:

IHD	aged below 50

CLINICAL DATA:

age	35
sex	male
Xanthoma	tuberous
Lipid Electrophoresis	β band

CASE 39

LAB DATA:

Cholesterol	5.1 mmol/L
Triglyceride	6.1 mmol/L

CLINICAL DATA:

age	48
sex	male
Abdominal Pain	yes
Stored Plasma	clear/creamy

CASE 40

LAB DATA:

Cholesterol	5.3 mmol/L
Triglyceride	6.3 mmol/L

CLINICAL DATA:

age	48
sex	male
Abdominal Pain	yes
Stored Plasma	cloudy/creamy

CASE	EXPERT	SYSTEM
1	FM H-C	FM H-C 1.0
2	FC H-L	FC H-L 0.879
3	FM H-C	FM H-C 1.0
4	FM H-C	FP H-L 0.783, FM H-C 0.209
5	FM H-C	FP H-C 0.571, FM H-C 0.429
6	FM H-C	FM H-C 1.0
7	FM H-C	FP H-L 0.463, FC H-L 0.463
8	FM H-C	FM H-C 1.0
9	RAISED HDL	RAISED HDL 1.0
10	RAISED HDL	COMBINED TYPE 0.667, RAISED HDL 0.267
11	FP H-L	COMBINED TYPE 0.489, FP H-C 0.483
12	FP H-L	COMBINED TYPE 0.685, FP H-C 0.164
13	FP H-C	FP H-C 0.657, RAISED HDL 0.219, FM H-C 0.109
14	FM H-C	FM H-C 1.0
15	FC H-L	FC H-L 0.747
16	RAISED HDL	RAISED HDL 0.986
17	FC H-L	FC H-L 0.741, FM H-C 0.103
18	FC H-L	FP H-L 0.889, FC H-L 0.111
19	FC H-L	FP H-L 0.723
20	FP H-L	FP H-L 0.615, FM H-C 0.137

FP H-L is Familial Polygenic Hyperlipidaemia FC H-L is Familial Combined Hyperlipidaemia

FP H-C is Familial Polygenic Hypercholesterolaemia FM H-C is Familial Monogenic Hypercholesterolaemia

Table A7.1 Summary of Diagnoses for Hyperlipidaemia

CASE	EXPERT	SYSTEM
21	FM H-C	FM H-C 1.0
22	FP H-C	FP H-C 0.667, FM H-C 0.333
23	FM H-C	FP H-C 0.768, FM H-C 0.128, RAISED HDL 0.104
24	FM H-C	FM H-C 0.939
25	FM H-C	RAISED CHOLESTEROL 1.0
26	FP H-C	FP H-C 0.714, RAISED HDL 0.197
27	FP H-C	FM H-C 0.956
28	FC H-L	FC H-L 0.773
29	FP H-C	FP H-C 0.864, RAISED HDL 0.104
30	FP H-C	FP H-C 0.657, RAISED HDL 0.219, FM H-C 0.109
31	FC H-L	FC H-L 0.489, FP H-L 0.489
32	FC H-L	FP H-C 0.318, FC H-L 0.268, FP H-L 0.239
33	FC H-L	FP H-L 0.819, FC H-L 0.102
34	FC H-L	COMBINED TYPE 0.545, FP H-C 0.39
35	NORMAL	NORMAL 0.679, RAISED HDL 0.235
36	FP H-L	COMBINED TYPE 0.685, FP H-C 0.164
37	FC H-L	FC H-L 0.75
38	TYPE III H-L	TYPE III H-L 1.0
39	EXOGENOUS RAISED TGC	EXOGENOUS RAISED TGC 1.0
40	ENDOGENOUS RAISED TGC	ENDOGENOUS RAISED TGC 1.0

FP H-L is Familial Polygenic Hyperlipidaemia FC H-L is Familial Combined Hyperlipidaemia

FP H-C is Familial Polygenic Hypercholesterolaemia FM H-C is Familial Monogenic Hypercholesterolaemia

Table A7.1 Summary of Diagnoses for Hyperlipidaemia (continued)

APPENDIX VIII

OVERVIEW OF PROLOG

A8.1 Foundations

The computer language PROLOG (PROgramming in LOGic) was developed in the early 1970s by Colmerauer and Roussel as a logical theorem prover based on the principle of resolution (Robinson, 1965). A number of different implementations of PROLOG exist - the standard version was developed at Edinburgh University for use on a DECsystem10 computer and is usually refered to as *Edinburgh syntax* or *DEC10* PROLOG.

The language is built from basic data objects called *terms*, of which there are three types: constants, variables and structures. *Constants* can be integer numbers or atoms (denoted by a string of characters starting with a lower case letter or enclosed in single quotes). Certain special symbols such as {}, [], +, * are also defined as atoms. Depending on the implementation of PROLOG used strings and real numbers may also be available as constant types.

Variables are denoted by character strings starting with a capital letter or underscore and represent any other, unspecified PROLOG term. Examples of variables are:

A Attribute Value V _value _

The variable denoted by a single underscore, '_', is the special anonomous variable that can represent several unrelated terms in the same expression when it is not necessary to know what the terms are, merely that they exist.

Structures are compound terms comprising a *functor* and one or more *arguments*. The functor is described by its name, which must be an atom, and the number of arguments it takes, called its arity. An example is given below of a PROLOG structure with the functor current_data/3 (*ie* its name is current_data and its arity is 3)

current_data(lab_data,"pH",7.41)

A list is a PROLOG structure that has a special notation: a sequence of terms separated by commas and enclosed in square brackets. For example

[symptom,lab_data,history]

The empty list [] is an atom and other lists are represented in the standard PROLOG syntax as structures with the functor . and two arguments, the head (which is the first element of the list) and the tail (which is the list of all remaining elements). Using this standard syntax, the list presented above would be written:

.(symptom,.(lab_data,.(history,[])))

which is rather cumbersome - hence the special square bracket notation. Within the square brackets, the head and tail of the list can be explicitly represented using the symbol |:

[symptom|lab_data,history]

PROLOG clauses are structures comprising a head (distinct from the head of a list) and a body, separated by the symbol :- (which can be read as *if*). The head of a clause is a goal and the body is a sequence of goals, so that the clause

A:-B,C.

can be read as *the goal A is true if sub-goals B and C are true.* A goal is a normal PROLOG structure with a functor and arguments - its functor is known as a predicate. The clause presented above might take the form:

has_been_observed(X):-variable(X),current_data(variable,X,Value).

A8.2 Programming in PROLOG

A PROLOG program is a database of clauses that are of two types: clauses that have no body (ie they are always true) can be considered as facts; clauses that have one or more goals in the body can be considered as rules. The program is *run* by posing a query, comprising the special symbol ?- and a sequence of one or more goals. The PROLOG interpreter attempts to satisfy the goals by matching them with the heads of clauses in the database, returning the answer *yes* if the goals succeed or *no* if they do not.

```
#1    observation("pH").
#2    obseravtion("pCO2").
#3    current_data(lab_data,"pCO2",5.2).
#4    has_been_observed(X):-
      observation(X),current_data(lab_data,X,,Value).
```

Figure A8.1 A Simple PROLOG program. The numbers for the clauses have been added to allow for reference in the text.

Consider, for example, the simple program shown in Figure A8.1. After posing the query ?- has_been_observed("pCO2"), the PROLOG interpreter attempts to satisfy the goal has_been_observed by searching the database for a clause with the head has_been_observed. The match is found with clause #4, its variable argument is instantiated to "pCO2", and an attempt is made to satisfy the sub-goals in its body. The first sub-goal observation(X) has the same variable argument as has_been_observed, which has already been instantiated to "pCO2". So the interpreter searches for observation("pCO2"), which it finds as clause #2, thus satisfying the sub-goal. The next sub-goal is evaluated by searching for current_data(lab_data,"pCO2",Value) - a match is found with clause #3 with the variable Value instantiated to 5.2. Both the sub-goals in the body of has_been_observed are now satisfied, so the original goal succeeds and the query returns *yes*.

Suppose that the query ?-has_been_observed(X) is posed. As before, clause #4 is matched and the sub-goal observation(X) is persued. However in this instance the argument to observation/1 is an uninstantiated variable and the first match is found with clause #1, instantiating X to "pH". A search is now made for current_data(lab_data,"pH",Value), but no match is found with the clauses in the database. The PROLOG interpreter now exhibits a behaviour called backtracking. Since the second sub-goal to has_been_observed failed, the interpreter backtracks to the first sub-goal and attempts to re-satisfy it by finding a different match for observation(X) and hence a different instantiation for the variable X. The sub-goal is re-satisfied by matching with clause #2, instantiating X to "pCO2". The second sub-goal current_data(lab_data,"pCO2", Value) now succeeds by matching with clause #3 and the original goal succeeds with its argument instantiated to "pCO2" - the query returns *yes* and the the value "pCO2".

The clauses that make up a PROLOG program are placed on the database using the predicate assert/1 which is defined as part of the language (the argument to assert is the clause to be added to the database). Clauses can be removed from the database by using the predicate retract/1. The two predicates assert/1 and retract/1 can can be used as sub-goals in PROLOG clasues themselves, providing the facility to dynamically alter the database whilst a program is running - note that there is no distinction between those clauses asserted as part of the program itself and those asserted whilst it is running.

Another important feature of PROLOG programming is the use of recursion (*ie* clauses that include themselves as a sub-goal in the body). Consider for example the simple recursive definition for the clause member/2 which checks whether its first argument is a member of the list in its second argument:

```
member(A, [A|L]).

member(A, [B|L]) :-
member(A,L).
```

The first clause for member/2 states that any element A is a member of the list which has A as its head. The second clause states that A is a member of a list if it is a member of the tail of the list.

The matching of two terms by the PROLOG interpreter is achieved using the principle of resolution discovered by Robinson (1965) which finds the most general match (*ie* the one with the least number of variable instantiations). Generally speaking, the head of a logical clause could contain any number of terms - the PROLOG notation is restricted to horn clauses in which the head contains one term at the most. This restriction allows a more efficient implementation of the resolution theorem prover. A full discussion of the relation of PROLOG to logic is given in Clocksin & Mellish (1984) which also serves as the standard for Edinburgh syntax PROLOG.

APPENDIX IX

PROGRAM LISTINGS

The complete Laboratory Data Analysis System consists of a top level program from which knowledge bases can be loaded from application sub-directories and where the FRAMBUILDER knowledge editing environment and the diagnostic system can used with the selected knowledge base. The files that make up the complete system are:

In the top level directory (\lda):

lda.run	front end and overall supervisor
utility.mod	some general utilities
applics.arc	archive of the applications available
lexicon.cor	the core lexicon (domain independent)
builder.run	loading FRAMEBUILDER
build.utl	FRAMEBUILDER utilities
build.gen	1st level of FRAMEBUILDER
build.tre	2nd level of FRAMEBUILDER
build.frm	3rd level of FRAMEBUILDER
dasystem.run	loading the diagnostic system
das.utl	diagnostic system utilities
das.blk	blackboard control and triggering
das.ks	knowledge sources
das.ipt	data input/output facilities
das.dia	dialogue management

In each application sub-directory:

kbase.dat	knowledge base
patients.dat	patient archive
lexicon.dat	application-specific lexicon

The system is supplied on two floppy disks - one with the files listed above and the other with a version of the Prolog 2 environment. The machine used must be compatible with an IBM AT with a hard disk and at least 512k of working memory. It should also be licensed to run Prolog 2. The Prolog 2 disk should be copied into the directory \P2 and the system files into \lda. The application subdirectories are named \lda\0, \lda\1, etc - these should be recreated on the hard disk and copied from the floppy disk. To run the system type *ldas* in the \P2 directory.

```
/* lda.run front end and overall supervisor */

/* Predicates defined in this file:

run_lda/0
set_state/0
load_application_directory/0
save_application_directory/0
save_applications_directory/0
draw_lda_backdrop/0
remove_lda_backdrop/0
draw_lda_footer/0
set_lda_menu/0
remove_lda_menu/0
display_lda_master/0
lda_action/1
define_menu_entry/1
select_application/0
set_current_application/1
display_application/0
add_application/0
set_application_directory/1
select_index/1
clear_knowledge_base/0   */

/* ---------------------------------------------------------------- */
/* ------------------------      KICKOFF      ------------------------ */

run_lda:-
set_state,
open_module(lda_mod,"\\lda\\utility.mod",none,actual),
load_application_directory,
draw_lda_backdrop,
display_application,
display_lda_master,
save_application_directory,!,
close_module(lda_mod),
remove_lda_backdrop.

/* set the state of the Prolog2 environment */

set_state:-
state(window_exception,_,lda_exception),
state(decimals,_,3).

/* load the applications */

load_application_directory:-
exists_file("\\lda\\applics.arc"),
reconsult("\\lda\\applics.arc").

load_application_directory:-
create("\\lda\\applics.arc").

/* save the applications */

save_application_directory:-
(delete_file("\\lda\\applics.tmp");true),
create("\\lda\\applics.tmp"),
create_stream(applics,readwrite,ascii,file("\\lda\\applics.tmp")),
open(applics,readwrite),
state(output,_,applics),
write_clauses(application/2),
write_clauses(current_application/2),
close(applics),
delete_stream(applics),
(delete_file("\\lda\\applics.arc");true),
rename_file("\\lda\\applics.tmp","\\lda\\applics.arc").

save_applications_directory.

/* ---------------------------------------------------------------- */
/* ------------------------BACKDROP ROUTINES------------------------ */

draw_lda_backdrop:-
create_stream(lda_backdrop,readwrite,byte,window(25,80,red on black)),
open(lda_backdrop,readwrite),
screen(lda_backdrop,create(0,0,lda_backdrop,0,0,0,none,None,25,80,reveal
ed)),
create_stream(lda_header,readwrite,byte,window(1,80,bright white on
red)), open(lda_header,readwrite),
screen(lda_header,create(0,0,lda_header,0,0,0,none,None,1,80,hidden)),
window(lda_header,cursor_address(0,0)),
window(lda_header,text(" Laboratory Data Analysis System  V1.2 ")),
screen(lda_header,unhide),
create_stream(lda_footer,readwrite,byte,window(1,80,bright white on
red)), open(lda_footer,readwrite),
screen(lda_footer,create(24,0,lda_footer,0,0,0,none,none,1,80,hidden)),
draw_lda_footer,
screen(lda_footer,unhide).

remove_lda_backdrop:-
close(lda_header),
delete_stream(lda_header),
close(lda_backdrop),
delete_stream(lda_backdrop),
close(lda_footer),
delete_stream(lda_footer).

draw_lda_footer:-
window(lda_footer,cursor_address(0,1)),
window(lda_footer,text("Current Application:")).


/* ---------------------------------------------------------------- */
/* --------------------DEEFINE THE MASTER MENU---------------------- */

set_lda_menu:-
create_stream(lda,readwrite,byte,window(6,22,white on black)),
open(lda,readwrite),
screen(lda,create(3,1,lda,0,0,0,all,black on red,6,22,hidden)),
retractall(menu_selection(_,_)),
assert(menu_selection(lda,0)),
assert(menu_selection(applications,0)).

remove_lda_menu:-
close(lda),
delete_stream(lda).


display_lda_master:-
set_lda_menu,
repeat,
once(menu_selection(lda,Selection)),
once(menu(lda,"LDA MASTER",
["Select Application"-"@"-true-0-help, "Add Application"-"@"-true-1-
help, "Edit Knowledge Base"-"@"-true-2-help, "Diagnostic System"-"@"-
true-3-help, ""-"@"-true-4-help,
"Quit"-"@"-true-5-help],Option,Selection)), once(screen(lda,unhide)),
once(retract(menu_selection(lda,Selection))),
once(assert(menu_selection(lda,Option))),
once(lda_action(Option)),
Option==5,                          /* fails until Exit selected */
remove_lda_menu.


lda_action(0):-
select_application.

lda_action(1):-
add_application.

lda_action(2):-
current_application(Application,Index),
Directory is_string "\\lda\\" & Index,
chdir(Directory),
reconsult("\\lda\\builder.run"),
chdir("\\lda"),
clear_knowledge_base.

lda_action(3):-
current_application(Application,Index),
Directory is_string "\\lda\\" & Index,
chdir(Directory),
reconsult("\\lda\\dasystem.run"),
chdir("\\lda"),
clear_knowledge_base.

lda_action(Selection).              /* catches failures */


/* ---------------------------------------------------------------- */
/* -------------------- SELECTION OF APPLICATION -------------------- */

/* define an entry in the selection menu */

define_menu_entry(Menu_entry):-
application(Application,_),
Menu_entry=Application-"@"-true-set_current_application(Application)-
help.

/* main loop for selecting application */

select_application:-
create_stream(applications,readwrite,byte,window(80,24,white on red)),
open(applications,readwrite),
screen(applications,create(3,28,applications,0,0,0,all,black on
red,3,24,hidden)), predicate_size(application/2,N),
((N<22,Menu_size is N);(Menu_size is 22)),
screen(applications,change(3,28,applications,0,0,0,all,black on
red,Menu_size,24,hidden)),
bagof(Menu_entry,define_menu_entry(Menu_entry),Menu_entry_list),
menu_selection(applications,Selection),
menu(applications,"SELECT
APPLICATION",Menu_entry_list,Option,Selection),
screen(applications,unhide),
window(applications,inquire_cursor_address(Y,X)),
New_selection is Y-1,
once(retract(menu_selection(applications,Selection))),
once(assert(menu_selection(applications,New_selection))),
call(Option),
close(applications),
delete_stream(applications).

/* set the selected current application */

set_current_application(Entry):-
retractall(current_application/2),
application(Entry,Index),
assert(current_application(Entry,Index)),
display_application.

/* display the current application */

display_application:-
current_application(Name,Index),
window(lda_footer,cursor_address(0,23)),
fill_out(Name,Display,23),
window(lda_footer,text(Display)).

display_application.

/* ---------------------------------------------------------------- */
/* -------------------- CREATION OF APPLICATION -------------------- */

/* add a new application */

add_application:-
fedit(5,40,23,"NEW APPLICATION","","",white on blue,Entry),
truncate_string(Entry,Application),
set_application_directory(Application).

/* set the directory for the new application */

set_application_directory("").

set_application_directory(Application):-
application(Application,_),
warning_box(5,40,"APPLICATION EXISTS").

set_application_directory(Application):-
select_index(Index),
assert(application(Application,Index)),
Directory is_string "\\lda\\" & Index,
```

```
mkdir(Directory).

/* Select the index number for the new application */

select_index(Index):-
repeat(X),
NI is_string string(X,ops),
not application(_,NI),
Index=NI.

/* ---------------------------------------------------------------- */
/* ------------ CLEARING K-BASE BETWEEN APPLICATIONS ---------------- */

clear_knowledge_base:-
retractall(current_data/3),
retractall(disorder/1),
retractall(disease/1),
retractall(disorder/3),
retractall(disease/3),
retractall(link/4),
retractall(variable/1),
retractall(data_parameter/3),
retractall(symptom/2),
retractall(relation/2),
retractall(history/2),
retractall(frame_variable/5),
retractall(frame_symptom/5),
retractall(frame_relation/4),
retractall(frame_history/5).
```

```
/* utility.lda some general utilities */

/* Predicates defined in this file:

lda_exception/2
sub_category/4
descendent/4
predecessor/3
display_decision_box/3
decision_box/4
decision_action/3
display_warning_box/3
warning_box/3
warning_action/3
write_clauses/1
number_of_clauses/3
member/2
subset/2
intersect/3
delete/3
append/3
next_element/3
substitute/4
sum_list/3
product_list/3
fill_out/3
rule_out/3
truncate_string/2
replace_character/4
delete_character/3
scroll_window_up/1
scroll_window_down/1
scroll_to_foot/1
write_window_entry/2
get_window_entry/2        */


/* ------------------------------------------------------------------ */
/* --------------------HANDLING PROLOG2 ERRORS---------------------- */

/* Trapping window exceptions */

lda_exception(line_no_room,window(Window,text(" "))):-
window(Window,text("\n\r")).

lda_exception(line_no_room,window(Window,text(Text))):-
window(Window,inquire_cursor_address(Y,X)),
NewY is Y+1,
window(Window,cursor_address(NewY,0)),
window(Window,text(Text)).

lda_exception(window_full,window(Window,text(Text))):-
window(Window,scroll_up),
window(Window,text(Text)).

lda_exception(cursor_undefined,window(Window,text(Text))):-
window(Window,scroll_up),
window(Window,text(Text)).

lda_exception(window_full,_):-
fail.

lda_exception(A,B).


/* ------------------------------------------------------------------ */
/* --------------------------TREE PROCESSING----------------------- */

/* tree utilities */


sub_category(Type,Root,Parent,Entry):-
Record=..[Type,Root,Parent,Entry_list],
call(Record),
member(Entry,Entry_list).


/* To find a descendent of a node */

descendent(Type,Root,Entry,Descendent):-
sub_category(Type,Root,Entry,Descendent).

descendent(Type,Root,Entry,Descendent):-
link(Root,Parent,Descendent,_),
descendent(Type,Root,Entry,Parent).


/* to find the ancestor of a node */

predecessor(Root,Entry,Predecessor):-
link(Root,Predecessor,Entry,_).

predecessor(Root,Entry,Predecessor):-
link(Root,Intermediate,Entry,_),
predecessor(Root,Intermediate,Predecessor).


/* ------------------------------------------------------------------ */
/* ----------------------GENERAL UTILITIES------------------------- */

/* Decision Box */

display_decision_box(Y,X,Title):-
screen(decision,create(Y,X,decision,0,0,0,all,red on cyan,3,24,hidden)),
window(decision,cursor_address(0,1)),
window(decision,text(Title)),
window(decision,attribute(cyan on red)),
window(decision,cursor_address(2,2)),
window(decision,text("   OK   ")),
window(decision,cursor_address(2,14)),
window(decision,text(" CANCEL ")),
screen(decision,unhide).


decision_box(Y,X,Title,Result):-
create_stream(decision,readwrite,byte,window(3,24,red on cyan)),
open(decision,readwrite),
display_decision_box(Y,X,Title),
BY is Y+2,
BX is X+23,
repeat,
locator(Y,X,RY,RX,Y,X,BY,BX),
OY is RY-Y,
OX is RX-X,
decision_action(OY,OX,Result),
Result\=continue,
close(decision),
delete_stream(decision).


decision_action(2,OX,ok):-
OX>1,
OX<10.

decision_action(2,OX,cancel):-
OX>13,
OX<22.

decision_action(OY,OX,continue).


/* Warning Box */

display_warning_box(Y,X,Title):-
screen(warning,create(Y,X,warning,0,0,0,all,white on red,3,24,hidden)),
window(warning,cursor_address(0,1)),
window(warning,text(Title)),
window(warning,cursor_address(2,2)),
window(warning,text(" CONTINUE ")),
screen(warning,unhide).


warning_box(Y,X,Title):-
create_stream(warning,readwrite,byte,window(3,24,white on red)),
open(warning,readwrite),
display_warning_box(Y,X,Title),
BY is Y+2,
BX is X+23,
repeat,
locator(Y,X,RY,RX,Y,X,BY,BX),
OY is RY-Y,
OX is RX-X,
warning_action(OY,OX,Result),
Result=continue,
close(warning),
delete_stream(warning).


warning_action(2,OX,continue):-
OX>2,
OX<11.

warning_action(OY,OX,loop).


/* writing clauses to file */

write_clauses(Predicate):-
clause(Predicate,Clause,N),
once(writeq(Clause)),
once(write(".\r\n")),
fail.

write_clauses(Predicate).


/* to find the number of clauses of a certain form */

number_of_clauses(Predicate,Clause,Number):-
bagof((N,Clause),clause(Predicate,Clause,N),Clause_list),
length(Clause_list,Number).

number_of_clauses(Predicate,Clause,0).


member(Element,[Element|_]).                    /* find member of a list */

member(Element,[_|Rest_of_list]):-
member(Element,Rest_of_list).


subset([A|X],Y):-                               /* finds a subset of a list */
member(A,Y),
subset(X,Y).

subset([],Y).


intersect([],D2,[]).                            /* find the intersection */

intersect([M|D1],D2,[M|I]):-
member(M,D2),
intersect(D1,D2,I).

intersect([M|D1],D2,I):-
intersect(D1,D2,I).

delete(_,[],[]).                                /* delete an element from a list */

delete(X,[X|L],M):- !,delete(X,L,M).

delete(X,[Y|L1],[Y|L2]):- delete(X,L1,L2).


append([],L,L).                                 /* append a list to a list */

append([X|L1],L2,[X|L3]):-append(L1,L2,L3).


next_element(E,N,[E,N|_]).                      /* find the next element in a list */

next_element(E,N,[_|L]):-
next_element(E,N,L).
```

```
substitute(_,[],_,[]).                /* substitute an element in a list */

substitute(X,[X|L],A,[A|M]):-
!,
substitute(X,L,A,M).

substitute(X,[Y|L],A,[Y|M]):-
substitute(X,L,A,M).


sum_list([],S,S).                              /* sums a list of values */

sum_list([V|L],S_so_far,S):-
New_S_so_far is S_so_far+V,
sum_list(L,New_S_so_far,S).


product_list([],P,P).                     /* multiplies a list of values */

product_list([V|L],P_so_far,P):-
New_P_so_far is P_so_far*V,
product_list(L,New_P_so_far,P).

fill_out(Input_string,Output_string,Length):-
length(Input_string)=Length,
Output_string=Input_string.

fill_out(Input_string,Output_string,Length):-
length(Input_string)>Length,
Current_length is length(Input_string),
Output_string is_string delete(Input_string,Length,(Current_length-
Length)).

fill_out(Input_string,Output_string,Length):-
New_string is_string Input_string & " ",
fill_out(New_string,Output_string,Length).


rule_out(Input_string,Output_string,Length):-
length(Input_string)=Length,
Output_string=Input_string.

rule_out(Input_string,Output_string,Length):-
length(Input_string)>Length,
Current_length is length(Input_string),
Output_string is_string delete(Input_string,Length,(Current_length-
Length)).

rule_out(Input_string,Output_string,Length):-
New_string is_string Input_string & "\196",
rule_out(New_string,Output_string,Length).


truncate_string("","").

truncate_string(String,String):-
E is length(String)-1,
End is_string substring(String,E,1),
End\=" ".

truncate_string(String,New_string):-
E is length(String)-1,
Next_string is_string delete(String,E,1),
truncate_string(Next_string,New_string).



replace_character(String,String,Character,Replacement):-
L is length(String),
I is index(String,Character,0),
L=I.

replace_character(String,New_string,Character,Replacement):-
P is index(String,Character,0),
Temp_string is_string delete(String,P,1),
Next_string is_string insert(Temp_string,Replacement,P),
replace_character(Next_string,New_string,Character,Replacement).


delete_character(String,String,Character):-
L is length(String),
I is index(String,Character,0),
L=I.

delete_character(String,New_string,Character):-
P is index(String,Character,0),
Next_string is_string delete(String,P,1),
delete_character(Next_string,New_string,Character).


/* WINDOW DISPLAY UTILITIES */

scroll_window_up(Window):-
stream(Window,_,_,window(Y,X,_),_),
screen(Window,info(SY,SX,Window,WY,WX,D,M,Matt,H,W,R)), WY<Y-H,
New_WY is WY+1,
screen(Window,change(SY,SX,Window,New_WY,WX,D,M,Matt,H,W,R)).

scroll_window_up(Window).


scroll_window_down(Window):-
screen(Window,info(SY,SX,Window,WY,WX,D,M,Matt,H,W,R)), WY>0,
New_WY is WY-1,
screen(Window,change(SY,SX,Window,New_WY,WX,D,M,Matt,H,W,R)).

scroll_window_down(Window).


scroll_to_foot(Window):-
once(screen(Window,info(SY,SX,Window,WY,WX,D,M,Matt,H,W,R))),
once(BY is WY+H-1),
once(window(Window,cursor_address(BY,0))),
once(window(Window,inquire_text(W,Bottom_line))),
once(truncate_string(Bottom_line,Test)),
Test="".

scroll_to_foot(Window):-
scroll_window_up(Window).

scroll_to_foot(Window).


write_window_entry(Window,no_data).          /* traps case of no_data */

write_window_entry(Window,Entry):-
window(Window,text(Entry)),
window(Window,inquire_cursor_address(Y,X)),
NY is Y+1,
window(Window,cursor_address(NY,1)).

write_window_entry(Window,Entry).

get_window_entry(Window,Entry):-
screen(Window,info(SY,SX,Window,PWY,PWX,D,M,Mat,H,W,R)),
Entry_length is W-2,
Xend is SX+W-1,
Yend is SY+H-2,
locator(SY,SX,Y,X,SY,SX,Yend,Xend),
WY is Y-SY,
AY is WY+PWY,                                          /* absolute Y pos */
window(Window,cursor_address(AY,1)),
window(Window,inquire_text(Entry_length,Entry)).
```

```
/* builder.run loading FRAMEBUILD R */

/* Predicates defined in this file:

load_system/0
remove_system/0
read_data/0
store_data/1
run_frame_builder/0 */

/* ------------------------------------------------------------------ */
/* -------- MAIN PROGRAM LOOP AND ACTIONS ------- */

/* load the frame builder system */

load_system:-
open_module(mod1,"\\lda\\build.gen",none,actual),
open_module(mod2,"\\lda\\build.tre",none,actual),
open_module(mod3,"\\lda\\build.frm",none,actual),
open_module(mod4,"\\lda\\build.utl",none,actual).

remove_system:-
close_module(mod1),
close_module(mod2),
close_module(mod3),
close_module(mod4).

/* loading/storing the knowledge base */

read_data:-
read_kbase,
load_lexicon.


store_data(ok):-
store_kbase,
store_lexicon.

store_data(Result).


/* main loop */

run_frame_builder:-
load_system,
draw_backdrop,
read_data,
set_protected_slots,
define_main_command_line,
main_command_loop,
remove_backdrop.


?- run_frame_builder.
?- remove_system.
```

```
/* build.utl FRAMEBUILDER utilities */

/* Predicates defined in this file:

related_to/4
get_tree_nodes/3
display_warning_box/5
warning_box/5
w_action/3
get_window_entry/3
draw_backdrop/0
remove_backdrop/0
value_display/2
define_input_window/2
string_input/1
get_rest_of_string/3
in_string/1
change_stream/2
enter_text/3
convert_value/2
enter_data/3
string_to_atoms/2
list_to_atoms/3
form_atom/2
replace_synonym/2
in_word/2
update_lexicon/3
load_lexicon/0
store_lexicon/0
get_rank/4
read_kbase/0
store_kbase/0
print_database/1
print_line/4
print_indented_list/1
print_continuous_list/1
print_variables/0
print_symptoms/0
print_history/0
print_relations/0
print_frame_class/2
print_frame_data/2
print_disorder_frames/0
print_disease_frames/0                    */


/* TREE PROCESSING UTILITIES */

/* To see if two nodes are related */

related_to(Type,Root,Node,Node).

related_to(Type,Root,Node1,Node2):-
descendent(Type,Root,Node1,Node2).

related_to(Type,Root,Node1,Node2):-
descendent(Type,Root,Node2,Node1).


/* Find all the nodes for a tree */

get_tree_nodes(Type,Root,[Root|Node_list]):-
bagof(Node,descendent(Type,Root,Root,Node),Node_list),!.

get_tree_nodes(Type,Root,[Root]).


/* GENERAL UTILITIES */

/* Warning Box */

display_warning_box(Y,X,M1,M2,M3):-
screen(warning,create(Y,X,warning,0,0,0,all,white on red,5,24,hidden)),
window(warning,cursor_address(0,1)),
window(warning,text(M1)),
window(warning,cursor_address(1,1)),
window(warning,text(M2)),
window(warning,cursor_address(2,1)),
window(warning,text(M3)),
window(warning,attribute(red on white)),
window(warning,cursor_address(4,6)),
window(warning,text(" CONTINUE ")),
screen(warning,unhide).


warning_box(Y,X,M1,M2,M3):-
create_stream(warning,readwrite,byte,window(5,24,white on red)),
open(warning,readwrite),
display_warning_box(Y,X,M1,M2,M3),
BY is Y+4,
BX is X+23,
repeat,
locator(Y,X,RY,RX,Y,X,BY,BX),
OY is RY-Y,
OX is RX-X,
w_action(OY,OX,Result),
Result=continue,
close(warning),
delete_stream(warning),!.            /* dont want backtracking here */


w_action(4,OX,continue):-
OX>5,
OX<16.

w_action(OY,OX,loop).


/* WINDOW UTILITIES */

get_window_entry(Window,Xpos,Entry):-
window_drop(Window,Drop),
Drop>0,                                  /* fails if no entries */
drop_window_width(Window,Width),
Entry_length is Width-2,
Xend is Xpos+Width-1,
locator(2,Xpos,Y,X,2,Xpos,Drop,Xend),
WY is Y-2,
screen(Window,info(_,_,Window,OY,_,_,_,_,_,_,_)),  /* get scroll info */
AY is WY+OY,                                       /* absolute Y pos */
window(Window,cursor_address(AY,1)),
window(Window,inquire_text(Entry_length,Display)),
truncate_string(Display,Entry),!,
Entry\="".                                  /* fails if nothing there */


draw_backdrop:-
create_stream(backdrop,readwrite,byte,window(25,80,black on black)),
open(backdrop,readwrite),
screen(backdrop,create(0,0,backdrop,0,0,0,none,None,25,80,revealed)).

remove_backdrop:-
close(backdrop),
delete_stream(backdrop).


value_display(none," none  ").

value_display(Value,Display):-
Value=< -10.000,
Value>= -999.999,
Display is_string string(Value,ops).

value_display(Value,Display):-
Value< 0.000,
Value> -10.000,
Display is_string " " & string(Value,ops).

value_display(Value,Display):-
Value>= 0.000,
Value< 10.000,
Display is_string "  " & string(Value,ops).

value_display(Value,Display):-
Value>= 10.000,
Value=< 99.999,
Display is_string " " & string(Value,ops).

value_display(Value,Display):-
Value>= 100.000,
Value=< 999.999,
Display is_string string(Value,ops).


value_display(Value,"       ").

/* ------------------------------------------------------------------*/
/* INPUT & DISPLAY 7 CHARACTER STRING ROUTINES */


define_input_window(Y,X):-
create_stream(input,readwrite,byte,window(1,7,bright red on cyan)),
open(input,readwrite),
screen(input,create(Y,X,input,0,0,0,none,None,1,7,hidden)),
window(input,cursor_home),
window(input,text("       ")),
window(input,cursor_home),
screen(input,unhide).

define_input_window(Y,X):-                  /* close on backtracking */
close(input),
delete_stream(input),!,
fail.


/* input string from current stream */

string_input(String):-
get0(C),
get_rest_of_string(C,"",String).


get_rest_of_string(13,String,String).

get_rest_of_string(C,S,String):-
in_string(C),
S1 is_string S & [C],
get0(C1),
get_rest_of_string(C1,S1,String).

get_rest_of_string(C,S,String):-
get0(C1),
get_rest_of_string(C1,S,String).

in_string(C):- C>31,C<123.


/* change the current input stream */

change_stream(Old,New):-
see(New).

change_stream(Old,New):-            /* revert to Old on backtracking */
see(Old),!,
fail.


/* routine enter text data */

enter_text(Y,X,String):-
define_input_window(Y,X),
seeing(Stream),
change_stream(Stream,input),
string_input(Input),
truncate_string(Input,String),
see(Stream),
close(input),
delete_stream(input).

/* convert values to real numbers */

convert_value(Value,Value):-
real(Value).
```

```
convert_value(V,Value):-
Value is float(V).

/* routine enter numerical data value */

enter_data(Y,X,Value):-
define_input_window(Y,X),
seeing(Stream),
change_stream(Stream,input),
once(string_input(Input)),
once(truncate_string(Input,String)),
V is value(String,ir),
convert_value(V,Value),
see(Stream),
close(input),
delete_stream(input).

enter_data(Y,X,none).

/* ---------------------------------------------------------------- */
/* ---------------- ROUTINES TO UPDATE THE LEXICON ---------------- */

/* Change a string to a list of atoms */

string_to_atoms(String,Atom_list):-
list(List,String),
list_to_atoms(List,"",Atom_list).

string_to_atoms(S,A):-
warning_box(10,30,S,"cannot be input",""),
fail.

/* Change a list of characters to a list of atoms */

list_to_atoms([],String,[Atom]):-
form_atom(String,Atom).

list_to_atoms([32|List],String,[Atom|Atom_list]):-
form_atom(String,Atom),
list_to_atoms(List,"",Atom_list).

list_to_atoms([C|List],String,Atom_list):-
in_word(C,L),
New_string is_string String & [L],
list_to_atoms(List,New_string,Atom_list).


/* Form an atom from a string, replacing synonyms */

form_atom(Input,Atom):-
name(A,Input),
replace_synonym(A,Atom).

form_atom(Input,Input).            /* leave as string if necessary */


/* replace synonyms and abbreviations */

replace_synonym(Atom,New_atom):-
synonym(New_atom,Atom).

replace_synonym(Atom,Atom).


/* characters that can appear */

in_word(C,L):- C>64, C<91, L is C+32.      /* upper case */
in_word(C,C):- C>96, C<123.                /* lower case */
in_word(C,C):- C>46, C<58.                 /* numbers */
in_word(38,38).                            /* & */
in_word(39,39).                            /* ' */
in_word(45,45).                            /* - */
in_word(46,46).                            /* . */


/* main clauses to update the lexicon */

update_lexicon(add,Data_type,Data):-          /* fails if data invalid */
string_to_atoms(Data,Atoms),
expand_term((noun(Data,Data_type) --> Atoms),Lexicon_entry),
asserta(Lexicon_entry).

update_lexicon(delete,Data_type,Data):-
retractall(noun(Data,Data_type,A,B)).


/* Loading and storing the lexicon */

load_lexicon:-
file("LEXICON.DAT",_),
current_application(_,Index),
Filename is_string "\\lda\\" & Index & "\\lexicon.dat",
reconsult(Filename).

load_lexicon.

store_lexicon:-                    /* stores lexicon data from session */
(delete_file("LEXICON.TMP");true),
create("LEXICON.TMP"),
create_stream(data_file,readwrite,ascii,file("LEXICON.TMP")),
open(data_file,readwrite),
state(output,_,data_file),
write_clauses(noun/4),
close(data_file),
delete_stream(data_file),
(delete_file("LEXICON.DAT");true),
rename_file("LEXICON.TMP","LEXICON.DAT").


/* ---------------------------------------------------------------- */
/* ---------------- ROUTINES TO HANDLE KNOWLEDGE BASE ---------------- */

/* get the rank of predicate in knowledge base */

get_rank(relation,none,Relation_no,Rank):-
clause(relation/2,relation(Relation_no,_),R),
Rank is R-1,!.

get_rank(frame_relation,Frame_name,Relation_no,Rank):-
clause(frame_relation/4,frame_relation(Frame_name,_,Relation_no,_),R),
Rank is R-1,!.

get_rank(Frame_slot,Frame_name,Facet,Rank):-
Archive=..[Frame_slot,Frame_name,_,Facet,_,_],
clause(Frame_slot/5,Archive,Rank),!.

get_rank(Frame_slot,Frame_name,Facet,0).


read_kbase:-                          /* read frame data for session */
file("KBASE.DAT",_),
current_application(_,Index),
Filename is_string "\\lda\\" & Index & "\\kbase.dat",
reconsult(Filename).

read_kbase.


store_kbase:-                         /* stores frame data from session */
(delete_file("KBASE.TMP");true),
create("KBASE.TMP"),
create_stream(data_file,readwrite,ascii,file("KBASE.TMP")),
open(data_file,readwrite),
state(output,_,data_file),
write_clauses(disorder/1),
write_clauses(disease/1),
write_clauses(disorder/3),
write_clauses(disease/3),
write_clauses(link/4),
write_clauses(variable/1),
write_clauses(data_parameter/3),
write_clauses(symptom/2),
write_clauses(relation/2),
write_clauses(history/2),
write_clauses(frame_variable/5),
write_clauses(frame_symptom/5),
write_clauses(frame_relation/4),
write_clauses(frame_history/5),
close(data_file),
delete_stream(data_file),
(delete_file("KBASE.DAT");true),
rename_file("KBASE.TMP","KBASE.DAT").


/* ---------------------------------------------------------------- */
/* ---------------- ROUTINES TO PRINT KNOWLEDGE BASE ---------------- */

/* main routine to print database */

print_database(ok):-
create_stream(kbout,write,ascii,file("\\lda\\kbase.out")),
open(kbout,write),
state(output,Old,kbout),
print_variables,
print_relations,
print_symptoms,
print_history,
print_disorder_frames,
print_disease_frames,
close(kbout),
state(output,_,Old).

print_database(Result).


/* some utilities */

print_line(A,B,C,D):-
Line is_string string(A,ops) & " " & string(B,ops) & " " & string(C,ops)
& " " & string(D,ops),
write(Line),
write("\r\n").


print_indented_list([]):-!.

print_indented_list([Element|List]):-
write("  "),
write(Element),
write("\r\n"),
print_indented_list(List).


print_continuous_list([]):-!.

print_continuous_list([Element|List]):-
write(Element),
print_continuous_list(List).


/* print details of data variables */


print_variables:-
not variable(V).

print_variables:-
write("DATA VARIABLES \r\n\n"),
variable(V),
data_parameter(V,units,Units),
print_line(V," ",Units,""),
data_parameter(V,upper_limit,U),
print_line("","upper limit"," ",U),
data_parameter(V,lower_limit,L),
print_line("","lower limit"," ",L),
data_parameter(V,mean,M),
print_line("","mean value "," ",M),
data_parameter(V,standard_deviation,S),
print_line("","std dev    "," ",S),
data_parameter(V,default,D),
print_line("","default    "," ",D),
write("\n"),
```

```
fail.

print_variables:-
write("\r\n\n").


/* Print signs and symptoms */

print_symptoms:-
not symptom(S,A).

print_symptoms:-
write("SIGNS & SYMPTOMS \r\n\n"),
symptom(S,["unknown"|A]),
write(S),write("\r\n"),
print_indented_list(A),
write("\r\n"),
fail.

print_symptoms:-
write("\r\n\n").


/* Print history */

print_history:-
not history(H,A).

print_history:-
write("PATIENT HISTORY \r\n\n"),
history(H,["unknown"|A]),
write(H),write("\r\n"),
print_indented_list(A),
write("\r\n"),
fail.

print_history:-
write("\r\n\n").


/* Print relationships for data derivation */

print_relations:-
not relation(N,R).

print_relations:-
write("RELATIONSHIPS FOR DATA DERIVATION \r\n\n"),
relation(N,R),
write(N),write(". "),
print_continuous_list(R),
write("\r\n\n"),
fail.

print_relations:-
write("\r\n\n").


/* General routines for printing frames */

/* print frames for a tree */

print_frame_class(Root,[]):-!.

print_frame_class(Root,[Root|List]):-
print_line("Frame for",Root,"",""),
print_frame_data(Root,Root),
print_frame_class(Root,List).

print_frame_class(Root,[Frame|List]):-
link(Root,Parent,Frame,W),
Parent_display is_string "  Type of " & Parent,
Weight_display is_string " (" & string(W) & ")",
print_line("Frame for",Frame,Parent_display,Weight_display),
print_frame_data(Root,Frame),
print_frame_class(Root,List).


/* Print data for a frame */

print_frame_data(Root,Frame):-
frame_object(Object,Type),
Frame_data=..[Object,Frame,Root,Facet,Attribute,P],
call(Frame_data),
write(" "),
print_line(Type,Facet,Attribute,P),
fail.

print_frame_data(Root,Frame):-
frame_relation(Frame,Root,N,Relation),
write(" relation "),
write(N),write(". "),
print_continuous_list(Relation),
fail.

print_frame_data(Root,Frame):-
write("\r\n\n").



/* Print data for frames of disorders */

print_disorder_frames:-
not disorder(Root).

print_disorder_frames:-
write("FRAMES FOR DISORDERS \r\n\n"),
disorder(Root),
print_line("CLASS:",Root,"\r","\n"),
get_tree_nodes(disorder,Root,Nodes),
print_frame_class(Root,Nodes),
write("\r\n"),
fail.

print_disorder_frames:-
write("\r\n\n").


/* Print data for frames of diseases */

print_disease_frames:-
not disease(Root).

print_disease_frames:-
write("FRAMES FOR DISEASES \r\n\n"),
disease(Root),
print_line("CLASS:",Root,"\r","\n"),
get_tree_nodes(disease,Root,Nodes),
print_frame_class(Root,Nodes),
write("\r\n"),
fail.

print_disease_frames.
```

```
/* build.gen 1st level of FRAMEBUILDER */

/* Predicates defined in this file:

object/4
predicate_arity/2
highlight_action/1
define_main_command_line/0
draw_main_command_line/0
main_command_loop/0
main_action/2
sub_command_attribute/3
define_sub_command_line/0
sub_command_line/4
sub_command_loop/2
frame_sub_command_loop/1
relation_sub_command_loop/1
sub_command/3
drop_window_width/2
drop_window/2
define_drop_window/1
drop_window_display/2
draw_drop_window/3
enter_frame/4
add_frame/2
delete_frame/2
frame_selection_command/2
set_protected_slots/0
set_age_slot/0
set_sex_slot/0
set_occupation_slot/0
set_disorder_slot/0
set_previous_disorder_slot/0
set_clinical_diagnosis_slot/0
default_attribute/3
enter_facet/3
add_facet/2
delete_facet/2
set_attribute/2
slot_command/2
enter_attribute/2
add_attribute/3
delete_attribute/3
define_attribute_window/1
draw_attribute_window/2
display_attributes/2
attribute_command_loop/2
attribute_action/3
parameter/2
check_data/2
define_data_parameter_window/0
draw_data_parameter_window/1
data_parameter_command_loop/1
data_parameter_action/3          */

/* ------------------------------------------------------------------*/
/* MAIN COMMAND LINE */

object(disorder,"Disorder",1,2).
object(disease,"Disease",12,13).
object(variable,"Variables",28,29).
object(symptom,"Signs/Symptoms",39,40).
object(relation,"Relations",55,25).
object(history,"History",66,43).

object(frame_variable,"DATA VARIABLE            LOW   USUAL
HIGH",3,29). object(frame_symptom,"SIGNS/SYMPTOMS          ATTRIBUTE
P(a)",55,7). object(frame_history,"HISTORY                 ATTRIBUTE
P(h)",1,27).


predicate_arity(disorder,1).
predicate_arity(disease,1).
predicate_arity(variable,1).
predicate_arity(symptom,2).
predicate_arity(relation,2).
predicate_arity(history,2).
predicate_arity(frame_variable,5).
predicate_arity(frame_symptom,5).
predicate_arity(frame_relation,4).
predicate_arity(frame_hisory,5).


highlight_action(Object):-
object(Object,Title,Position,Wp),
window(main,cursor_address(0,Position)),
window(main,attribute(cyan on red)),
window(main,text(Title)).


define_main_command_line:-
create_stream(main,readwrite,byte,window(1,80,red on cyan)),
open(main,readwrite),
screen(main,create(0,0,main,0,0,0,none,None,1,80,hidden)),
retractall(builder_state/3),
assert(builder_state(main_command,none,none)),
draw_main_command_line,
retractall(cursor_location(main,SY,SX)),
assert((cursor_location(main,0,0))).


draw_main_command_line:-
window(main,cursor_address(0,0)),
window(main,attribute(red on cyan)),
window(main,text(" Disorder   Disease          Variables  Signs/Symptoms
Relations  History  EXIT ")), screen(main,pull_up),
screen(main,unhide).


main_command_loop:-
repeat,
once(cursor_location(main,SY,SX)),
once(locator(SY,SX,Y,X,0,0,0,79)),
once(retract(cursor_location(main,SY,SX))),
once(assert(cursor_location(main,Y,X))),
once(main_action(X,Result)),
Result=exit,
close(main),
delete_stream(main).


main_action(X,exit):-                                  /* exit clicked */
X>74,
X<79,
decision_box(3,40,"Saving changes....",Result),
store_data(Result),
decision_box(3,40,"Print database....",Result2),
print_database(Result2).


main_action(X,continue):-                        /* disorder selection */
X>0,
X<10,
retract(builder_state(main_command,none,none)),
assert(builder_state(frame_selection_command,disorder,none)),
drop_window(short,disorder).


main_action(X,continue):-                        /* disease selection */
X>11,
X<19,
retract(builder_state(main_command,none,none)),
assert(builder_state(frame_selection_command,disease,none)),
drop_window(short,disease).

main_action(X,continue):-                            /* laboratory data */
X>27,
X<37,
retract(builder_state(main_command,none,none)),
assert(builder_state(slot_command,variable,none)),
drop_window(short,variable).

main_action(X,continue):-                      /* sign/symptom selection */
X>38,
X<53,
retract(builder_state(main_command,none,none)),
assert(builder_state(slot_command,symptom,none)),
drop_window(short,symptom).

main_action(X,continue):-                              /* set relations */
X>54,
X<64,
relation_drop_window(relation).

main_action(X,continue):-                          /* history selection */
X>65,
X<73,
retract(builder_state(main_command,none,none)),
assert(builder_state(slot_command,history,none)),
drop_window(short,history).

main_action(X,continue).

/* ------------------------------------------------------------------*/
/* SUB COMMAND LINE */

sub_command_attribute(tiny,23,"\24  Add  Delete  Exit  \25").
sub_command_attribute(short,29,"\24  Add  Delete  Edit  Exit  \25").
sub_command_attribute(long,45,"\24       Add     Delete     Edit     Exit
\25").


define_sub_command_line:-              /* if stream is already defined */
stream(sub_command,_,_,_,_).

define_sub_command_line:-
create_stream(sub_command,readwrite,byte,window(1,45,white on blue)),
open(sub_command,readwrite),
screen(sub_command,create(0,0,sub_command,0,0,0,none,None,1,45,hidden)).


sub_command_line(Window,Ypos,Xpos,Return):-
sub_command_attribute(Window,Width,Text),
screen(sub_command,change(Ypos,Xpos,sub_command,0,0,0,none,None,1,Width,
hidden)), window(sub_command,cursor_address(0,0)),
window(sub_command,text(Text)),
screen(sub_command,unhide),
screen(sub_command,pull_up),
Xend is Xpos+Width-1,
cursor_location(sub_command(Window),_,SX),
locator(Ypos,SX,Y,X,Ypos,Xpos,Ypos,Xend),
retract(cursor_location(sub_command(Window),_,_)),
assert(cursor_location(sub_command(Window),Y,X)),
Key is X-Xpos,
sub_command(Window,Key,Return).


sub_command_loop(Window,Position):-
builder_state(State,_,_),
retractall(cursor_location(sub_command(Window),_,_)),
assert(cursor_location(sub_command(Window),0,Position)), repeat,
once(window_drop(Window,Drop)),
once(CY is Drop+1),
once(sub_command_line(Window,CY,Position,Result)),
once(Action=..[State,Result,Drop]),
once(call(Action)),
Result=exit,
close(sub_command),
delete_stream(sub_command),
close(Window),
delete_stream(Window).

frame_sub_command_loop(Frame_slot):-
object(Frame_slot,Title,Pos,Wpos),
retractall(cursor_location(sub_command(long),_,_)),
assert(cursor_location(sub_command(long),0,Wpos)),
repeat,
once(window_drop(long,Drop)),
once(CY is Drop+1),
once(sub_command_line(long,CY,Wpos,Result)),
once(frame_slot_command(Result,Frame_slot)),
Result=exit,
close(sub_command),
delete_stream(sub_command),
```

```
close(long),
delete_stream(long).

relation_sub_command_loop(Frame_slot):-
retractall(cursor_location(sub_command(long),_,_)),
assert(cursor_location(sub_command(long),0,25)),
repeat,
once(window_drop(long,Drop)),
once(CY is Drop+1),
once(sub_command_line(long,CY,25,Result)),
once(relation_command(Result,Frame_slot)),
Result=exit,
close(sub_command),
delete_stream(sub_command),
close(long),
delete_stream(long).



sub_command(Window,0,scroll_up).

sub_command(long,44,scroll_down).

sub_command(long,Key,add):-
Key>7,
Key<11.

sub_command(long,Key,delete):-
Key>14,
Key<21.

sub_command(long,Key,edit):-
Key>24,
Key<29.

sub_command(long,Key,exit):-
Key>32,
Key<37.


sub_command(short,28,scroll_down).

sub_command(short,Key,add):-
Key>2,
Key<6.

sub_command(short,Key,delete):-
Key>7,
Key<14.

sub_command(short,Key,edit):-
Key>15,
Key<20.

sub_command(short,Key,exit):-
Key>21,
Key<26.


sub_command(tiny,22,scroll_down).

sub_command(tiny,Key,add):-
Key>2,
Key<6.

sub_command(tiny,Key,delete):-
Key>7,
Key<14.

sub_command(tiny,Key,exit):-
Key>15,
Key<20.

sub_command(Window,Key,continue).


/* ----------------------------------------------------------------*/
/* DROP DOWN MENUS */

drop_window_width(long,45).
drop_window_width(short,29).
drop_window_width(tiny,23).


drop_window(Window,Object):-
highlight_action(Object),
object(Object,Title,Position,Wposition),
define_drop_window(Window),
draw_drop_window(Window,Object,Wposition),
define_sub_command_line,
sub_command_loop(Window,Wposition),
draw_main_command_line.


define_drop_window(Window):-
drop_window_width(Window,Width),
create_stream(Window,readwrite,byte,window(60,Width,white on black)),
open(Window,readwrite),
screen(Window,create(1,1,Window,0,0,0,all,blue on
black,23,Width,hidden)).


drop_window_display(Window,Object):-
once(predicate_arity(Object,A)),
once(length(Argument_list,A)),
once(Archive=..[Object|Argument_list]),
call(Archive),
once(Argument_list=[Entry|Other_arguments]),
once(write_window_entry(Window,Entry)),
fail.

drop_window_display(Window,Object).

draw_drop_window(Window,Object,Position):-
predicate_arity(Object,A),
number_of_clauses(Object/A,_,N),
((N<22,Drop is N+1);(Drop is 22)),
retractall(window_drop(Window,_)),
assert(window_drop(Window,Drop)),
drop_window_width(Window,W),
screen(Window,change(2,Position,Window,0,0,0,all,blue on
black,Drop,W,hidden)),
window(Window,clear),
window(Window,cursor_address(0,1)),
drop_window_display(Window,Object),
scroll_to_foot(Window),
screen(Window,unhide).


/* ----------------------------------------------------------------*/
/* FRAME SELECTION */

enter_frame(Action,Frame_type,Drop,Frame_name):-
object(Frame_type,Title,_,Xposition),
((Drop<21,Edit_Yposition is Drop+3);(Edit_Yposition is 23)),
Edit_Xposition is Xposition+32,
Instruction is_string Action & Title,
fedit(Edit_Yposition,Edit_Xposition,24,Instruction," ","",blue on
white,Entry), truncate_string(Entry,Frame_name).


add_frame(Frame_type,"").

add_frame(Frame_type,Frame_name):-
disorder(Frame_name).

add_frame(Frame_type,Frame_name):-
disease(Frame_name).

add_frame(Frame_type,Frame_name):-
update_lexicon(add,Frame_type,Frame_name),
Archive=..[Frame_type,Frame_name],
assert(Archive).


delete_frame(Frame_type,Frame_name):-
Archive=..[Frame_type,Frame_name],                    /* remove the root */
retractall(Archive),
retractall(link(Frame_name,_,_,_)),
update_lexicon(delete,Frame_type,Frame_name),
Tree_archive=..[Frame_type,Frame_name,Node,_],        /* remove the tree */
retract(Tree_archive),
update_lexicon(delete,Frame_type,Node),
fail.

delete_frame(Frame_type,Frame_name):-          /* remove the frame data */
frame_object(Slot,_),
Archive=..[Slot,Node,Frame_name,_,_,_],
retractall(Archive),
fail.

delete_frame(Frame_type,Frame_name).


frame_selection_command(add,Drop):-
builder_state(_,Frame_type,_),
enter_frame("Add ",Frame_type,Drop,Frame_name),
add_frame(Frame_type,Frame_name),
object(Frame_type,Title,Position,Wposition),
draw_drop_window(short,Frame_type,Wposition).


frame_selection_command(delete,Drop):-
builder_state(_,Frame_type,_),
enter_frame("Delete ",Frame_type,Drop,Frame_name),
delete_frame(Frame_type,Frame_name),
object(Frame_type,Title,Position,Wposition),
draw_drop_window(short,Frame_type,Wposition).


frame_selection_command(edit,Drop):-
Drop>0,
builder_state(_,Frame_type,_),
object(Frame_type,Title,_,Xposition),
get_window_entry(short,Xposition,Frame_name),
retractall(builder_state/3),
assert(builder_state(Frame_name,Frame_type,Frame_name)), run_tree,
draw_main_command_line,                    /* gets overwritten in frames */
define_sub_command_line,                   /* gets closed in frame edits */
highlight_action(Frame_type),
retractall(builder_state/3),
assert(builder_state(frame_selection_command,Frame_type,none)).


frame_selection_command(scroll_up,Drop):-
scroll_window_up(short).

frame_selection_command(scroll_down,Drop):-
scroll_window_down(short).

frame_selection_command(exit,Drop):-
retractall(builder_state/3),
assert(builder_state(main_command,none,none)).


/* ----------------------------------------------------------------*/
/* DEFINITION OF PROTECTED SLOTS */

set_protected_slots:-
set_age_slot,
set_sex_slot,
set_occupation_slot,
set_disorder_slot,
set_previous_disorder_slot,
set_clinical_diagnosis_slot.

set_age_slot:-                                  /* always replace age */
retractall(variable("age")),
retractall(data_parameter("age",_,_)),
update_lexicon(delete,variable,"age"),
assert(variable("age")),
assert(data_parameter("age",units,"years")),
assert(data_parameter("age",upper_limit,120)),
assert(data_parameter("age",lower_limit,0)),
```

```
assert(data_parameter("age",mean,none)),
assert(data_parameter("age",standard_deviation,none)),
assert(data_parameter("age",default,none)).


set_sex_slot:-
history("sex",_).

set_sex_slot:-
add_facet(history,"sex"),
add_attribute(history,"sex","male"),
add_attribute(history,"sex","female").


set_occupation_slot:-
history("occupation",_).

set_occupation_slot:-
add_facet(history,"occupation").

set_disorder_slot:-
history("disorder",_).

set_disorder_slot:-
add_facet(history,"disorder").

set_previous_disorder_slot:-
history("previous disorder",_).

set_previous_disorder_slot:-
add_facet(history,"previous disorder").

set_clinical_diagnosis_slot:-
history("clinical diagnosis",_).

set_clinical_diagnosis_slot:-
add_facet(history,"clinical diagnosis").


/* default attributes for disorder histories */

/* previous disorders have same root as present disorder */
default_attribute(Root,Facet,["unknown","was present","was
absent","unknown"]):- link(Root,_,Facet,_).

default_attribute(Root,Facet,["unknown","present","absent","unknown"]).

/* ----------------------------------------------------------------
*/ /* FACET DEFINITION ie Signs/symptoms and History */

enter_facet(Action,Drop,Facet):-
builder_state(_,Slot,_),
object(Slot,Title,_,Xposition),
((Drop<21,Edit_Yposition is Drop+3);(Edit_Yposition is 23)),
Edit_Xposition is Xposition-25,
Instruction is_string Action & Title,
fedit(Edit_Yposition,Edit_Xposition,24,Instruction," ","",blue on
white,Entry), truncate_string(Entry,Facet).


add_facet(Slot,""):-warning_box(8,30,"Cannot add nothing","","").

add_facet(variable,Data):-            /* Data variable already defined */
variable(Data),
warning_box(8,30,"Cannot add the variable",Data,"").

add_facet(variable,Data):-
update_lexicon(add,variable,Data),
assert(variable(Data)),
assert(data_parameter(Data,units,"")),
assert(data_parameter(Data,upper_limit,none)),
assert(data_parameter(Data,lower_limit,none)),
assert(data_parameter(Data,mean,none)),
assert(data_parameter(Data,standard_deviation,none)),
assert(data_parameter(Data,default,none)).

add_facet(Slot,Facet):-                       /* slot already exists */
object(Any_slot,_,_,_),
Archive=..[Any_slot,Facet,Attribute_list],
call(Archive),
warning_box(8,30,"Cannot add the facet...",Facet,"").

add_facet(Slot,Facet):-
update_lexicon(add,Slot,Facet),
update_lexicon(add,attribute(Slot,Facet),"unknown"),
Archive=..[Slot,Facet,["unknown","unknown"]],
assert(Archive).

/* delete slot facets */

delete_facet(variable,"age"):-warning_box(8,30,"Age cannot be
deleted","","").

delete_facet(history,Facet):-
member(Facet,["occupation","sex","disorder","previous
disorder","clinical diagnosis"]),
warning_box(8,30,Facet,"cannot be deleted","").

delete_facet(variable,Data):-                 /* variable slots */
retractall(variable(Data)),
retractall(data_parameter(Data,_,_)),
retractall(frame_variable(Frame_name,Root,Data,Level,Certainty)),
update_lexicon(delete,variable,Data).

delete_facet(Slot,Facet):-                    /* other slots */
Archive=..[Slot,Facet,["unknown"|Attribute_list]],
retract(Archive),
frame_object(Frame_slot,Slot),
Frame_archive=..[Frame_slot,Frame_name,Root,Facet,_,_],
retractall(Frame_archive),
update_lexicon(delete,Slot,Facet),
member(Attribute,Attribute_list),
update_lexicon(delete,attribute(Slot,Facet),Attribute),
Attribute=="unknown".


/* set slot attributes */

set_attribute(variable,"age"):-
warning_box(8,30,"The attributes for age","are fixed","").

set_attribute(history,"sex"):-
warning_box(8,30,"The attributes for sex","are fixed as male
or","female").

set_attribute(history,"disorder"):-
warning_box(8,30,"Disorder attributes","are set in the
disorder","trees").

set_attribute(history,"previous disorder"):-
warning_box(8,30,"Disorder attributes","are set in the
disorder","trees").

set_attribute(history,"clinical diagnosis"):-
warning_box(8,30,"Disease attributes","are set in the disease","trees").


set_attribute(variable,Data):-
define_data_parameter_window,
draw_data_parameter_window(Data),
data_parameter_command_loop(Data),
close(data_parameter),
delete_stream(data_parameter).


set_attribute(Slot,Facet):-
define_attribute_window(Slot),
draw_attribute_window(Slot,Facet),
attribute_command_loop(Slot,Facet),
close(attribute_header),
delete_stream(attribute_header),
close(attributes),
delete_stream(attributes).


slot_command(add,Drop):-
builder_state(_,Slot,_),
enter_facet("Add ",Drop,Facet),
add_facet(Slot,Facet),
object(Slot,Title,Position,Wposition),
draw_drop_window(short,Slot,Wposition).


slot_command(delete,Drop):-
builder_state(_,Slot,_),
enter_facet("Delete ",Drop,Facet),
delete_facet(Slot,Facet),
object(Slot,Title,Position,Wposition),
draw_drop_window(short,Slot,Wposition).


slot_command(edit,Drop):-
builder_state(_,Slot,_),
object(Slot,Title,_,Xposition),
get_window_entry(short,Xposition,Facet),
set_attribute(Slot,Facet).

slot_command(scroll_up,Drop):-
scroll_window_up(short).

slot_command(scroll_down,Drop):-
scroll_window_down(short).

slot_command(exit,Drop):-
retractall(builder_state/3),
assert(builder_state(main_command,none,none)).


/* ----------------------------------------------------------------*/
/* SLOT ATTRIBUTE DEFINITION */


enter_attribute(Action,Attribute):-
builder_state(_,Slot,_),
object(Slot,Title,_,Xposition),
window_drop(attributes,Drop),
((Drop<22,Edit_Yposition is Drop+3);(Edit_Yposition is 23)),
Edit_Xposition is Xposition-39,
Instruction is_string Action & Title,
fedit(Edit_Yposition,Edit_Xposition,17,Instruction," ","",blue on
white,Entry), truncate_string(Entry,Attribute).


add_attribute(Slot,Facet,""):-
warning_box(8,30,"Cannot add nothing","","").

add_attribute(Slot,Facet,Attribute):-
Archive=..[Slot,Facet,["unknown"|Attribute_list]],
call(Archive),
member(Attribute,Attribute_list),
warning_box(8,30,Attribute,"already exists","").

add_attribute(Slot,Facet,Attribute):-
update_lexicon(add,attribute(Slot,Facet),Attribute),
Archive=..[Slot,Facet,["unknown"|Attribute_list]],
call(Archive),
New_attribute_list=[Attribute|Attribute_list],
retractall(Archive),
New_archive=..[Slot,Facet,["unknown"|New_attribute_list]],
assert(New_archive).


delete_attribute(Slot,Facet,"unknown"):-
warning_box(8,30,Attribute,"This is the default","It cannot be
deleted","").

delete_attribute(Slot,Facet,Attribute):-
Archive=..[Slot,Facet,Attribute_list],
call(Archive),
delete(Attribute,Attribute_list,New_attribute_list),
retractall(Archive),
New_archive=..[Slot,Facet,New_attribute_list],
assert(New_archive),
```

```
update_lexicon(delete,attribute(Slot,Facet),Attribute).

delete_attribute(Slot,Facet,Attribute).


define_attribute_window(Slot):-
object(Slot,Title,Position,Wposition),
Xpos is Wposition-26,
create_stream(attributes,readwrite,byte,window(60,23,red on cyan)),
open(attributes,readwrite),
screen(attributes,create(3,Xpos,attributes,0,0,0,none,None,20,23,hidden)
), create_stream(attribute_header,readwrite,byte,window(1,23,white on
blue)), open(attribute_header,readwrite),
screen(attribute_header,create(2,Xpos,attribute_header,0,0,0,none,None,1
,23,hidden)).


draw_attribute_window(Slot,Facet):-
once(Archive=..[Slot,Facet,Attribute_list]),
once(call(Archive)),
object(Slot,Title,Position,Wposition),
Xpos is Wposition-26,
length(Attribute_list,N),
((N<20,Drop is N);(Drop is 20)),
retractall(window_drop(attributes,_)),
assert(window_drop(attributes,Drop)),
screen(attributes,change(3,Xpos,attributes,0,0,0,none,None,Drop,23,revea
led)),
window(attribute_header,cursor_address(0,1)),
window(attribute_header,text(Facet)),
screen(attribute_header,unhide),
window(attributes,clear),
window(attributes,cursor_address(0,1)),
display_attributes(Slot,Facet),
screen(attributes,unhide).


display_attributes(Slot,Facet):-
once(Archive=..[Slot,Facet,["unknown"|Attribute_list]]),
once(call(Archive)),
member(Attribute,Attribute_list),            /* display each attribute */
once(write_window_entry(attributes,Attribute)),
fail.

display_attributes(Slot,Facet):-
window(attributes,cursor_home),
scroll_to_foot(attributes).


attribute_command_loop(Slot,Facet):-
object(Slot,Title,Position,Wposition),
Xpos is Wposition-26,
retractall(cursor_location(sub_command(tiny),_,_)),
assert(cursor_location(sub_command(tiny),0,Xpos)),
repeat,
once(window_drop(attributes,Drop)),
once(CY is Drop+2),
once(sub_command_line(tiny,CY,Xpos,Result)),
once(attribute_action(Result,Slot,Facet)),
Result=exit.


attribute_action(exit,Slot,Facet).

attribute_action(add,Slot,Facet):-
enter_attribute("Add ",Attribute),!,
add_attribute(Slot,Facet,Attribute),
draw_attribute_window(Slot,Facet).


attribute_action(delete,Slot,Facet):-
enter_attribute("Delete ",Attribute),
delete_attribute(Slot,Facet,Attribute),
draw_attribute_window(Slot,Facet).

attribute_action(scroll_up,Slot,Facet):-
scroll_window_up(attributes).

attribute_action(scroll_down,Slot,Facet):-
scroll_window_down(attributes).


/* ------------------------------------------------------------------*/
/* DATA PARAMETER DEFINITION */

parameter(units,"Units       ").
parameter(upper_limit,"Upper Limit ").
parameter(lower_limit,"Lower Limit ").
parameter(mean,"Mean        ").
parameter(standard_deviation,"S.D.        ").
parameter(default,"Default     ").


/* check values entered as data parameter attributes */

check_data(standard_deviation,Value):-
Value=<0,!,
warning_box(8,30,"standard deviation","must be > zero",""), fail.

check_data(Parameter,Value).


define_data_parameter_window:-
create_stream(data_parameter,readwrite,byte,window(8,22,red on cyan)),
open(data_parameter,readwrite),
screen(data_parameter,create(0,0,data_parameter,0,0,0,none,None,8,22,hid
den)).


draw_data_parameter_window(Data):-
window(data_parameter,attribute(white on blue)),
fill_out(Data,Data_name,22),
window(data_parameter,cursor_address(0,0)),
window(data_parameter,text(Data_name)),
window(data_parameter,attribute(red on cyan)),
data_parameter(Data,units,Units),
fill_out(Units,U,7),
Units_display is_string "Units       " & U,
window(data_parameter,cursor_address(1,1)),
window(data_parameter,text(Units_display)),
member(Parameter,[upper_limit,lower_limit,mean,standard_deviation,defaul
t]), once(data_parameter(Data,Parameter,Value)),
once(parameter(Parameter,Parameter_display)),
once(value_display(Value,Value_display)),
once(Display is_string "\r\n " & Parameter_display & Value_display),
once(window(data_parameter,text(Display))),
Parameter=default,
window(data_parameter,text("\r\n EXIT")),
screen(data_parameter,change(3,5,data_parameter,0,0,0,none,None,8,22,rev
ealed)).


data_parameter_command_loop(Data):-
retractall(cursor_location(dp,_,_)),
assert(cursor_location(dp,9,6)),
repeat,
once(cursor_location(dp,SY,SX)),
once(locator(SY,5,Y,X,4,6,10,26)),
once(EY is Y-3),                                  /* EY Y pos in window */
once(window(data_parameter,cursor_address(EY,1))),
once(window(data_parameter,inquire_text(12,Action))),
once(data_parameter_action(Data,Action,Y)),
once(retract(cursor_location(dp,SY,SX))),
once(assert(cursor_location(dp,Y,X))),
Action="EXIT        ".


data_parameter_action(Data,"Units       ",Y):-
data_parameter(Data,units,Units),
enter_text(Y,19,New_units),
retractall(data_parameter(Data,units,Units)),
assert(data_parameter(Data,units,New_units)),
draw_data_parameter_window(Data).

data_parameter_action(Data,Parameter_display,Y):-
parameter(Parameter,Parameter_display),
enter_data(Y,19,Value),!,
check_data(Parameter,Value),
retractall(data_parameter(Data,Parameter,_)),
assert(data_parameter(Data,Parameter,Value)),
draw_data_parameter_window(Data).

data_parameter_action(Data,Parameter_display,Y):-
parameter(Parameter,Parameter_display),
retractall(data_parameter(Data,Parameter,_)),
assert(data_parameter(Data,Parameter,none)),
draw_data_parameter_window(Data).


data_parameter_action(Data,Action,Y).
```

```
/* buildtre 2nd level of FRAMEBUILDER */

/* Predicates defined in this file:

sub_category/2
find_descendent/2
entry_exists/1
run_tree/0
draw_horizontal_scale/0
draw_vertical_scale/0
draw_instructions/0
create_base/0
remove_base/0
display_windows/0
scroll_left/0
scroll_right/0
scroll_up/0
scroll_down/0
set_scrolls/0
follows/2
create_window/1
set_tree_windows/1
remove_tree_windows/1
load_tree_data/0
save_tree_data/0
highlight/1
set_action/1
get_entry/3
input_Yoffset/2
input_Xoffset/2
input_data/3
check_input/1
command_loop/0
base_action/3
perform_action/3
edit_entry/2
edit_entry_list/2
edit_frame_data/2
delete_tree_entry/1
delete_frame_data/1
delete_descendents/2
add_tree_entry/2
check_link/3
link_action/2
define_weight_window/0
remove_weight_window/0
draw_weight_window/3
draw_weight_entries/3
draw_weight_line/3
weight_command_loop/3
weight_action/5
update_weight_window/4
get_weight_window_entry/2
update_weight/3
check_weights/2
distribute_residue_weights/3
create_format/1
create_tree_format/0
draw_tree/0
draw_generation/1
entry_display/2
write_entry/2
draw_generation_entries/2
draw_brackets/2
bracket_display/2
bracket/2
make_connections/2
create_origin_list/4
add_origins/3
reformat_list/4
generation_size/3
centre_generation/3
get_offset/3
space_out_format/4
create_destination_list/3
draw_connections/3
set_direction/2
set_depth/1
connect/3
horizontal_connection/2
upward_connection/2
downward_connection/2  */


/* ------------------------------------------------------------------ */
/* ------------------------ SOME EXTRA UTILITIES -------------------- */

/* Check if Entry is in the subset of Parent (tree version) */

sub_category(Parent,Entry):-
tree(Parent,Entry_list),
member(Entry,Entry_list).


/* To find a descendent of a node (tree version) */

find_descendent(Node,Descendent):-
sub_category(Node,Descendent).

find_descendent(Node,Descendent):-
sub_category(Node,Child),
find_descendent(Child,Descendent).


/* Check to see if a node exists */

entry_exists(Entry):-              /* disorder root */
disorder(Entry).

entry_exists(Entry):-              /* disease root */
disease(Entry).

entry_exists(Entry):-              /* tree entry */
link(_,_,Entry,_).


/* ------------------------------------------------------------------ */
/* ------------------------ MAIN FUNCTIONS -------------------------- */


/* To run the tree display, set the environment, load the tree, display
it,run round the command loop until exit and then reset the database and
environment */


run_tree:-
create_base,
set_scrolls,
set_tree_windows(root),
load_tree_data,
create_tree_format,
draw_tree,
command_loop,
remove_tree_windows(root),
remove_base.


/* SETTING THE ENVIRONMENT */

/* Set background */

draw_horizontal_scale:-
window(base,attribute(yellow on red)),
window(base,cursor_address(0,0)),
window(base,text("  \17\17")),
window(base,cursor_address(0,6)),
repeat(X),
window(base,text("\176")),
X=19,
window(base,cursor_address(0,28)),
window(base,text("\16\16")).

draw_vertical_scale:-
window(base,attribute(yellow on red)),
window(base,cursor_address(1,0)),
window(base,text("\30\30")),
repeat(R),
Y is R+3,
window(base,cursor_address(Y,0)),
window(base,text("\176\176")),
R=19,
window(base,cursor_address(24,0)),
window(base,text("\31\31")).


draw_instructions:-
window(base,attribute(red on cyan)),
window(base,cursor_address(0,41)),
window(base,text(" VIEW ")),
window(base,cursor_address(0,48)),
window(base,text(" ADD ")),
window(base,cursor_address(0,54)),
window(base,text(" EDIT ")),
window(base,cursor_address(0,61)),
window(base,text(" DEL ")),
window(base,cursor_address(0,67)),
window(base,text(" LINK ")),
window(base,cursor_address(0,74)),
window(base,text(" EXIT ")).


create_base:-
create_stream(base,readwrite,byte,window(25,80,black on black)),
open(base,readwrite),
screen(base,create(0,0,base,0,0,0,none,None,25,80,hidden)),
draw_horizontal_scale,
draw_vertical_scale,
draw_instructions,
set_action(view),
screen(base,unhide).

remove_base:-
close(base),
delete_stream(base).


/* Scrolling & Display Operations */

/* Displaying the tree windows */

display_windows:-
current_window(Window1),
follows(Window1,Window2),
follows(Window2,Window3),
vertical(V),
Ypos is (V-1)*4,
screen(Window1,change(3,4,Window1,Ypos,0,0,none,None,20,27,revealed)),
screen(Window2,change(3,31,Window2,Ypos,0,0,none,None,20,27,revealed)),
screen(Window3,change(3,58,Window3,Ypos,0,0,none,None,20,22,revealed)).


/* Scrolling the tree display */

scroll_left:-
horizontal(H),
H>1,
NH is H-1,
retractall(horizontal(H)),
assert(horizontal(NH)),
Xpos is (2*NH)+4,
window(base,attribute(yellow on red)),
window(base,cursor_address(0,Xpos)),
window(base,text("\219\219\176\176")),
retract(current_window(Window)),
follows(New_window,Window),
assert(current_window(New_window)),
display_windows.

scroll_left.

scroll_right:-
current_window(Window),
follows(Window,Second),
```

```
follows(Second,Third),
follows(Third,_),
horizontal(H),
NH is H+1,
retractall(horizontal(H)),
assert(horizontal(NH)),
Xpos is (2*H)+4,
window(base,attribute(yellow on red)),
window(base,cursor_address(0,Xpos)),
window(base,text("\176\176\219\219")),
retract(current_window(Window)),
follows(Window,New_window),
assert(current_window(New_window)),
display_windows.


scroll_right.


scroll_up:-
vertical(V),
V>1,
NV is V-1,
retractall(vertical(V)),
assert(vertical(NV)),
Ypos is NV+2,
window(base,attribute(yellow on red)),
window(base,cursor_address(Ypos,0)),
window(base,text("\219\219\n\r\176\176")),
display_windows.

scroll_up.

scroll_down:-
vertical(V),
V<20,
NV is V+1,
retractall(vertical(V)),
assert(vertical(NV)),
Ypos is V+2,
window(base,attribute(yellow on red)),
window(base,cursor_address(Ypos,0)),
window(base,text("\176\176\n\r\219\219")),
display_windows.

scroll_down.

set_scrolls:-
retractall(vertical(V)),
assert(vertical(10)),
window(base,attribute(yellow on yellow)),
window(base,cursor_address(12,0)),
window(base,text("  ")),
retractall(horizontal(H)),
assert(horizontal(1)),
window(base,cursor_address(0,6)),
window(base,text("  ")),
retractall(current_window(_)),
assert(current_window(root)).


/* SETTING UP THE WINDOWS */

/* define order of windows */

follows(root,first).
follows(first,second).
follows(second,third).
follows(third,fourth).
follows(fourth,fifth).
/* extra generations commented out to save space */
/* follows(fifth,sixth).*/
/* follows(sixth,seventh).*/
/* follows(seventh,eighth).  */
/* follows(eighth,nineth).*/


/* create window for tree column */

create_window(Window):-
create_stream(Window,readwrite,byte,window(96,27,red on black)),
open(Window,readwrite),
screen(Window,create(0,0,Window,0,0,0,none,None,22,27,hidden)),
window(Window,cursor_home).


/* set up tree display windows */

set_tree_windows(Window):-
once(create_window(Window)),
follows(Window,Next_window),
set_tree_windows(Next_window).

set_tree_windows(Window).


/* remove tree display windows */

remove_tree_windows(Window):-
close(Window),
delete_stream(Window),
follows(Window,Next_window),
remove_tree_windows(Next_window).

remove_tree_windows(Window).


/* Setting the current tree from the database */

load_tree_data:-
once(builder_state(Root,Type,_)),
once(assert(tree(root,[Root]))),
once(Archive=..[Type,Root,Entry,Entry_list]),
call(Archive),
assert(tree(Entry,Entry_list)),
fail.

load_tree_data.


/* Replacing tree data into general database */

save_tree_data:-
once(builder_state(Root,Type,_)),
once(Archive=..[Type,Root,_,_]),
once(retractall(Archive)),
once(retract(tree(root,_))),
tree(Entry,Entry_list),
once(New_archive=..[Type,Root,Entry,Entry_list]),
once(assert(New_archive)),
fail.

save_tree_data:-
retractall(tree/2).


/* ---------------------------------------------------------------- */
/* -------------------- BASE COMMANDS AND ACTIONS ------------------- */

/* A FEW DISPLAY AND INPUT ROUTINES */

/* Highlighting the command */

highlight(view):-
window(base,attribute(cyan on red)),
window(base,cursor_address(0,41)),
window(base,text(" VIEW ")).

highlight(add):-
window(base,attribute(cyan on red)),
window(base,cursor_address(0,48)),
window(base,text(" ADD ")).

highlight(edit):-
window(base,attribute(cyan on red)),
window(base,cursor_address(0,54)),
window(base,text(" EDIT ")).

highlight(delete):-
window(base,attribute(cyan on red)),
window(base,cursor_address(0,61)),
window(base,text(" DEL ")).

highlight(link):-
window(base,attribute(cyan on red)),
window(base,cursor_address(0,67)),
window(base,text(" LINK ")).


/* set the current command selection */

set_action(Action):-
retractall(current_action/1),
assert(current_action(Action)),
draw_instructions,
highlight(Action).


/* get the node under a mouse click */

get_entry(Window,Ypos,Entry):-
window(Window,cursor_address(Ypos,1)),
window(Window,inquire_text(20,Display)),
truncate_string(Display,Truncated_display),
delete_character(Truncated_display,Entry,"\196"),!,
Entry\="".


/* To input a node for add or edit routines */

input_Yoffset(Y,YO):-
Y<12,
YO is Y+2.

input_Yoffset(Y,YO):-
YO is Y-2.

input_Xoffset(X,3):-
X<31.

input_Xoffset(X,57):-
X>57.

input_Xoffset(X,30).

input_data(Instructions,Edit_string,Input):-
cursor_location(base,Y,X),
input_Yoffset(Y,YO),
input_Xoffset(X,XO),
fedit(YO,XO,22,Instructions," ",Edit_string,red on cyan,Entry),
truncate_string(Entry,Input),!,
check_input(Input).


check_input("":-!,fail.

check_input(Input):-
entry_exists(Input),
warning_box(8,30,Input,"already exists","",),!,fail.

check_input(Input).


/* BASIC TREE COMMANDS */

command_loop:-
retractall(cursor_location(base,_,_)),
assert(cursor_location(base,0,0)),
repeat,
once(cursor_location(base,SY,SX)),
once(locator(SY,SX,Y,X,0,0,24,79)),
once(retractall(cursor_location(base,SY,SX))),
once(assert(cursor_location(base,Y,X))),
once(base_action(Y,X,Result)),
```

```
Result=exit.


base_action(0,2,scroll_left):-
scroll_left.

base_action(0,3,scroll_left):-
scroll_left.

base_action(0,28,scroll_right):-
scroll_right.

base_action(0,29,scroll_right):-
scroll_right.


base_action(1,0,scroll_up):-
scroll_up.

base_action(1,1,scroll_up):-
scroll_up.

base_action(24,0,scroll_down):-
scroll_down.

base_action(24,1,scroll_down):-
scroll_down.

base_action(0,X,view):-
X>41,
X<48,
set_action(view).

base_action(0,X,add):-
X>48,
X<54,
set_action(add).

base_action(0,X,edit):-
X>54,
X<61,
set_action(edit).

base_action(0,X,delete):-
X>61,
X<67,
set_action(delete).

base_action(0,X,link):-
X>67,
X<74,
set_action(link).

base_action(0,X,exit):-
X>74,
save_tree_data.

base_action(Y,X,first_window):-
X>3,
X<31,
Y>2,
current_window(Window),
current_action(Action),
vertical(V),
Ypos is ((V-1)*4)+Y-3,
get_entry(Window,Ypos,Entry),
perform_action(Window,Entry,Action).

base_action(Y,X,second_window):-
X>31,
X<58,
Y>2,
current_window(Window),
follows(Window,Second_window),
current_action(Action),
vertical(V),
Ypos is ((V-1)*4)+Y-3,
get_entry(Second_window,Ypos,Entry),
perform_action(Second_window,Entry,Action).

base_action(Y,X,third_window):-
X>58,
Y>2,
current_window(Window),
follows(Window,Second_window),
follows(Second_window,Third_window),
current_action(Action),
vertical(V),
Ypos is ((V-1)*4)+Y-3,
get_entry(Third_window,Ypos,Entry),
perform_action(Third_window,Entry,Action).

base_action(Y,X,continue).


/* PERFORM TREE ACTIONS AFTER MOUSE CLICK */

perform_action(Window,Entry,view):-
retract(builder_state(Root,Frame_type,_)),
assert(builder_state(Root,Frame_type,Entry)),
define_frame_command_line,
frame_command_loop.

perform_action(root,Entry,add):-
warning_box(8,30,"Root of tree","Can only LINK or VIEW","").

perform_action(Window,Entry,add):-
input_data("Input new entry","",New_entry),
sub_category(Parent,Entry),
add_tree_entry(Parent,New_entry).

perform_action(root,Entry,edit):-
warning_box(8,30,"Root of tree","Can only LINK or VIEW","").

perform_action(Window,Entry,edit):-
input_data("Edit entry",Entry,New_entry),
New_entry\=Entry,                          /* don't edit if no change */
builder_state(_,Type,_),
update_lexicon(add,Type,New_entry),
update_lexicon(delete,Type,Entry),
edit_entry(Entry,New_entry),
edit_entry_list(Entry,New_entry),
edit_frame_data(Entry,New_entry),
create_tree_format,
draw_tree.

perform_action(root,Entry,delete):-
warning_box(8,30,"Root of tree","Can only LINK or VIEW","").

perform_action(Window,Entry,delete):-
builder_state(_,Type,_),
update_lexicon(delete,Type,Entry),
delete_tree_entry(Entry),
create_tree_format,
draw_tree.

perform_action(Window,Entry,link):-
check_link(Window,Entry,Action),
link_action(Entry,Action).

perform_action(Window,Entry,Action).


/* ROUTINES FOR EDITING A NODE */

/* edit the parent node entry */

edit_entry(Entry,New_entry):-
sub_category(Parent,Entry),
retract(tree(Parent,Entry_list)),
substitute(Entry,Entry_list,New_entry,New_entry_list),
assert(tree(Parent,New_entry_list)),
retract(link(Root,Parent,Entry,W)),
assert(link(Root,Parent,New_entry,W)).


/* edit the node itself */

edit_entry_list(Entry,New_entry):-
once(retract(tree(Entry,List))),
once(assert(tree(New_entry,List))),
retract(link(Root,Entry,Descendent,W)),
assert(link(Root,New_entry,Descendent,W)),
fail.

edit_entry_list(Entry,New_entry).


/* edit the frame data for a node */

edit_frame_data(Entry,New_entry):-
builder_state(Root,Type,_),
frame_object(Slot,_),
Archive=..[Slot,Entry,Root,F,A,C],
retract(Archive),
New_archive=..[Slot,New_entry,Root,F,A,C],
assert(New_archive),
fail.

edit_frame_data(Entry,New_entry):-
retract(frame_relation(Entry,Root,N,R)),
assert(frame_relation(New_entry,Root,N,R)),
fail.

edit_frame_data(Entry,New_entry):-
retract(frame_history(F,R,Entry,A,C)),
assert(frame_history(F,R,New_entry,A,C)),
fail.

edit_frame_data(Entry,New_entry).


/* ROUTINES FOR DELETING A NODE */

/* delete the node from the tree */

delete_tree_entry(Entry):-
sub_category(Parent,Entry),
retract(tree(Parent,Entry_list)),
delete(Entry,Entry_list,New_entry_list),
assert(tree(Parent,New_entry_list)),
retractall(link(_,Parent,Entry,_)),
retractall(tree(Parent,[])),
delete_frame_data(Entry),
delete_descendents(Type,Entry),
retractall(tree(Entry,_)),
retractall(link(_,Entry,_,_)).

/* delete the frame data for a node */

delete_frame_data(Entry):-
builder_state(Root,Type,_),
retractall(frame_variable(Entry,Root,_,_,_)),
retractall(frame_history(Entry,Root,_,_,_)),
retractall(frame_symptom(Entry,Root,_,_,_)),
retractall(frame_relation(Entry,Root,_,_)),
retractall(frame_history(F,R,Entry,A,P)).


/* To delete all the descendents of a node */

delete_descendents(Type,Entry):-
once(bagof(Descendent,find_descendent(Entry,Descendent),Descendent_list)
),
member(Descendent_entry,Descendent_list),
retractall(tree(Descendent_entry,_)),
retractall(link(_,Descendent_entry,_,_)),
update_lexicon(delete,Type,Descendent_entry),
once(delete_frame_data(Descendent_entry)),
fail.

delete_descendents(Type,Entry).
```

```
/* ROUTINES FOR ADDING A NODE */

add_tree_entry(Parent,New_entry):-
tree(Parent,Descendents),
length(Descendents,L),
L=10,
warning_box(8,30,"Cannot add to",Parent,"Too many descendents").

add_tree_entry(Parent,New_entry):-
builder_state(Root,Type,_),
update_lexicon(add,Type,New_entry),
retract(tree(Parent,Entry_list)),
assert(tree(Parent,[New_entry|Entry_list])),
assert(link(Root,Parent,New_entry,0.000)),
create_tree_format,
draw_tree.


/* ROUTINES FOR LINK OPTION */

check_link(Window,Entry,weight):-            /* link defined */
sub_category(Entry,_).

check_link(Window,Entry,create):-            /* link undefined */
follows(Window,_).

check_link(Window,Entry,fail):-              /* link undefined */
warning_box(8,30,"Cannot link to",Entry,""),!,fail.


link_action(Parent,create):-
input_data("Connect to","",New_entry),
builder_state(Root,Type,_),
update_lexicon(add,Type,New_entry),
assert(link(Root,Parent,New_entry,0.000)),
assert(tree(Parent,[New_entry])),
create_tree_format,
draw_tree.

link_action(Node,weight):-
builder_state(Root,_,_),
define_weight_window,
draw_weight_window(Root,Node,Drop),
weight_command_loop(Root,Node,Drop),
remove_weight_window.


/* ROUTINES FOR WEIGHTING LINKS */

/* setting the window */

define_weight_window:-
create_stream(weight,readwrite,byte,window(12,29,black on red)),
open(weight,readwrite),
screen(weight,create(5,10,weight,0,0,0,lr,black on red,12,29,hidden)),
state(decimals,_,2).

remove_weight_window:-
close(weight),
delete_stream(weight),
state(decimals,_,3).


/* drawing the window */

draw_weight_window(Root,Node,Drop):-
window(weight,cursor_address(0,1)),
window(weight,text(Node)),
window(weight,cursor_address(0,24)),
window(weight,text("LINK")),
tree(Node,Links),
length(Links,L),
Drop is L+2,
screen(weight,change(5,10,weight,0,0,0,lr,black on red,Drop,29,hidden)),
window(weight,attribute(red on cyan)),
window(weight,cursor_address(1,0)),
draw_weight_entries(Root,Node,Links),
window(weight,attribute(black on red)),
window(weight,text("              EXIT")),
window(weight,attribute(red on cyan)),
screen(weight,unhide).


/* drawing entries in weight window */

draw_weight_entries(Root,Node,[]).

draw_weight_entries(Root,Node,[Descendent|Links]):-
draw_weight_line(Root,Node,Descendent),
draw_weight_entries(Root,Node,Links).


draw_weight_line(Root,Node,Descendent):-
link(Root,Node,Descendent,W),
fill_out(Descendent,Node_display,23),          /* auto lf by window */
Display is_string "\r " & Node_display & string(W,ops) & " ",
window(weight,text(Display)).


/* command loop for adding weights */

weight_command_loop(Root,Node,Drop):-
BY is Drop+4,
retractall(cursor_location(weight,_,_)),
assert(cursor_location(weight,BY,23)),
repeat,
once(cursor_location(weight,SY,SX)),
once(locator(SY,SX,Y,X,6,11,BY,37)),
once(WY is Y-5),
once(retractall(cursor_location(weight,_,_))),
once(assert(cursor_location(weight,Y,X))),
once(weight_action(Root,Node,Drop,WY,X)),
Y=BY,
check_weights(Root,Node).


weight_action(Root,Node,Drop,WY,X):-             /* range for exit */
WY is Drop-1,
X>21,
X<26.

weight_action(Root,Node,Drop,WY,34):-
update_weight_window(Root,Node,WY,u).

weight_action(Root,Node,Drop,WY,36):-
update_weight_window(Root,Node,WY,dp1).

weight_action(Root,Node,Drop,WY,37):-
update_weight_window(Root,Node,WY,dp2).


/* Updating the weight window */

/* do the update */

update_weight_window(Root,Node,WY,Digit):-
get_weight_window_entry(Entry,WY),
retract(link(Root,Node,Entry,W)),
update_weight(W,Digit,NW),
assert(link(Root,Node,Entry,NW)),
Display is_string string(NW,ops),
window(weight,text(Display)).


/* get the node */

get_weight_window_entry(Entry,WY):-
window(weight,cursor_address(WY,1)),
window(weight,inquire_text(22,Input)),
truncate_string(Input,Entry),!,
window(weight,cursor_address(WY,24)).          /* prepare for display */


/* update the weight */

update_weight(1.000,u,0.000).              /* units */

update_weight(W,u,1.000).

update_weight(W,dp1,NW):-              /* first decimal place */
W<0.9,
NW is W+0.100.

update_weight(W,dp1,NW):-
NW is W-0.900.

update_weight(W,dp2,NW):-              /* second decimal place */
U is (10*W)-truncate((10*W)+0.0001),
U<0.9,
NW is W+0.010.

update_weight(W,dp2,NW):-
NW is W-0.090.


/* Checking the sum of link weights for a node */

check_weights(Root,Parent):-
bagof(W,Node^link(Root,Parent,Node,W),Weight_list),
sum_list(Weight_list,0,Sum),
length(Weight_list,N),
R is (1-Sum)/N,
Residue is truncate(R*100)/100,!,
distribute_residue_weights(Root,Parent,Residue).


/* Distribute the residue weight evenly among nodes */

distribute_residue_weights(Root,Parent,0.0).          /* no residue */

distribute_residue_weights(Root,Parent,Residue):-     /* residue -ve*/
Residue<0.0,
warning_box(8,30,"Link weights for",Parent,"badly defined"),!,fail.

distribute_residue_weights(Root,Parent,Residue):-     /* residue +ve */
tree(Parent,Node_list),
member(Node,Node_list),
once(retract(link(Root,Parent,Node,W))),
NW is W+Residue,
assert(link(Root,Parent,Node,NW)),
fail.

distribute_residue_weights(Root,Parent,Residue).

/* ------------------------------------------------------------------ */
/* --------------- ROUTINES TO DRAW AND DISPLAY A TREE -------------- */

/* The drawing of a tree is performed by two main actions. First the
format of the tree is prepared from the database, then the tree is drawn
from the prepared formats */

/* CREATING THE FORMAT LISTS FOR DRAWING TREES */

/* creating the format for a generation */

create_format(Window):-
once(retractall(format(Window,_))),
once(assert(format(Window,[]))),
once(follows(Parent_window,Window)),
once(format(Parent_window,Parent_format_list)),
member((Parent_entry,_),Parent_format_list),
tree(Parent_entry,First_generation_list),
member(First_generation_entry,First_generation_list),
once(tree(First_generation_entry,Second_generation_list)),
once(length(Second_generation_list,Size)),
once(retract(format(Window,Format_list))),
once(append(Format_list,[(space,1),(First_generation_entry,Size)],New_fo
rmat_list)),
once(assert(format(Window,New_format_list))),
fail.

create_format(Window).
```

```
/* Creating the format for a whole tree */

create_tree_format:-
retractall(format(root,_)),
assert(format(root,[(space,46),(root,1)])),
follows(Parent_window,Window),
once(create_format(Window)),
fail.

create_tree_format.


/* ROUTINES TO DRAW THE TREE FROM THE PREPARED FORMATS */

/* Drawing the whole tree: draw the generation for each window */

draw_tree:-
once(draw_generation(root)),
follows(Parent_window,Window),
once(draw_generation(Window)),
fail.

draw_tree:-
display_windows.


/* To draw a generation, get the format, write the nodes, draw the front
brackets and make the connections to the next generation */

draw_generation(Window):-
format(Window,Format_list),
window(Window,clear),
window(Window,cursor_address(0,1)),
draw_generation_entries(Window,Format_list),
window(Window,cursor_address(0,0)),
draw_brackets(Window,Format_list),
retractall(path(_,_)),
assert(path(level,0)),
make_connections(Window,Format_list).


/* DRAWING THE NODES */

/* prepare the node display */

entry_display(Entry,Display):-
tree(Entry,_),
rule_out(Entry,Display,20).

entry_display(Entry,Entry).


/* Draw a cluster of nodes in a subset */

write_entry(Window,(space,N)):-
repeat(X),
window(Window,cursor_down),
X is N-1.

write_entry(Window,(Parent,_)):-
tree(Parent,First_generation_list),
member(First_generation_entry,First_generation_list),
once(entry_display(First_generation_entry,Display)),
once(window(Window,text(Display))),
once(window(Window,text("\r\n "))),
fail.

write_entry(Window,Entry).


/* Draw all nodes for a generation */

draw_generation_entries(Window,[]).

draw_generation_entries(Window,[Entry|Format_list]):-
write_entry(Window,Entry),
draw_generation_entries(Window,Format_list).


/* DRAWING THE FRONT BRACKETS FOR A GENERATION */

/* Main routine to draw the brackets. For each cluster in the
generation, get the bracket display that fits the cluster and then
display it */

draw_brackets(root,_).                    /* don't draw brackets for root */

draw_brackets(Window,[]).                 /* end condition */

draw_brackets(Window,[Entry|Format_list]):-    /* loop for each entry */
bracket_display(Window,Entry),
draw_brackets(Window,Format_list).


/* Retrieve the bracket display that fits the cluster */

bracket_display(Window,(space,N)):-          /* no brackets for spaces */
repeat(X),
window(Window,cursor_down),
X is N-1.

bracket_display(Window,(Category,1)):-       /* one element in group */
window(Window,text("\196\r\n")).

bracket_display(Window,(Category,2)):-       /* two elements in group */
window(Window,text("\194\r\n\192\r\n")).

bracket_display(Window,(Category,Size)):-    /* >2 elements in group */
window(Window,text("\218\r\n")),
Upper_drop is (Size-3)//2,
bracket(Window,Upper_drop),
window(Window,text("\197\r\n")),
Lower_drop is (Size-2)//2,
bracket(Window,Lower_drop),
window(Window,text("\192\r\n")).


/* Display brackets for a drop of N */

bracket(Window,0).

bracket(Window,N):-
repeat(X),
window(Window,text("\195\r\n")),
X is N-1.


/* DRAWING THE CONNECTIONS BETWEEN NODES */

/* Make connections to next generation. First create the list of origins
to the connections. Next reformat the next generation, create the list
of destinations for the connections and finally draw the connections. */

make_connections(Window,Format_list):-
follows(Window,Next_window),
format(Next_window,Next_format_list),
create_origin_list(Format_list,Origin_list,[],0),
Format_list=[(space,N)|Rest],
generation_size(Rest,0,Size),
reformat_list(Origin_list,Size,Next_format_list,New_format_list),
retract(format(Next_window,_)),
assert(format(Next_window,New_format_list)),
create_destination_list(New_format_list,Destination_list,0),
draw_connections(Window,Origin_list,Destination_list).

make_connections(Window,Format_list). /* no connections for last */


/* To create the list of origin positions of connections from the format
list of a generation */

create_origin_list([],Origin_list,Origin_list,Position).

create_origin_list([(space,N)|Format_list],Origin_list,List_so_far,Posit
ion):-
New_position is Position+N,
create_origin_list(Format_list,Origin_list,List_so_far,New_position).

create_origin_list([(Category,N)|Format_list],Origin_list,List_so_far,Po
sition):-
New_position is Position+N,
tree(Category,Category_list),
add_origins(Category_list,Extra_origin_list,Position),
append(List_so_far,Extra_origin_list,New_list_so_far),
create_origin_list(Format_list,Origin_list,New_list_so_far,New_position)
.


/* adding all the origins for a cluster of nodes in a subset */

add_origins([],[],Position).

add_origins([Entry|Category_list],[Position|List],Position):-
New_position is Position+1,
tree(Entry,_),
add_origins(Category_list,List,New_position).

add_origins([Entry|Category_list],List,Position):-
New_position is Position+1,
add_origins(Category_list,List,New_position).


/* Reformat a generation based on the format list of the preceeding
generation. If the generation size is greater than the preceeding
generation then the generation is centred, otherwise it is spaced
out */

reformat_list(Origin_list,Limit_size,Format_list,New_format_list):-
generation_size(Format_list,0,Size),
Size>Limit_size,
centre_generation(Format_list,Size,New_format_list).

reformat_list(Origin_list,Limit_size,Format_list,New_format_list):-
space_out_format(Origin_list,Format_list,New_format_list,0).


/* Finding the size of a generation */

generation_size([],Size,Size).

generation_size([(_,N)|Format_list],Size_so_far,Size):-
New_size_so_far is Size_so_far+N,
generation_size(Format_list,New_size_so_far,Size).

/* Centring a generation in its window */

centre_generation([(space,N)|Rest_of_list],Size,[(space,Offset)|Rest_of_
list]):-
Offset is (95-Size)//2.

/* Getting the offset between origin and destination of a connection */

get_offset(Origin,Destination,0):-
Destination>Origin.

get_offset(Origin,Destination,Offset):-
Offset is Origin-Destination.


/* Spacing out a generation so that origins and destinations match
nicely. If the destination is above the origin then it is left where it
is, otherwise it is spaced down to make a level connection */

space_out_format([],[],[],Position).

space_out_format([Origin|Origin_list],[(space,S)|Format_list],[(space,NS
),(Entry,E)|List],Position):-
Format_list=[(Entry,E)|New_format_list],
Destination is Position+S+((E-1)//2),
get_offset(Origin,Destination,Offset),
NS is S+Offset,
```

```
New_position is Position+NS+E,
space_out_format(Origin_list,New_format_list,List,New_position).


/* To create the list of destination points for the connections */

create_destination_list([],[],Position).

create_destination_list([(space,N)|Format_list],Destination_list,Positio
n):-
New_position is Position+N,
create_destination_list(Format_list,Destination_list,New_position).

create_destination_list([(Category,N)|Format_list],[Destination_position
|Destination_list],Position):-
New_position is Position+N,
Destination_position is Position+((N-1)//2),
create_destination_list(Format_list,Destination_list,New_position).


/* Drawing the connections */

/*
12345  (depth)            direction
|
|'
|-'
|--'                       up
|---'
|----'
|----- ----------------- level
|----,
|---,
|--,                       down
|-,
|.                                  */


/* Draw the connections for a generation from the origin and destination
lists. For each origin-destination pair, get the direction and depth
of the connection and then draw it. */

draw_connections(Window,[],[]).

draw_connections(Window,[Origin|Origin_list],[Destination|Destination_li
st]):-
Connection_length is Origin-Destination,
set_direction(Connection_length,Direction),
set_depth(Direction),
connect(Window,Origin,Connection_length),
draw_connections(Window,Origin_list,Destination_list).


/* Set the direction of the connection between nodes */

set_direction(0,level).

set_direction(Connection_length,up):-
Connection_length>0.

set_direction(Connection_length,down).


/* Set the nesting depth of the connection */

set_depth(level):-                          /* direction is level */
retract(path(Direction,Depth)),
assert(path(level,0)).

set_depth(Direction):-                     /* same direction again */
retract(path(Direction,Depth)),
New_depth is Depth+1,
assert(path(Direction,New_depth)).

set_depth(Direction):-                      /* new direction */
retract(path(_,_)),
assert(path(Direction,1)).


/* To draw a single connection when direction and depth have been set */

connect(Window,Origin,Connection_length):-
path(level,0),
window(Window,cursor_address(Origin,21)),
window(Window,text("\196\196\196\196\196\196")).

connect(Window,Origin,Connection_length):-
path(up,Depth),
window(Window,cursor_address(Origin,21)),
horizontal_connection(Window,Depth),
window(Window,text("\217")),
upward_connection(Window,Connection_length),
window(Window,text("\218")),
Finish_length is 5-Depth,
horizontal_connection(Window,Finish_length).

connect(Window,Origin,Connection_length):-
path(down,Depth),
window(Window,cursor_address(Origin,21)),
Start_length is 5-Depth,
horizontal_connection(Window,Start_length),
window(Window,text("\191")),
downward_connection(Window,Connection_length),
window(Window,text("\192")),
horizontal_connection(Window,Depth).


/* To draw a horizontal connection of given length */

horizontal_connection(Window,0).

horizontal_connection(Window,Length):-
repeat(X),
window(Window,text("\196")),
X is Length-1.


/* To draw an upward connection of given length */

upward_connection(Window,1):-
window(Window,cursor_up),
window(Window,cursor_left).

upward_connection(Window,Length):-
repeat(X),
window(Window,cursor_up),
window(Window,cursor_left),
window(Window,text("\179")),
X is Length-2,
window(Window,cursor_up),
window(Window,cursor_left).


/* To draw a downward connection of given length */

downward_connection(Window,-1):-
window(Window,cursor_down),
window(Window,cursor_left).

downward_connection(Window,Length):-
repeat(X),
window(Window,cursor_down),
window(Window,cursor_left),
window(Window,text("\179")),
X is -Length-2,
window(Window,cursor_down),
window(Window,cursor_left).
```

```
/* build.frm 3rd level of FRAMEBUILDER */

/* Predicates defined in this file:

frame_object/2
retrieve_frame_data/6
retrieve_frame_data/4
retrieve_slot/3
define_frame_command_line/0
draw_frame_command_line/0
frame_command_loop/0
frame_action/2
define_frame_drop_window/0
draw_frame_drop_header/2
display_level/3
write_data/2
frame_drop_window_display/4
draw_frame_drop_window/1
frame_drop_window/1
check_frame_slot/4
check_tree/5
add_frame_slot/4
history_selection/4
history_selection/3
add_history/5
delete_frame_slot/4
frame_slot_command/2
get_attribute/1
get_certainty/1
edit_variable/4
edit_slot/3
frame_data_edit_action/5
define_selection_window/0
header_display/2
selection_position/2
draw_selection_header/2
draw_selection_window/3
form_selection_list/3
display_selection_list/1
write_slot_entry/3
define_selection_command_window/0
selection_command_loop/2
selection_action/3
pad_key/5
define_pad_windows/0
remove_pad_windows/0
display_pad_key/5
display_keys/0
draw_pad/0
get_pad_entry/3
pad_action/2
pad_input/1
relation_drop_window/1
define_relation_drop_window/0
display_relation_list/1
write_relation_entry/3
set_archive/2
draw_relation_drop_window/1
add_relation/2
enter_relation/2
delete_relation/2
relation_command/2
define_test_window/0
split_list/5
split_process/5
define_relation_window/1
display_edit_relation_list/1
draw_relation/2
set_relation_window_position/2
delete_element/3
insert_element/4
get_relation_archive/2
edit_relation/1
edit_relation_action/4          */

/* ------------------------------------------------------------------ */
/* ------------------------ FRAME INSTANCES ------------------------ */

/* definition of frame slots */

frame_object(frame_variable,variable).
frame_object(frame_symptom,symptom).
frame_object(frame_relation,relation).
frame_object(frame_history,history).


/* To retrieve a slot of frame information */

/* Retrieve the root of disorder or disease added as history */

retrieve_frame_data(frame_history,Frame_name,Root,D_root,Attribute,Certa
inty):-
frame_history(Frame_name,Root,Facet,Attribute,Certainty),
not history(Facet,_),
link(D_root,_,Facet,_).

/* Otherwise just get the facet */
retrieve_frame_data(Frame_slot,Frame_name,Root,Facet,Attribute,Certainty
):-
Archive=..[Frame_slot,Frame_name,Root,Facet,Attribute,Certainty],
call(Archive),
frame_object(Frame_slot,Slot),
retrieve_slot(Slot,Facet,_).     /* dont want disorders or diseases */

/* Retrieve facet information only */

retrieve_frame_data(Frame_slot,Frame_name,Root,Facet):-
Archive=..[Frame_slot,Frame_name,Root,Facet,Attribute,Certainty],
call(Archive).


/* To retrieve a slot */

retrieve_slot(variable,Variable,none):-
variable(Variable).

retrieve_slot(Slot,Facet,Attributes):-
Archive=..[Slot,Facet,Attributes],
call(Archive).


/* The main command line for frame instances */

define_frame_command_line:-
create_stream(frame_backdrop,readwrite,byte,window(25,80,black on
black)), open(frame_backdrop,readwrite),
screen(frame_backdrop,create(0,0,frame_backdrop,0,0,0,none,None,25,80,re
vealed)),
draw_frame_command_line,
screen(main,pull_up),
screen(main,unhide),
retractall(cursor_location(frame,SY,SX)),
assert((cursor_location(frame,0,28))).


draw_frame_command_line:-
builder_state(Root,Frame_type,Frame_name),
window(main,cursor_address(0,1)),
window(main,attribute(red on cyan)),
fill_out(Frame_name,Display,22),
window(main,text(Display)),
window(main,cursor_address(0,28)),
window(main,text("Variables  Signs/Symptoms  Relations  History  EXIT
")).


frame_command_loop:-
repeat,
once(cursor_location(frame,SY,SX)),
once(locator(SY,SX,Y,X,0,28,0,79)),
once(retract(cursor_location(frame,SY,SX))),
once(assert(cursor_location(frame,Y,X))),
once(frame_action(X,Result)),
Result=exit.


frame_action(X,exit):-                          /* exit from frame */
X>74,
X<79,
screen(main,hide),
close(frame_backdrop),
delete_stream(frame_backdrop).

frame_action(X,continue):-                  /* set lab data for frame */
X>27,
X<36,
frame_drop_window(frame_variable).

frame_action(X,continue):-             /* set signs/symptoms for frame */
X>37,
X<53,
frame_drop_window(frame_symptom).

frame_action(X,continue):-                 /* set relations for frame */
X>54,
X<64,
relation_drop_window(frame_relation).

frame_action(X,continue):-                   /* set history for frame */
X>65,
X<73,
frame_drop_window(frame_history).

frame_action(X,continue).


/* ------------------------------------------------------------------*/
/* FRAME DROP WINDOWS */

define_frame_drop_window:-
define_drop_window(long),
create_stream(frame_drop_header,readwrite,byte,window(2,45,white on
black)),
open(frame_drop_header,readwrite),
screen(frame_drop_header,create(1,1,frame_drop_header,0,0,0,lrt,blue on
black,2,45,hidden)).


draw_frame_drop_header(Header,Wpos):-
screen(frame_drop_header,change(2,Wpos,frame_drop_header,0,0,0,lrt,blue
on black,2,45,hidden)),
window(frame_drop_header,attribute(bright red on black)),
window(frame_drop_header,cursor_address(0,1)),
window(frame_drop_header,text(Header)),
window(frame_drop_header,cursor_address(1,1)),
repeat(Cycle),
window(frame_drop_header,text("\196")),
Cycle=42,
screen(frame_drop_header,unhide).


display_level(Data,Level,Colour):-
builder_state(Root,Frame_type,Frame_name),
frame_variable(Frame_name,Root,Data,Level,Certainty),
state(decimals,_,1),
Display is_string " " & string(Certainty,ops) & " ",
state(decimals,_,3),
window(long,attribute(black on Colour)),
window(long,text(Display)),
window(long,attribute(red on black)).

display_level(Data,Level,Colour):-
window(long,attribute(Colour on black)),
window(long,text("\176\176\176\176\176")),
window(long,attribute( red on black)).


write_data(Archive,no_data).

write_data(variable(Data_item),data):-       /* only want data that.. */
data_parameter(Data_item,standard_deviation,none)./*.can be classified*/

write_data(variable(Data_item),data):-
window(long,inquire_cursor_address(Y,X)),
```

```
window(long,text(Data_item)),
window(long,cursor_address(Y,25)),
display_level(Data_item,low,blue),
window(long,cursor_address(Y,32)),
display_level(Data_item,usual,green),
window(long,cursor_address(Y,39)),
display_level(Data_item,high,red),
NY is Y+2,
window(long,cursor_address(NY,1)).


write_data(Archive,data):-
Archive=..[Frame_slot,Frame_name,Root,Facet,Attribute,Certainty],
window(long,inquire_cursor_address(Y,X)),
window(long,text(Facet)),
window(long,cursor_address(Y,24)),
window(long,text(Attribute)),
window(long,cursor_address(Y,40)),
state(decimals,_,1),
Display is_string string(Certainty,r) & "  ",
state(decimals,_,3),
window(long,text(Display)),
NY is Y+2,
window(long,cursor_address(NY,1)).


frame_drop_window_display(Title,Archive,N,Wpos):-
N2 is N*2,
((N2<19,Drop is N2+1);(Drop is 19)),
Total_drop is Drop+2,
draw_frame_drop_header(Title,Wpos),
retractall(window_drop(long,_)),
assert(window_drop(long,Total_drop)),
screen(long,change(4,Wpos,long,0,0,0,lrb,blue on black,Drop,45,hidden)),
window(long,clear),
window(long,attribute( red on black)),
window(long,cursor_address(0,1)),
((call(Archive),Data=data);(Data=no_data)),
once(write_data(Archive,Data)),
Data=no_data,
screen(long,unhide).

draw_frame_drop_window(frame_variable):-
number_of_clauses(variable/1,_,N1),
number_of_clauses(data_parameter/3,data_parameter(_,standard_deviation,n
one),N2),
N is N1-N2,            /* N is no of clauses for variable with SD > 0 */
object(frame_variable,Title,Pos,Wpos),
frame_drop_window_display(Title,variable(Data),N,Wpos).

draw_frame_drop_window(Frame_slot):-
builder_state(Root,Frame_type,Frame_name),
Archive=..[Frame_slot,Frame_name,Root,Facet,Attribute,Certainty],
number_of_clauses(Frame_slot/5,Archive,N),
object(Frame_slot,Title,Pos,Wpos),
frame_drop_window_display(Title,Archive,N,Wpos).


frame_drop_window(Frame_slot):-
frame_object(Frame_slot,Object),
highlight_action(Object),
define_frame_drop_window,
draw_frame_drop_window(Frame_slot),
define_sub_command_line,
frame_sub_command_loop(Frame_slot),
close(frame_drop_header),
delete_stream(frame_drop_header),
draw_frame_command_line.


/* ------------------------------------------------------------------*/
/* ---------------------- FRAME SLOT COMMANDS ---------------------- */


/* CHECKS ON SLOT INSTANTIATIONS */

/* check the probability sum for a frame variable slot (must be = 1) */
/* passes if check is ok */

check_frame_slot(frame_variable,Frame_name,Root,[]).

check_frame_slot(frame_variable,Frame_name,Root,[Variable|Variable_list]
):-
once(bagof(C,A^retrieve_frame_data(frame_variable,Frame_name,Root,Variab
le,A,C),Certainties)),
once(sum_list(Certainties,0,Sum)),
Sum=1.0,!,
check_frame_slot(frame_variable,Frame_name,Root,Variable_list).

check_frame_slot(frame_variable,Frame_name,Root,[Variable|Variable_list]
):- warning_box(6,30,Variable,"is wrongly defined",""),!, fail.


/* check the probability sum for a frame slot (must be =< 1) */
/* passes if check is ok */

check_frame_slot(Frame_slot,Frame_name,Root,[]).

check_frame_slot(Frame_slot,Frame_name,Root,[Facet|Facet_list]):-
once(bagof(C,A^retrieve_frame_data(Frame_slot,Frame_name,Root,Facet,A,C)
,Certainties)),
once(sum_list(Certainties,0,Sum)),
Sum=1.0,!,
check_frame_slot(Frame_slot,Frame_name,Root,Facet_list).

check_frame_slot(Frame_slot,Frame_name,Root,[Facet|Facet_list]):-
warning_box(6,30,Facet,"is wrongly defined",""),!,
fail.


/* check tree when adding frame slots */
/* fails if check was ok */

check_tree(Frame_type,Frame_name,Root,Entry,[]):-!,fail.

check_tree(Frame_type,Frame_name,Root,Entry,[Frame|Frame_list]):-
related_to(Frame_type,Root,Frame_name,Frame),
warning_box(8,30,Entry,"is already defined for",Frame).

check_tree(Frame_type,Frame_name,Root,Entry,[Frame|Frame_list]):-!,
check_tree(Frame_type,Frame_name,Root,Entry,Frame_list).


/* add a slot for a frame */

add_frame_slot(Frame_slot,Frame_name,Root,"").

add_frame_slot(frame_history,Frame_name,Root,"disorder"):-
define_selection_window,
draw_selection_window(add,Frame_name,disorder),
define_selection_command_window,
selection_command_loop(1,Entry),!,
history_selection(Frame_name,Root,disorder,Entry).

add_frame_slot(frame_history,Frame_name,Root,"clinical diagnosis"):-
define_selection_window,
draw_selection_window(add,Frame_name,disease),
define_selection_command_window,
selection_command_loop(1,Entry),!,
history_selection(Frame_name,Root,disease,Entry).


add_frame_slot(frame_history,Frame_name,Root,"previous disorder"):-
history_selection(Frame_name,Root,disorder,Root).

add_frame_slot(Frame_slot,Frame_name,Root,Entry):-
setof(Frame,(A,C)^retrieve_frame_data(Frame_slot,Frame,Root,Entry,A,C),F
rame_list),
delete(Frame_name,Frame_list,Check_list),
builder_state(_,Frame_type,_),
check_tree(Frame_type,Frame_name,Root,Entry,Check_list).

add_frame_slot(Frame_slot,Frame_name,Root,Entry):-
Archive=..[Frame_slot,Frame_name,Root,Entry,"unknown",1.0],
get_rank(Frame_slot,Frame_name,Entry,Rank),
assert(Archive,Rank).

/* adding a disorder or disease as history */

history_selection(Frame_name,Root,Type,"").

history_selection(Frame_name,Root,Type,D_root):-
builder_state(_,Frame_type,_),
define_selection_window,
draw_selection_window(add(D_root),Frame_name,Type),
define_selection_command_window,
selection_command_loop(1,Entry),!,
add_history(Frame_type,Frame_name,Root,D_root,Entry).

history_selection(Frame_name,Root,Disorder_root):-
warning_box(8,30,"Cannot add for",Disorder_root,"").


add_history(Frame_type,Frame_name,Root,D_root,"").

add_history(Frame_type,Frame_name,Root,D_root,Entry):-
once(bagof(Facet,(A,C)^frame_history(Frame_name,Root,Facet,A,C),Check_li
st)),
check_tree(Frame_type,Frame_name,D_root,Entry,Check_list).

add_history(Frame_type,Frame_name,Root,D_root,Entry):-
assert(frame_history(Frame_name,Root,Entry,"unknown",1.0)).


/* delete a slot for a frame */

delete_frame_slot(Frame_slot,Frame_name,Root,Entry):-
Archive=..[Frame_slot,Frame_name,Root,Entry,_,_],
retractall(Archive).


frame_slot_command(scroll_up,Frame_slot):-
scroll_window_up(long).

frame_slot_command(scroll_down,Frame_slot):-
scroll_window_down(long).

frame_slot_command(add,frame_variable).

frame_slot_command(add,Frame_slot):-
builder_state(Root,Frame_type,Frame_name),
object(Frame_slot,Title,Spos,Wpos),
define_selection_window,
draw_selection_window(add,Frame_name,Frame_slot),
define_selection_command_window,
selection_command_loop(Spos,Entry),
add_frame_slot(Frame_slot,Frame_name,Root,Entry),
draw_frame_drop_window(Frame_slot).


frame_slot_command(delete,frame_variable).

frame_slot_command(delete,Frame_slot):-
builder_state(Root,Frame_type,Frame_name),
object(Frame_slot,Title,Spos,Wpos),
define_selection_window,
draw_selection_window(delete,Frame_name,Frame_slot),
define_selection_command_window,
selection_command_loop(Spos,Entry),
delete_frame_slot(Frame_slot,Frame_name,Root,Entry),
draw_frame_drop_window(Frame_slot).


frame_slot_command(edit,Frame_slot):-
builder_state(Root,Frame_type,Frame_name),
object(Frame_slot,Title,Pos,Wpos),
window_drop(long,Drop),
Drop>2,
Xend is Wpos+44,
retractall(cursor_location(fde,_,_)),
assert(cursor_location(fde,4,Wpos)),
repeat,
```

```
once(cursor_location(fde,SY,SX)),
once(locator(SY,SX,Y,X,4,Wpos,Drop,Xend)),
once(retractall(cursor_location(fde,_,_))),
once(assert(cursor_location(fde,Y,X))),
screen(long,info(_,_,long,OY,_,_,_,_,_,_,_)),        /* get scroll info */
WY is Y+OY-4,
WX is X-Wpos,
once(window(long,cursor_address(WY,1))),
once(window(long,inquire_text(22,Entry))),
once(truncate_string(Entry,Data)),
once(frame_data_edit_action(Frame_slot,Data,WY,WX,Result)), Result=exit.


frame_slot_command(exit,Frame_slot):-
builder_state(Root,Frame_type,Frame_name),
setof(Facet,(A,C)^retrieve_frame_data(Frame_slot,Frame_name,Root,Facet,A
,C),Facet_list),!,
check_frame_slot(Frame_slot,Frame_name,Root,Facet_list).


frame_slot_command(Result,Frame_slot).


/* ----------------------------------------------------------------*/
/* FRAME DATA EDIT COMMANDS */

/* get the attribute of an observation from the window display */

get_attribute(Attribute):-
window(long,inquire_cursor_address(Y,X)),
window(long,cursor_address(Y,24)),
window(long,inquire_text(15,Entry)),
truncate_string(Entry,Attribute).


/* get the probability of an observation from the window display */

get_certainty(Certainty):-
window(long,inquire_cursor_address(Y,X)),
window(long,cursor_address(Y,40)),
window(long,inquire_text(3,Entry)),
truncate_string(Entry,String),
Certainty is value(String,ir).



edit_variable(Frame_name,Root,Data,Level):-        /* old certainty 1 */
retract(frame_variable(Frame_name,Root,Data,Level,1.0)).

edit_variable(Frame_name,Root,Data,Level):-
retract(frame_variable(Frame_name,Root,Data,Level,Certainty)),
next_element(Certainty,New_certainty,[0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,
0.9,1.0,0]), get_rank(frame_variable,Frame_name,Data,Rank),
assert(frame_variable(Frame_name,Root,Data,Level,New_certainty),Rank).

edit_variable(Frame_name,Root,Data,Level):-        /* old certainty 0 */
not frame_variable(Frame_name,Root,Data,_,_),
once(setof(Frame,(L,C)^retrieve_frame_data(frame_variable,Frame,Root,Dat
a,L,C),Frame_list)),
builder_state(_,Frame_type,_),
check_tree(Frame_type,Frame_name,Root,Data,Check_list).

edit_variable(Frame_name,Root,Data,Level):-        /* old certainty 0 */
assert(frame_variable(Frame_name,Root,Data,Level,0.1)).



edit_slot(WY,WX,Archive):-
WX>23,
WX<40,
Archive=..[Frame_slot,Frame_name,Root,Facet,Attribute,Certainty],
frame_object(Frame_slot,Slot),
Slot_archive=..[Slot,Facet,Attribute_list],
call(Slot_archive),
next_element(Attribute,New_attribute,Attribute_list),
New_archive=..[Frame_slot,Frame_name,Root,Facet,New_attribute,Certainty]
, retract(Archive),
assert(New_archive),
window(long,cursor_address(WY,24)),
fill_out(New_attribute,Display,15),
window(long,text(Display)).

edit_slot(WY,WX,Archive):-              /* default for undefined facets */
WX>23,
WX<40,
Archive=..[Frame_slot,Frame_name,Root,Facet,Attribute,Certainty],
default_attribute(Root,Facet,Attribute_list),
next_element(Attribute,New_attribute,Attribute_list),
New_archive=..[Frame_slot,Frame_name,Root,Facet,New_attribute,Certainty]
, retract(Archive),
assert(New_archive),
window(long,cursor_address(WY,24)),
fill_out(New_attribute,Display,15),
window(long,text(Display)).


edit_slot(WY,WX,Archive):-
WX>39,
WX<43,
Archive=..[Frame_slot,Frame_name,Root,Facet,Attribute,Certainty],
next_element(Certainty,New_certainty,[1.0,0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.
7,0.8,0.9,1.0]),
New_archive=..[Frame_slot,Frame_name,Root,Facet,Attribute,New_certainty]
, retract(Archive),
assert(New_archive),
window(long,cursor_address(WY,40)),
state(decimals,_,1),
Display is_string string(New_certainty,r) & "  ",
state(decimals,_,3),
window(long,text(Display)).

edit_slot(WY,WX,Archive).


frame_data_edit_action(Frame_slot,"",WY,WX,exit).  /* exit if no data */

frame_data_edit_action(frame_variable,Data,WY,WX,continue):-
builder_state(Root,Frame_type,Frame_name),
variable(Data),
WX>24,
WX<30,
edit_variable(Frame_name,Root,Data,low),
window(long,cursor_address(WY,25)),
display_level(Data,low,blue).

frame_data_edit_action(frame_variable,Data,WY,WX,continue):-
builder_state(Root,Frame_type,Frame_name),
variable(Data),
WX>31,
WX<37,
edit_variable(Frame_name,Root,Data,usual),
window(long,cursor_address(WY,32)),
display_level(Data,usual,green).

frame_data_edit_action(frame_variable,Data,WY,WX,continue):-
builder_state(Root,Frame_type,Frame_name),
variable(Data),
WX>38,
WX<44,
edit_variable(Frame_name,Root,Data,high),
window(long,cursor_address(WY,39)),
display_level(Data,high,red).

frame_data_edit_action(Frame_slot,Facet,WY,WX,continue):-
builder_state(Root,Frame_type,Frame_name),
get_attribute(Attribute),
get_certainty(Certainty),
Archive=..[Frame_slot,Frame_name,Root,Facet,Attribute,Certainty],
edit_slot(WY,WX,Archive).


frame_data_edit_action(Frame_slot,Data,WY,WX,exit). /* catch failures */


/* ----------------------------------------------------------------*/

/* FRAME SLOT FILLER SELECTION ROUTINES */

/* define the window for displaying selections */

define_selection_window:-
create_stream(selection,readwrite,byte,window(60,24,white on black)),
open(selection,readwrite),
screen(selection,create(1,1,selection,0,0,0,lrb,blue on
black,23,24,hidden)).

define_selection_window:-              /* close window on backtracking */
close(selection),
delete_stream(selection),
fail.

/* the headers for the selection window */

header_display(delete,"DELETING.....").
header_display(add,"ADDING.....").
header_display(add(Root),"ADDING.....").


/* the position of the selection window */

selection_position(disease,1).

selection_position(Selector,Spos):-
object(Selector,Title,Spos,Wpos).

/* draw the header */

draw_selection_header(Xpos,Action):-
create_stream(selection_header,readwrite,byte,window(1,24,white on
blue)), open(selection_header,readwrite),
screen(selection_header,create(2,Xpos,selection_header,0,0,0,lrt,blue on
black,1,24,hidden)), window(selection_header,cursor_address(0,1)),
header_display(Action,Header),
window(selection_header,text(Header)),
screen(selection_header,unhide).

/* draw the window */

draw_selection_window(Action,Frame_name,Selector):-
selection_position(Selector,Spos),
form_selection_list(Action,Selector,List),
length(List,N),
((N<21,Drop is N+1);(Drop is 21)),
retractall(window_drop(selection,_)),
assert(window_drop(selection,Drop)),
screen(selection,change(3,Spos,selection,0,0,0,lrb,blue on
black,Drop,24,hidden)), window(selection,clear),
window(selection,cursor_address(0,1)),
display_selection_list(List),
draw_selection_header(Spos,Action),
screen(selection,unhide).

/* form the list of selections to display in the window */

form_selection_list(add,disorder,List):-!,
bagof(Root,disorder(Root),L),
builder_state(R,_,_),
delete(R,L,List).

form_selection_list(add,disease,List):-!,
bagof(Root,disease(Root),List).

form_selection_list(add(Root),Type,List):-
bagof(D,descendent(Type,Root,Root,D),List).

form_selection_list(add,Frame_slot,List):-
frame_object(Frame_slot,Slot),
bagof(Facet,A^retrieve_slot(Slot,Facet,A),List).

form_selection_list(delete,Frame_slot,List):-
builder_state(Root,Frame_type,Frame_name),
setof(Facet,retrieve_frame_data(Frame_slot,Frame_name,Root,Facet),List).
```

```
/* display the list of selections in the window */

display_selection_list([]).

display_selection_list([Entry|List]):-
write_window_entry(selection,Entry),
display_selection_list(List).

/* write a selection in the window */

write_slot_entry(add,Archive,N):-
once(Archive=..[Slot,Facet,_]),
call(Archive),
write_window_entry(selection,Facet).

write_slot_entry(delete,Archive,N):-
once(builder_state(Root,Frame_type,Frame_name)),
once(Archive=..[Slot,Facet,_]),
once(frame_object(Frame_slot,Slot)),
once(Frame_archive=..[Frame_slot,Frame_name,Root,Facet,_,_]),
call(Frame_archive),
write_window_entry(selection,Facet).

write_slot_entry(Action,Archive,N):-
window(selection,inquire_cursor_address(N,_)).

/* define the command line for the selection window */

define_selection_command_window:-
create_stream(selection_command,readwrite,byte,window(1,24,white on
blue)), open(selection_command,readwrite),
screen(selection_command,create(0,0,selection_command,0,0,0,none,None,1,
24,hidden)),
window(selection_command,text("\24          EXIT          \25")).

/* command loop for selection window */

selection_command_loop(Spos,Selection):-
window_drop(selection,Drop),
BY is Drop+2,
Xend is Spos+23,
screen(selection_command,change(BY,Spos,selection_command,0,0,0,none,Non
e,1,24,hidden)),
screen(selection_command,unhide),
retractall(cursor_location(selection,_,_)),
assert(cursor_location(selection,BY,Spos)),
repeat,
once(cursor_location(selection,SY,SX)),
once(locator(SY,SX,Y,X,3,Spos,BY,Xend)),
once(WY is Y-3),
once(WX is X-Spos),
once(retractall(cursor_location(selection,_,_))),
once(assert(cursor_location(selection,Y,X))),
once(selection_action(WY,WX,Result)),
Result=exit,
screen(selection,info(_,_,selection,OY,_,_,_,_,_,_,_)),  /*scroll info*/
AY is WY+OY,                                              /* absolute Y pos */
window(selection,cursor_address(AY,1)),
window(selection,inquire_text(22,Entry)),
truncate_string(Entry,Selection),
close(selection_command),
delete_stream(selection_command),
close(selection_header),
delete_stream(selection_header),
close(selection),
delete_stream(selection).


selection_action(WY,WX,scroll_up):-
window_drop(selection,Drop),
WY is Drop-1,
WX=0,
scroll_window_up(selection).

selection_action(WY,WX,scroll_down):-
window_drop(selection,Drop),
WY is Drop-1,
WX=23,
scroll_window_down(selection).

selection_action(WY,WX,exit):-
window_drop(selection,Drop),
WY is Drop-1,
WX>9,
WX<14.


selection_action(WY,WX,exit):-
window_drop(selection,Drop),
WY<Drop-1.

selection_action(WY,WX,continue).


/* ------------------------------------------------------------------ */
/* ----------------------- ENTRY PAD DISPLAY ------------------------ */

pad_key("^"," ^ ",0,1,green).
pad_key("log","log",0,5,green).
pad_key("exp","exp",0,9,green).
pad_key("\127"," \127 ",0,13,green).
pad_key("<"," < ",0,17,red).

pad_key("7"," 7 ",2,1,yellow).
pad_key("8"," 8 ",2,5,yellow).
pad_key("9"," 9 ",2,9,yellow).
pad_key("/"," / ",2,13,green).
pad_key("=<","=< ",2,17,red).

pad_key("4"," 4 ",4,1,yellow).
pad_key("5"," 5 ",4,5,yellow).
pad_key("6"," 6 ",4,9,yellow).
pad_key("*"," * ",4,13,green).
pad_key(">="," >=",4,17,red).

pad_key("1"," 1 ",6,1,yellow).
pad_key("2"," 2 ",6,5,yellow).
pad_key("3"," 3 ",6,9,yellow).
pad_key("-"," - ",6,13,green).
pad_key(">"," > ",6,17,red).

pad_key("0"," 0 ",8,1,yellow).
pad_key("."," . ",8,5,yellow).
pad_key("="," = ",8,9,red).
pad_key("+"," + ",8,13,green).
pad_key("E"," E ",8,17,blue).

pad_key("\24"," \24 ",10,1,blue).
pad_key("("," ( ",10,5,green).
pad_key(")"," ) ",10,9,green).
pad_key("DEL","DEL",10,13,blue).
pad_key("\25"," \25 ",10,17,blue).


define_pad_windows:-
create_stream(pad,readwrite,byte,window(12,21,white on black)),
open(pad,readwrite),
screen(pad,create(2,1,pad,0,0,0,lrt,blue on black,12,21,hidden)),
create_stream(pad_data,readwrite,byte,window(60,23,white on black)),
open(pad_data,readwrite),
screen(pad_data,create(14,1,pad_data,0,0,0,lrb,blue on
black,10,21,hidden)), window(pad_data,cursor_address(0,1)),
retractall(cursor_location(pad,_,_)),
assert(cursor_location(pad,1,1)).

remove_pad_windows:-
close(pad),
delete_stream(pad),
close(pad_data),
delete_stream(pad_data).


display_pad_key(Key,Display,Y,X,Colour):-
window(pad,attribute(black on Colour)),
window(pad,cursor_address(Y,X)),
window(pad,text(Display)),!.


display_keys:-
pad_key(Key,Display,Y,X,Colour),
display_pad_key(Key,Display,Y,X,Colour),
fail.

display_keys.


draw_pad:-
display_keys,
((variable(Data);(Data=no_data)),
once(write_window_entry(pad_data,Data)),
Data=no_data,
number_of_clauses(variable/1,_,N),
((N<11,Drop is N);(Drop is 10)),
retractall(window_drop(pad,_)),
assert(window_drop(pad,Drop)),
screen(pad_data,change(14,1,pad_data,0,0,0,lrb,blue on
black,Drop,21,hidden)),
screen(pad,unhide),
screen(pad_data,unhide).


get_pad_entry(Y,X,Entry):-
Y>11,
screen(pad_data,info(_,_,pad_data,OY,_,_,_,_,_,_,_)), /* scroll info */
WY is Y+OY-12,
window(pad_data,cursor_address(WY,1)),
window(pad_data,inquire_text(22,Display)),
truncate_string(Display,Entry).

get_pad_entry(Y,X,Entry):-
pad_key(Entry,Display,Y,TX,_),
X-TX>=0,
X-TX<3.

get_pad_entry(Y,X,none).


pad_action("\24",continue):-
scroll_window_up(pad_data).

pad_action("\25",continue):-
scroll_window_down(pad_data).

pad_action(none,continue).

pad_action(_,exit).


pad_input(Entry):-
window_drop(pad,Drop),
BY is Drop+13,
repeat,
once(cursor_location(pad,SY,SX)),
once(locator(SY,SX,Y,X,2,1,BY,21)),
once(retractall(cursor_location(pad,SY,SX))),
once(assert(cursor_location(pad,Y,X))),
once(WY is Y-2),
once(WX is X-1),
once(get_pad_entry(WY,WX,Entry)),
once(pad_action(Entry,Action)),
Action=exit.

/* ------------------------------------------------------------------ */
/* RELATIONS DROP WINDOWS */

relation_drop_window(Frame_slot):-
highlight_action(relation),
define_relation_drop_window,
draw_relation_drop_window(Frame_slot),
scroll_to_foot(long),
scroll_to_foot(marker),
define_sub_command_line,
relation_sub_command_loop(Frame_slot),
close(marker),
```

```
delete_stream(marker).

define_relation_drop_window:-
create_stream(long,readwrite,byte,window(60,42,white on black)),
open(long,readwrite),
screen(long,create(1,1,long,0,0,0,none,None,23,42,hidden)),
create_stream(marker,readwrite,byte,window(60,45,red on black)),
open(marker,readwrite),
screen(marker,create(1,1,marker,0,0,0,all,blue on black,23,45,hidden)).


display_relation_list([]):-
window(long,text("\r\n\n")).

display_relation_list([Entry|List]):-
window(long,text(Entry)),
display_relation_list(List).


write_relation_entry(data,Archive,N):-
once(Archive=..[relation,Relation_no,List]),
once(window(long,inquire_cursor_address(N,_))),
once(window(marker,cursor_address(N,0))),    /* position marker cursor */
once(window(marker,text(Relation_no))),
once(display_relation_list(List)),
fail.

write_relation_entry(data,Archive,N):-
once(Archive=..[frame_relation,Frame_name,Root,Relation_no,List]),
once(window(long,inquire_cursor_address(N,_))),
once(window(marker,cursor_address(N,0))),    /* position marker cursor */
once(window(marker,text(Relation_no))),
once(display_relation_list(List)),
fail.

write_relation_entry(no_data,Archive,N):-
window(long,inquire_cursor_address(N,_)).

set_archive(relation,Archive):-
Archive=relation(_,_).

set_archive(frame_relation,Archive):-
builder_state(Root,Frame_type,Frame_name),
Archive=frame_relation(Frame_name,Root,_,_).


draw_relation_drop_window(Frame_slot):-
set_archive(Frame_slot,Archive),
window(long,clear),
window(marker,cursor_address(0,0)),
((call(Archive),Data=data);(Data=no_data)),  /* loop to... */
once(write_relation_entry(Data,Archive,N)),  /* write data in window */
Data=no_data,                                /* end of loop */
((N<21,Drop is N+1);(Drop is 21)),
retractall(window_drop(long,_)),
assert(window_drop(long,Drop)),
screen(long,info(_,_,_,PY,_,_,_,_,_,_,_,_)),     /* get scroll position */
screen(long,change(2,28,long,PY,0,0,none,None,Drop,42,hidden)),
screen(marker,change(2,25,marker,PY,0,0,all,blue on
black,Drop,45,hidden)),
screen(marker,unhide),
screen(long,unhide),
screen(long,pull_up).

/* ---------------------------------------------------------------- */
/* RELATIONS ROUTINES */


add_relation(none,_):-
repeat(X),
T is X+1,
Relation_no is_string string(T,ops),
not relation(Relation_no,_),
assert(relation(Relation_no,[])).

add_relation(Frame_name,Root):-
repeat(X),
T is X+1,
Relation_no is_string string(T,ops),
not frame_relation(Frame_name,Root,Relation_no,_),
assert(frame_relation(Frame_name,Root,Relation_no,[])).


enter_relation(Instructions,Relation_no):-
fedit(3,5,15,Instructions," ","",blue on white,Entry),
truncate_string(Entry,Relation_no).

delete_relation(none,_):-
enter_relation("DELETE No..",Relation_no),
retractall(relation(Relation_no,List)).

delete_relation(Frame_name,Root):-
enter_relation("DELETE No..",Relation_no),
retractall(frame_relation(Frame_name,Root,Relation_no,List)).


relation_command(add,Frame_slot):-
builder_state(Root,Frame_type,Frame_name),
add_relation(Frame_name,Root),
draw_relation_drop_window(Frame_slot),
scroll_to_foot(marker),
scroll_to_foot(long).

relation_command(delete,Frame_slot):-
builder_state(Root,Frame_type,Frame_name),
delete_relation(Frame_name,Root),
draw_relation_drop_window(Frame_slot).

relation_command(edit,Frame_slot):-
window_drop(long,Drop),
Drop>2,
enter_relation("Edit relation",Relation_no),
define_pad_windows,
draw_pad,
edit_relation(Relation_no),
remove_pad_windows,
draw_relation_drop_window(Frame_slot).

relation_command(scroll_up,Frame_slot):-
scroll_window_up(marker),
scroll_window_up(long).

relation_command(scroll_down,Frame_slot):-
scroll_window_down(marker),
scroll_window_down(long).

relation_command(exit,relation):-
retractall(relation(_,[])),
draw_main_command_line.

relation_command(exit,frame_relation):-
retractall(frame_relation(Frame_name,_,[])),
draw_frame_command_line.

relation_command(Result,Frame_slot).

/* ---------------------------------------------------------------- */
/* EDITING RELATIONS */


/* routines to split a list */

define_test_window:-
create_stream(test,readwrite,byte,window(3,42,white on black)),
open(test,readwrite),
screen(test,create(1,1,test,0,0,0,none,None,3,42,hidden)),
window(test,cursor_home).


/* To split a list at the cursor position */

split_list([],Back,Back,Y,X):-          /* position is reached */
window(test,inquire_cursor_address(TY,TX)),
(42*TY)+TX>=(42*Y)+X.

split_list([],[],[],Y,X).               /* if list end is reached */

split_list([Element|Front],Back,[Element|List],Y,X):-
window(test,text(Element)),
split_list(Front,Back,List,Y,X).


split_process(Front,Back,List,Y,X):-
define_test_window,
split_list(Front,Back,List,Y,X),
close(test),
delete_stream(test).


/* display the selected relation for editing */

define_relation_window(PY):-
create_stream(relation,readwrite,byte,window(60,42,black on white)),
open(relation,readwrite),
screen(relation,create(PY,1,relation,0,0,0,none,None,23,42,hidden)).

display_edit_relation_list([]):-
window(relation,erase_end_of_line).

display_edit_relation_list([Entry|List]):-
window(relation,text(Entry)),
display_edit_relation_list(List).

draw_relation(Relation_list,H):-
window(relation,cursor_address(0,0)),
display_edit_relation_list(Relation_list),
window(relation,inquire_cursor_address(Y,X)),
((Y<21,H is Y+1);(H is 21)),
screen(relation,change(2,28,relation,0,0,0,none,None,H,42,revealed)),
screen(relation,unhide).


set_relation_window_position(Relation_no,2):-
window(marker,cursor_address(0,0)),
repeat(Shift),
once(window(marker,inquire_text(2,Entry))),
once(truncate_string(Entry,Relation_entry)),
once(window(marker,cursor_down)),
((Relation_entry=Relation_no);(Shift=59)),
screen(marker,info(SY,SX,marker,WY,WX,D,M,Matt,H,W,R)),
screen(marker,change(SY,SX,marker,Shift,WX,D,M,Matt,H,W,R)),
screen(long,info(SY1,SX1,long,WY1,WX1,D1,M1,Matt1,H1,W1,R1)),
screen(long,change(SY1,SX1,long,Shift,WX1,D1,M1,Matt1,H1,W1,R1)).


/* edit routines */

delete_element(Front,[Element|Back],Result):-
append(Front,Back,Result).

delete_element(Front,[],Front).


insert_element(Front,Back,Element,Result):-
append(Front,[Element|Back],Result).


get_relation_archive(Relation_no,Relation_list):-
builder_state(_,_,none),
relation(Relation_no,Relation_list).

get_relation_archive(Relation_no,Relation_list):-
builder_state(Root,Frame_type,Frame_name),
frame_relation(Frame_name,Root,Relation_no,Relation_list).


edit_relation(Relation_no):-
get_relation_archive(Relation_no,_),          /* fails if no relation */
set_relation_window_position(Relation_no,PY),
define_relation_window(PY),
retractall(cursor_location(relation,_,_)),
assert(cursor_location(relation,PY,28)),
```

```
repeat,
once(cursor_location(relation,SY,SX)),
once(get_relation_archive(Relation_no,Relation_list)),
once(draw_relation(Relation_list,H)),
once(BY is PY+H-1),
once(locator(SY,SX,Y,X,PY,28,BY,69)),
once(retractall(cursor_location(relation,_,_))),
once(assert(cursor_location(relation,Y,X))),
once(WY is Y-2),
once(WX is X-28),
once(pad_input(Pad_entry)),
once(edit_relation_action(Relation_no,WY,WX,Pad_entry)), Pad_entry="E",
close(relation),
delete_stream(relation).

edit_relation(Relation_no).            /* catches case of no archive */


edit_relation_action(Relation_no,Y,X,"E").          /* edit finished */

edit_relation_action(Relation_no,Y,X,"DEL"):-     /* data derivation */
once(builder_state(_,_,none)),
once(get_rank(relation,none,Relation_no,Rank)),
once(retract(relation(Relation_no,List))),
once(split_process(F,B,List,Y,X)),
delete_element(F,B,New_list),
assert(relation(Relation_no,New_list),Rank).

edit_relation_action(Relation_no,Y,X,"DEL"):-        /* frame relation */
builder_state(Root,Frame_type,Frame_name),
once(get_rank(frame_relation,Frame_name,Relation_no,Rank)),
once(retract(frame_relation(Frame_name,Root,Relation_no,List))),
once(split_process(F,B,List,Y,X)),
delete_element(F,B,New_list),
assert(frame_relation(Frame_name,Root,Relation_no,New_list),Rank).

edit_relation_action(Relation_no,Y,X,Pad_entry):-  /* data derivation */
builder_state(_,_,none),
get_rank(relation,none,Relation_no,Rank),
retract(relation(Relation_no,List)),
split_process(F,B,List,Y,X),
insert_element(F,B,Pad_entry,New_list),
assert(relation(Relation_no,New_list),Rank).

edit_relation_action(Relation_no,Y,X,Pad_entry):-   /* frame relation */
builder_state(Root,Frame_type,Frame_name),
get_rank(frame_relation,Frame_name,Relation_no,Rank),
retract(frame_relation(Frame_name,Root,Relation_no,List)),
split_process(F,B,List,Y,X),
insert_element(F,B,Pad_entry,New_list),
assert(frame_relation(Frame_name,Root,Relation_no,New_list),Rank).
```

```
/* dasystem.run loading diagnostic system */

/* Predicates defined in this file:

run_dasystem/0
load_system/0
remove_system/0
set_system/0
set_blackboards/0
set_database/0          */


/* ---------------------------------------------------------------- */
/* --------------------KICK OFF THE SYSTEM------------------------- */

run_dasystem:-
load_system,
once(draw_backdrop),
set_system,
display_current_diseases,
display_master,
remove_disease_window,
close_blackboard_display,
archive_database,
remove_backdrop.

run_dasystem:-                    /* 1st clause fails if no kbase */
remove_backdrop.

/* load the system modules */

load_system:-
open_module(mod1,"\\lda\\das.utl",none,actual),
open_module(mod2,"\\lda\\das.blk",none,actual),
open_module(mod3,"\\lda\\das.ks",none,actual),
open_module(mod4,"\\lda\\das.dia",none,actual),
open_module(mod5,"\\lda\\das.ipt",none,actual).


remove_system:-
close_module(mod1),
close_module(mod2),
close_module(mod3),
close_module(mod4),
close_module(mod5).

/* set up ready for operation */

set_system:-
window(backdrop,cursor_address(5,5)),
window(backdrop,text("Please wait while I prepare myself..")),
set_database,
window(backdrop,text(".")),
set_blackboards,
window(backdrop,text(".")),
set_derivation_information,
window(backdrop,text(".")),
set_relationship_information,
window(backdrop,text(".")),
set_trees,
window(backdrop,cursor_address(5,5)),
window(backdrop,text("                                        ")).

set_blackboards:-
clear_blackboard(Blackboard),
set_blackboard_display.

set_database:-
exists_file("KBASE.DAT"),
current_application(_,Index),
Filename is_string "\\lda\\" & Index & "\\kbase.dat",
reconsult(Filename),
Pat_filename is_string "\\lda\\" & Index & "\\patients.dat",
(exists_file("PATIENTS.DAT"),reconsult(Pat_filename);true),
Lexicon is_string "\\lda\\" & Index & "\\lexicon.dat",
(exists_file("LEXICON.DAT"),reconsult(Lexicon);true),
consult("\\lda\\lexicon.cor").

set_database:-
warning_box(5,5,"NO KNOWLEDGE BASE"),!,fail.


?- run_dasystem.
?- remove_system.
```

```
/* das.utl diagnostic system utilities */

/* Predicates defined in this file:

related_to/4
find_singleton_descendent/4
find_tree_nodes/3
assert_singleton_descendents/3
find_apriori/4
assert_apriori_probabilities/3
set_trees/0
set_tree_information/2
replace_deltas/2
create_dependents_list/3
set_derivation_information/0
set_derivations/0
comparator/2
divide_expression/4
set_relationship_information/0
set_relationships/0
set_occurrence/2
set_change/3
substitute_values/4
create_expression_string/3
evaluate_expression/2
get_facet_entry/3                    */


/* TREE PROCESSING */

/* To see if two nodes are related */

related_to(Type,Root,Node1,Node2):-
descendent(Type,Root,Node1,Node2).

related_to(Type,Root,Node1,Node2):-
descendent(Type,Root,Node2,Node1).


/* set singleton lists */

/* find a singleton descendent of a node */
                                                /* entry is singleton */
find_singleton_descendent(Type,Root,Entry,Entry):-
once(Record=..[Type,Root,Entry,_]),
not call(Record),!.
                                              /* entry not singleton */
find_singleton_descendent(Type,Root,Entry,Descendent):-
descendent(Type,Root,Entry,Descendent),
once(Record=..[Type,Root,Descendent,_]),
not call(Record).


/* Find all the nodes in a tree */

find_tree_nodes(Type,Root,Node_list):-
bagof(Node,descendent(Type,Root,Root,Node),Node_list).

/* assert terms for singleton descendents */

assert_singleton_descendents(Type,Root,[]).

assert_singleton_descendents(Type,Root,[Node|List]):-
bagof(Descendent,find_singleton_descendent(Type,Root,Node,Descendent),Si
ngleton_list),
assert(singleton_descendents(Type,Root,Node,Singleton_list)),
assert_singleton_descendents(Type,Root,List).


/* set apriori probabilities */

/* find the apriori probability of a node from its link weights */

find_apriori(Root,Root,P,P).

find_apriori(Root,Node,P_so_far,P):-
link(Root,Parent,Node,W),
New_P_so_far is P_so_far*W,
find_apriori(Root,Parent,New_P_so_far,P).


/* assert terms for apriori probabilities */

assert_apriori_probabilities(Type,Root,[]).

assert_apriori_probabilities(Type,Root,[Node|List]):-
find_apriori(Root,Node,1.0,P),
assert(apriori(Type,Root,Node,P)),
assert_apriori_probabilities(Type,Root,List).


/* pre-processing for trees */

set_trees:-
disorder(Root),
once(set_tree_information(disorder,Root)),
fail.

set_trees:-
disease(Root),
once(set_tree_information(disease,Root)),
fail.

set_trees.


set_tree_information(Type,Root):-
find_tree_nodes(Type,Root,Node_list),
retractall(singleton_descendents(Type,Root,_,_)),
assert_singleton_descendents(Type,Root,Node_list),
retractall(apriori(Type,Root,_,_)),
assert_apriori_probabilities(Type,Root,Node_list).


/* ---------------------DERIVING VARIABLES--------------------------- */
/* ------------------------------------------------------------------- */

/* search for delta signs in expressions */

replace_deltas([],[]).

replace_deltas(["\127",Element|Expression],[change(Element)|New_expressi
on]):-
variable(Element),
replace_deltas(Expression,New_expression).

replace_deltas([Element|Expression],[Element|New_expression]):-
replace_deltas(Expression,New_expression).


/* create a list of variables in an expression */

create_dependents_list([],Dependents_list,List_so_far):-
setof(Element,member(Element,List_so_far),Dependents_list).

create_dependents_list(["\127",Element|Expression],Dependents_list,List_
so_far):-
variable(Element),
data_parameter(Element,mean,_),
create_dependents_list(Expression,Dependents_list,[change(Element)|List_
so_far]).

create_dependents_list([Element|Expression],Dependents_list,List_so_far)
:- variable(Element),
create_dependents_list(Expression,Dependents_list,[Element|List_so_far])
.

create_dependents_list([Element|Expression],Dependents_list,List_so_far)
:- create_dependents_list(Expression,Dependents_list,List_so_far).


/* data derivation */

set_derivation_information:-
retractall(derivation/3),
set_derivations.

set_derivations:-
relation(Relation_no,[Variable,"="|Expression]),
once(create_dependents_list(Expression,Dependents_list,[])),
once(replace_deltas(Expression,New_expression)),
once(assert(derivation(Variable,New_expression,Dependents_list))), fail.

set_derivations.


/* ---- setting frame relationships ---- */

comparator("=",=).
comparator(">=",>=).
comparator("=<",=<).
comparator("<",<).
comparator(">",>).


/* Divide an expression into lhs, rhs and comparator */

divide_expression([C|Expression],[],Expression,C):-
comparator(C,_).

divide_expression([A|Expression],[A|Left],Right,C):-
divide_expression(Expression,Left,Right,C).


set_relationship_information:-
retractall(relation_evidence/7),
retractall(relation_occurrence/2),
set_relationships.

set_relationships:-
frame_relation(Frame,Root,Relation,Expression),
once(replace_deltas(Expression,New_expression)),
once(divide_expression(New_expression,Left,Right,C)),
once(create_dependents_list(Expression,Dependents_list,[])),
once(comparator(C,Comparator)),
once(assert(relation_evidence(Frame,Root,Relation,Left,Right,Comparator,
Dependents_list))),
once(set_occurrence(Root,Dependents_list)),
fail.

set_relationships.


/* set the occurrence of each relationship */

set_occurrence(Root,Dependents_list):-
relation_occurrence(Root,Dependents_list).

set_occurrence(Root,Dependents_list):-
assert(relation_occurrence(Root,Dependents_list)).


/* ---------- evaluating expressions ---------- */

set_change(Mean,Value,Change):-
Mean>Value,
Change is Mean-Value.

set_change(Mean,Value,Change):-
Change is Value-Mean.

substitute_values(Blackboard,Expression,[],Expression).

substitute_values(Blackboard,Expression,[change(Variable)|Dependents_lis
t],New_expression):-
blackboard(Blackboard,raw_data,variable,Variable,Value,Status),
not member(Value,[low,usual,high]),
not data_parameter(Variable,mean,none),
data_parameter(Variable,mean,M),
set_change(M,Value,C),
V is_string "(" & string(C,ops) & ")",
substitute(change(Variable),Expression,V,Next_expression),!,
substitute_values(Blackboard,Next_expression,Dependents_list,New_express
ion).
```

```
substitute_values(Blackboard,Expression,[Variable|Dependents_list],New_e
xpression):-
blackboard(Blackboard,raw_data,variable,Variable,Value,Status), not
member(Value,[low,usual,high]),
V is_string string(Value,ops),
substitute(Variable,Expression,V,Next_expression),!,
substitute_values(Blackboard,Next_expression,Dependents_list,New_express
ion).


create_expression_string([],Expression_string,String_so_far):-
Expression_string is_string String_so_far & ".".

create_expression_string([Element|Expression],Expression_string,String_s
o_far):-
New_string_so_far is_string String_so_far & Element,
create_expression_string(Expression,Expression_string,New_string_so_far)
.

evaluate_expression(Expression,Value):-
create_expression_string(Expression,Expression_string,""),
V is value(Expression_string),
convert_value(V,Value).

get_facet_entry(Window,AY,Facet):-
window(Window,cursor_address(AY,1)),
window(Window,inquire_text(22,Entry)),
truncate_string(Entry,Facet).
```

```
/* das.blk blackboard control & triggering */

/* Predicates defined in this file:

clear_blackboard/1
set_blackboard_display/0
close_blackboard_display/0
display_blackboard/0
clear_blackboard_display/0
copy_blackboard/2
entry_display/4
diagnosis_exists/4
no_diagnosis_exists/3
make_diagnosis/1
find_triggered_ks/3
active_ks/2
execution_cycle/3
schedule_all_actions/1
schedule_action/3
trigger/4                    */


/* ------------------------------------------------------------------ */
/* ---------------------- BLACKBOARD UTILITIES---------------------- */

/* routines for blackboard set up */

clear_blackboard(Blackboard):-
retractall(ksar(Blackboard,_,_)),
retractall(executed_ks(Blackboard,Knowledge_source,Condition_list)),
retractall(blackboard(Blackboard,Level,_,_)),
retractall(blackboard(Blackboard,Level,_,_,_)),
retractall(blackboard(Blackboard,Level,_,_,_,_)).


set_blackboard_display:-
create_stream(blackboard_background,readwrite,byte,window(20,53,black on
black)),
create_stream(blackboard1,readwrite,byte,window(4,53,bright blue on
black)),
create_stream(blackboard2,readwrite,byte,window(4,53,blue on black)),
create_stream(blackboard3,readwrite,byte,window(4,53,bright cyan on
black)),
create_stream(blackboard4,readwrite,byte,window(4,53,cyan on black)),
open(blackboard_background,readwrite),
open(blackboard1,readwrite),
open(blackboard2,readwrite),
open(blackboard3,readwrite),
open(blackboard4,readwrite),
screen(blackboard_background,create(3,26,blackboard_background,0,0,0,all
,red on red,19,53,hidden)),
screen(blackboard1,create(3,26,blackboard1,0,0,0,none,red on
black,4,53,hidden)),
screen(blackboard2,create(8,26,blackboard2,0,0,0,t,red on
black,4,53,hidden)),
screen(blackboard3,create(13,26,blackboard3,0,0,0,t,red on
black,4,53,hidden)),
screen(blackboard4,create(18,26,blackboard4,0,0,0,t,red on
black,4,53,hidden)).


close_blackboard_display:-
close(blackboard_background),
close(blackboard1),
close(blackboard2),
close(blackboard3),
close(blackboard4),
delete_stream(blackboard_background),
delete_stream(blackboard1),
delete_stream(blackboard2),
delete_stream(blackboard3),
delete_stream(blackboard4).


display_blackboard:-
screen(blackboard_background,info(SY,SX,blackboard_background,WY,WX,D,M,
Matt,H,W,R)),
R==hidden,                                  /* fails if revealed */
screen(blackboard_background,unhide),
screen(blackboard1,unhide),
screen(blackboard2,unhide),
screen(blackboard3,unhide),
screen(blackboard4,unhide),!.

display_blackboard:-
!,
screen(blackboard1,hide),
screen(blackboard2,hide),
screen(blackboard3,hide),
screen(blackboard4,hide),
screen(blackboard_background,hide).


clear_blackboard_display:-
window(blackboard1,clear),
window(blackboard2,clear),
window(blackboard3,clear),
window(blackboard4,clear).


/* Copy a blackboard */

copy_blackboard(From,To):-
once(retractall(blackboard(To,Level,_,_))),
blackboard(From,Level,A1,A2),
assert(blackboard(To,Level,A1,A2)),
fail.

copy_blackboard(From,To):-
once(retractall(blackboard(To,Level,_,_,_))),
blackboard(From,Level,A1,A2,A3),
assert(blackboard(To,Level,A1,A2,A3)),
fail.

copy_blackboard(From,To):-
once(retractall(blackboard(To,Level,_,_,_,_))),
blackboard(From,Level,A1,A2,A3,A4),
assert(blackboard(To,Level,A1,A2,A3,A4)),
fail.

copy_blackboard(From,To):-
once(retractall(executed_ks(To,_,_))),
executed_ks(From,A1,A2),
assert(executed_ks(To,A1,A2)),
fail.

copy_blackboard(From,To).


/* displaying an entry on the blackboard */

entry_display(Entry,Value,Certainty,Display):-
var(Value),
Display is_string string(Entry,ops) & " unknown " &
string(Certainty,ops) & " ".

entry_display(Entry,Value,Certainty,Display):-
Display is_string string(Entry,ops) & " " & string(Value,ops) & " " &
string(Certainty,ops) & " ".


/* check the existence of a diagnosis on the blackboard */

diagnosis_exists(Blackboard,Root,Disorder,P):-
disorder(Root),
blackboard(Blackboard,sub_diagnosis,Root,Disorder,P), P>0.1.

no_diagnosis_exists(Blackboard,Root,Diagnosed_list):- disorder(Root),
not member(Root,Diagnosed_list).


/* ------------------------------------------------------------------ */
/* ----------------------BLACKBOARD CONTROL---------------- --------- */

/* make a dignosis using the blackboard system */

make_diagnosis(Blackboard):-
clear_blackboard_display,
execution_cycle(Blackboard,first_trigger,[data_handler]).


/* find all triggered knowledge sources */

find_triggered_ks(Blackboard,first_trigger,Active_ks):-
once(retractall(ksar(Blackboard,_,_))),
trigger(Blackboard,Knowledge_source,Condition_list,Active_ks),
assert(ksar(Blackboard,Knowledge_source,Condition_list)).
/* fail here to find all triggered ks - one enough for bottom-up */

/* for mixed strategy find all ksars except for data_handlers */

find_triggered_ks(Blackboard,mixed_schedule,Active_ks):-
once(retractall(ksar(Blackboard,_,_))),
trigger(Blackboard,Knowledge_source,Condition_list,Active_ks),
assert(ksar(Blackboard,Knowledge_source,Condition_list)),
member(Knowledge_source,[truth_maintenance,write_raw_data,write_default_
data,derive_data]).


/* for full_schedule find all ksars */

find_triggered_ks(Blackboard,full_schedule,Active_ks):-
once(retractall(ksar(Blackboard,_,_))),
trigger(Blackboard,Knowledge_source,Condition_list,Active_ks),
assert(ksar(Blackboard,Knowledge_source,Condition_list)), fail.

find_triggered_ks(Blackboard,Strategy,Active_ks).


/* check to see if ks is active */

active_ks([],Active_list):-!,fail.

active_ks([Type|Type_list],Active_list):-
member(Type,Active_list),!.

active_ks([Type|Type_list],Active_list):-
active_ks(Type_list,Active_list).


/* blackboard execution cycle */
                                        /* else schedule & execute ks */
execution_cycle(Blackboard,Strategy,Active_ks):-
once(find_triggered_ks(Blackboard,Strategy,Active_ks)),
schedule_action(Blackboard,Strategy,Next_active_ks),!,
execution_cycle(Blackboard,Strategy,Next_active_ks).

                                        /* finish when no triggered ks */
execution_cycle(Blackboard,Strategy,Active_ks).


/* knowledge source execution scheduler */

/* schedule all ksars of one type */

schedule_all_actions(Ks):-
ksar(Blackboard,Knowledge_source,Conditions),
once(call(knowledge_source(Blackboard,Knowledge_source,Conditions,[Ks|_]
))), fail.

schedule_all_actions(Ks).


/* mixed schedule identifies multiple ksar executions */

schedule_action(Blackboard,mixed_schedule,Next_active_ks):-
once(ksar(Blackboard,Knowledge_source,Conditions)),
once(clause(knowledge_source/4,knowledge_source(_,Knowledge_source,_,[Ks
|Next_active_ks]):-G)),
member(Ks,[classify_data,evidence_handler,sum_hypothesis,rank_hypothesis
]),
schedule_all_actions(Ks).
```

```
/* bottom up takes ks in order of triggering */

schedule_action(Blackboard,Strategy,Next_active_ks):-
ksar(Blackboard,Knowledge_source,Condition_list),
call(knowledge_source(Blackboard,Knowledge_source,Condition_list,Next_ac
tive_ks)).



/* ------------------------------------------------------------------*/
/* --------------- TRIGGERING FOR KNOWLEDGE SOURCES --------------- */


/* bottom up triggering */

/* Truth maintenance */

trigger(Blackboard,truth_maintenance,Condition_list,Active_ks):-
active_ks([truth_maintenance,data_handler],Active_ks),
blackboard(Blackboard,raw_data,Type,Facet,Value,Status), not
current_data(Type,Facet,V),
Condition_list=[Type,Facet,Value],
not executed_ks(Blackboard,truth_maintenance,Condition_list).


/* Write raw data */

trigger(Blackboard,write_raw_data,Condition_list,Active_ks):-
active_ks([write_raw_data,data_handler],Active_ks),
current_data(Type,Facet,Value),
hypothesis_type(Type,_),
Condition_list=[Type,Facet,Value],
not blackboard(Blackboard,raw_data,Type,Facet,Value,S).


/* Data derivation */

trigger(Blackboard,derive_data,Condition_list,Active_ks):-
active_ks([derive_data,data_handler],Active_ks),
derivation(Variable,Expression,Dependents_list),
not blackboard(Blackboard,raw_data,variable,Variable,Value,measurement),
substitute_values(Blackboard,Expression,Dependents_list,Instantiated_lis
t), Condition_list=[Variable,Dependents_list,Instantiated_list], not
executed_ks(Blackboard,derive_data,Condition_list).


/* Write default data */

trigger(Blackboard,write_default_data,Condition_list,Active_ks):-
active_ks([write_default_data,data_handler],Active_ks),
data_parameter(Variable,default,Value),
Value\=none,
Condition_list=[Variable,Value],
not blackboard(Blackboard,raw_data,variable,Variable,V,S).


/* Classify raw data */

trigger(Blackboard,classify_data,Condition_list,Active_ks):-
active_ks([classify_data,data_handler],Active_ks),
blackboard(Blackboard,raw_data,variable,Variable,Value,Status), not
data_parameter(Variable,standard_deviation,none),
Condition_list=[Variable,Value],
not executed_ks(Blackboard,classify_data,Condition_list).


/* Considering Variable Evidence */

trigger(Blackboard,variable_evidence,Condition_list,Active_ks):-
active_ks([variable_evidence,evidence_handler],Active_ks),
blackboard(Blackboard,classified_data,Variable,low,Certainty_low),
blackboard(Blackboard,classified_data,Variable,usual,Certainty_usual),
blackboard(Blackboard,classified_data,Variable,high,Certainty_high),
Condition_list=[Variable,Certainty_low,Certainty_usual,Certainty_high],
not executed_ks(Blackboard,variable_evidence,Condition_list).


/* Considering Signs/symptoms */

trigger(Blackboard,evidence,Condition_list,Active_ks):-
active_ks([evidence,evidence_handler],Active_ks),
blackboard(Blackboard,raw_data,symptom,Facet,Attribute,Status),
Condition_list=[symptom,Facet,Attribute],
not executed_ks(Blackboard,evidence,Condition_list).


/* Considering History Evidence */

trigger(Blackboard,evidence,Condition_list,Active_ks):-
active_ks([evidence,evidence_handler],Active_ks),
blackboard(Blackboard,raw_data,history,Facet,Attribute,Status),
Condition_list=[history,Facet,Attribute],
not executed_ks(Blackboard,evidence,Condition_list).


/* Considering Relation Evidence */

trigger(Blackboard,relation_evidence,Condition_list,Active_ks):-
active_ks([relation_evidence,evidence_handler],Active_ks),
relation_occurrence(Root,Dependents_list),
once(substitute_values(Blackboard,Dependents_list,Dependents_list,Instan
tiated_list)),
Condition_list=[Root,Dependents_list,Instantiated_list],
not executed_ks(Blackboard,relation_evidence,Condition_list).


/* Combining the hypotheses */

trigger(Blackboard,sum_hypothesis,Condition_list,Active_ks):-
active_ks([sum_hypothesis],Active_ks),
disorder(Root),
once(setof((Disorder,L),(E,H)^hypothesis_exists(Blackboard,Root,H,Disord
er,E,L),Hypothesis_list)),
Condition_list=[Root,Hypothesis_list],
not executed_ks(Blackboard,sum_hypothesis,Condition_list).


/* Ranking the hypotheses */

trigger(Blackboard,rank_hypothesis,Condition_list,Active_ks):-
active_ks([rank_hypothesis],Active_ks),
disorder(Root),
once(setof((Disorder,L),blackboard(Blackboard,hypothesis,Root,Disorder,L
),Hypothesis_list)),
Condition_list=[Root,Hypothesis_list],
not executed_ks(Blackboard,rank_hypothesis,Condition_list).


/* Transfer the diagnosis of disorders */

trigger(Blackboard,transfer_data,Condition_list,Active_ks):-
active_ks([transfer_data],Active_ks),
once(setof((Root,Disorder,P),diagnosis_exists(Blackboard,Root,Disorder,P
),Condition_list)),
not executed_ks(Blackboard,transfer_data,Condition_list).


/* top down triggering */

/* write disease diagnoses */

trigger(Blackboard,write_disease_diagnosis,Condition_list,Active_ks):-
active_ks([write_disease_diagnosis,data_handler,diagnosis_ks],Active_ks)
,
once(setof((Root,Disease),current_data(disease,Root,Disease),Condition_l
ist)),
not executed_ks(Blackboard,write_disease_diagnosis,Condition_list).


/* predict disorders from disease diagnoses */

trigger(Blackboard,predict_disorders,Condition_list,Active_ks):-
active_ks([predict_disorders],Active_ks),
once(setof((Root,Disease),blackboard(Blackboard,diagnosis,Root,Disease),
Condition_list)),
not executed_ks(Blackboard,predict_disorders,Condition_list).


/* critque disease diagnoses */

trigger(Blackboard,critique_diagnoses,Condition_list,Active_ks):-
active_ks([critique_diagnoses,diagnosis_ks],Active_ks),
once(setof((Disease,Prediction),blackboard(Blackboard,prediction,Disease
,Prediction),Prediction_list)),
disorder(Root),
once(bagof(Disorder,blackboard(Blackboard,manifestations,Root,Disorder),
Manifestation_list)),
type_of_critique(Blackboard,Manifestation_list,Type),
Condition_list=[Root,Type,Manifestation_list,Prediction_list],
not executed_ks(Blackboard,critique_diagnoses,Condition_list).
```

```
/* das.ks knowledge sources */

/* Predicates defined in this file:

knowledge_source/4
truth_maintenance_checks/3
check_derived_data/2
dependent_variable/2
check_relation_data/2
transfer_raw_data/3
derive_variable/4
write_level_entry/4
set_level/3
set_low_level/3
set_usual_level/4
set_high_level/3
get_decimal_places/4
distribution_value/2
assign_distribution_value/2
gaussian/2
hypothesis_type/2
hypothesis_exists/6
sum_frame_certainties/4
get_evidence/6
sum_variable_evidence_certainties/7
get_variable_evidence_probability/6
find_variable_incidence_list/4
assert_variable_evidence/9
impact_variable_evidence/3
sum_evidence_certainties/4
assert_evidence/7
get_evidence_probability/6
find_evidence_incidence_list/6
get_affected_root/3
impact_evidence/3
test_relationship/3
find_relationship_certainty/4
sum_relation_certainties/6
impact_relation_evidence/4
assert_relation_evidence/6
get_current_hypothesis/4
update_hypotheses/4
combine_evidence/3
normalize_likelihoods/3
get_rank/5
set_diagnosis/4
confirmed_diagnosis/3
confirmed_compound_diagnosis/3
compound_diagnosis/4
sum_compound_evidence/4
retract_current_disorders/1
assert_current_disorder/2
transfer_diagnosis_data/2
transfer_data/2
write_disease_diagnosis/2
assert_disorder_hypotheses/2
find_predicted_disorders/3
type_of_critique/3
assert_critique/4
form_critique_list/5                    */


/* ------------------------------------------------------------------ */
/* -------------------- KNOWLEDGE SOURCE KERNELS ------------------- */

/* KNOWLEDGE SOURCE - Truth Maintenance */

knowledge_source(Blackboard,truth_maintenance,[Type,Facet,Value],[data_h
andler,sum_hypothesis]):-
retractall(blackboard(Blackboard,raw_data,Type,Facet,Value,_)),
retractall(executed_ks(Blackboard,write_raw_data,[Type,Facet,_])),
retractall(executed_ks(Blackboard,derive_data,[Facet,_,_])),
retractall(executed_ks(Blackboard,write_default_data,[Facet,_])),
truth_maintenance_checks(Blackboard,Type,Facet),
retractall(executed_ks(Blackboard,truth_maintenance,[Type,Facet,_])),
assert(executed_ks(Blackboard,truth_maintenance,[Type,Facet,Value])).


/* KNOWLEDGE SOURCE - Write Raw Data*/

knowledge_source(Blackboard,write_raw_data,[Type,Facet,Value],[write_raw
_data,write_default_data,derive_data,classify_data,evidence_handler]):-
entry_display(Facet,Value,"",Entry_display),
window(blackboard4,text(Entry_display)),
retractall(blackboard(Blackboard,raw_data,Type,Facet,_,_)),
asserta(blackboard(Blackboard,raw_data,Type,Facet,Value,measurement)),
retractall(executed_ks(Blackboard,derive_data,[Facet,_,_])),
retractall(executed_ks(Blackboard,write_default_data,[Facet,_])),
retractall(executed_ks(Blackboard,write_raw_data,[Type,Facet,_])),
assert(executed_ks(Blackboard,write_raw_data,[Type,Facet,Value])).


/* KNOWLEDGE SOURCE - Write Default Data*/

knowledge_source(Blackboard,write_default_data,[Variable,Value],[write_d
efault_data,derive_data,classify_data,evidence_handler]):-
entry_display(Variable,Value,"",Entry_display),
window(blackboard4,text(Entry_display)),
retractall(blackboard(Blackboard,raw_data,variable,Variable,_,_)),
asserta(blackboard(Blackboard,raw_data,variable,Variable,Value,default))
, transfer_raw_data(Blackboard,Variable,Value),
retractall(executed_ks(Blackboard,write_default_data,[Variable,_])),
assert(executed_ks(Blackboard,write_default_data,[Variable,Value])).


/* KNOWLEDGE SOURCE - Data Derivation */

knowledge_source(Blackboard,derive_data,[Variable,Dependents_list,Expres
sion],[derive_data,write_default_data,classify_data,evidence_handler]):-
retractall(blackboard(Blackboard,raw_data,variable,Variable,_,_)),
derive_variable(Blackboard,Variable,Expression,Value),
entry_display(Variable,Value,"",Entry_display),
window(blackboard4,text(Entry_display)),
retractall(executed_ks(Blackboard,derive_data,[Variable,_,_])),
assert(executed_ks(Blackboard,derive_data,[Variable,Dependents_list,Expr
ession])).


/* KNOWLEDGE SOURCE - Classify raw data */

knowledge_source(supposition,classify_data,[Variable,Value],[classify_da
ta,evidence_handler]):-
member(Value,[low,usual,high]),
retractall(blackboard(supposition,classified_data,Variable,_,_)),
set_level(supposition,Variable,Value),
retractall(executed_ks(supposition,classify_data,[Variable,_])),
assert(executed_ks(supposition,classify_data,[Variable,Value])).

knowledge_source(Blackboard,classify_data,[Variable,Value],[classify_dat
a,evidence_handler]):-
data_parameter(Variable,mean,Mean),
data_parameter(Variable,standard_deviation,SD),
UX is (Value+(2*SD)-Mean)/SD,
LX is (Value-(2*SD)-Mean)/SD,
retractall(blackboard(Blackboard,classified_data,Variable,_,_)),
set_low_level(Blackboard,Variable,UX),
set_usual_level(Blackboard,Variable,UX,LX),
set_high_level(Blackboard,Variable,LX),
retractall(executed_ks(Blackboard,classify_data,[Variable,_])),
assert(executed_ks(Blackboard,classify_data,[Variable,Value])).


/* KNOWLEDGE SOURCE - Considering variable evidence */

knowledge_source(Blackboard,variable_evidence,Condition_list,[evidence_h
andler,sum_hypothesis,diagnosis_ks]):-
Condition_list=[Variable,Certainty_low,Certainty_usual,Certainty_high],
setof(Root,(F,L,C)^frame_variable(F,Root,Variable,L,C),Root_list),
retractall(blackboard(Blackboard,variable_hypothesis,_,_,Variable,_)),
impact_variable_evidence(Blackboard,Root_list,Condition_list),
retractall(executed_ks(Blackboard,variable_evidence,[Variable|_])),
assert(executed_ks(Blackboard,variable_evidence,Condition_list)).


knowledge_source(Blackboard,variable_evidence,Condition_list,[evidence_h
andler,sum_hypothesis,diagnosis_ks]):-
Condition_list=[Variable,Certainty_low,Certainty_usual,Certainty_high],
retractall(executed_ks(Blackboard,variable_evidence,[Variable|_])),
assert(executed_ks(Blackboard,variable_evidence,Condition_list)).


/* KNOWLEDGE SOURCE - Considering Signs & Symptoms / History Evidence */

knowledge_source(Blackboard,evidence,Condition_list,[evidence_handler,su
m_hypothesis,diagnosis_ks]):-
Condition_list=[Evidence_type,Facet,Attribute],
setof(Root,get_affected_root(Evidence_type,Facet,Root),Root_list),
hypothesis_type(Evidence_type,Hypothesis_type),
retractall(blackboard(Blackboard,Hypothesis_type,_,_,Facet,L)),
impact_evidence(Blackboard,Root_list,Condition_list),
retractall(executed_ks(Blackboard,evidence,[Evidence_type,Facet,_])),
assert(executed_ks(Blackboard,evidence,Condition_list)).

knowledge_source(Blackboard,evidence,Condition_list,[evidence_handler,su
m_hypothesis,diagnosis_ks]):-
Condition_list=[Evidence_type,Facet,Attribute],
retractall(executed_ks(Blackboard,evidence,[Evidence_type,Facet,_])),
assert(executed_ks(Blackboard,evidence,Condition_list)).


/* KNOWLEDGE SOURCE - Numerical Relationships Evidence */

knowledge_source(Blackboard,relation_evidence,Condition_list,[evidence_h
andler,sum_hypothesis,diagnosis_ks]):-
Condition_list=[Root,Dependents_list,Instantiated_list],
setof([Frame],(R,Left,Right,Comp)^relation_evidence(Frame,Root,R,Left,Ri
ght,Comp,Dependents_list),Occurrence_list),
retractall(blackboard(Blackboard,relation_hypothesis,Root,_,Dependents_l
ist,L)),
impact_relation_evidence(Blackboard,Root,Dependents_list,Occurrence_list
),
retractall(executed_ks(Blackboard,relation_evidence,[Root,Dependents_lis
t,_])),
assert(executed_ks(Blackboard,relation_evidence,Condition_list)).

knowledge_source(Blackboard,relation_evidence,Condition_list,[evidence_h
andler,sum_hypothesis,diagnosis_ks]):-
Condition_list=[Root,Dependents_list,Instantiated_list],
retractall(executed_ks(Blackboard,relation_evidence,[Root,Dependents_lis
t,_])),
assert(executed_ks(Blackboard,relation_evidence,Condition_list)).


/* KNOWLEDGE SOURCE - Sum Hypotheses */

knowledge_source(Blackboard,sum_hypothesis,[Root,Hypothesis_list],[sum_h
ypothesis,rank_hypothesis]):-
retractall(blackboard(Blackboard,hypothesis,Root,Frame,L)),
combine_evidence(Blackboard,Root,Hypothesis_list),
retractall(executed_ks(Blackboard,sum_hypothesis,[Root,_])),
assert(executed_ks(Blackboard,sum_hypothesis,[Root,Hypothesis_list])).


/* KNOWLEDGE SOURCE - Rank Hypotheses */

knowledge_source(Blackboard,rank_hypothesis,[Root,Hypothesis_list],[rank
_hypothesis,transfer_data,diagnosis_ks]):-
normalize_likelihoods(Hypothesis_list,0,NF),
retractall(blackboard(Blackboard,sub_diagnosis,Root,_,_)),
set_diagnosis(Blackboard,Root,NF,Hypothesis_list),
compound_diagnosis(Blackboard,Root,[Root],[]),!,
retractall(executed_ks(Blackboard,rank_hypothesis,[Root,_])),
assert(executed_ks(Blackboard,rank_hypothesis,[Root,Hypothesis_list])).

knowledge_source(Blackboard,rank_hypothesis,[Root,Hypothesis_list],[rank
_hypothesis,transfer_data,diagnosis_ks]):-
retractall(blackboard(Blackboard,sub_diagnosis,Root,_,_)),
retractall(executed_ks(Blackboard,rank_hypothesis,[Root,_])),
assert(executed_ks(Blackboard,rank_hypothesis,[Root,Hypothesis_list])).
```

```
/* KNOWLEDGE SOURCE - Transfer Data*/

knowledge_source(Blackboard,transfer_data,Condition_list,[data_handler,d
iagnosis_ks]):-
transfer_diagnosis_data(Blackboard,Condition_list),
retractall(executed_ks(Blackboard,transfer_data,_)),
assert(executed_ks(Blackboard,transfer_data,Condition_list)).


/* Knowledge sources used for top down operation */

/* KNOWLEDGE SOURCE - Write Disease Diagnosis */

knowledge_source(Blackboard,write_disease_diagnosis,Condition_list,[pred
ict_disorders]):-
clear_blackboard_display,
retractall(blackboard(Blackboard,diagnosis,_,_)),
write_disease_diagnosis(Blackboard,Condition_list),
retractall(executed_ks(Blackboard,write_disease_diagnosis,_)),
assert(executed_ks(Blackboard,write_disease_diagnosis,Condition_list)).



/* KNOWLEDGE SOURCE - Predict disorders from disease diagnosis */

knowledge_source(Blackboard,predict_disorders,Condition_list,[critique_d
iagnoses]):-
retractall(blackboard(Blackboard,prediction,_,_)),
assert_disorder_hypotheses(Blackboard,Condition_list),
retractall(executed_ks(Blackboard,predict_disorders,_)),
assert(executed_ks(Blackboard,predict_disorders,Condition_list)).


/* KNOWLEDGE SOURCE - Critique clinical diagnosis */

knowledge_source(Blackboard,critique_diagnoses,Condition_list,[critique_
diagnoses]):-
Condition_list=[Root,Type,Manifestation_list,Prediction_list],
retractall(blackboard(Blackboard,critique,Root,_,_,_)),
form_critique_list(Root,Manifestation_list,Prediction_list,List,Type),
assert_critique(Blackboard,Root,List,Type),
retractall(executed_ks(Blackboard,critique_diagnoses,[Root,_,_,_])),
assert(executed_ks(Blackboard,critique_diagnoses,Condition_list)).


/* ---------------------------------------------------------------- */
/* ----------------ROUTINES USED BY KNOWLEDGE SOURCES---------------- */

/* ROUTINES USED BY TRUTH MAINTENANCE KS */

/* checks for variables */

truth_maintenance_checks(Blackboard,variable,Variable):-
retractall(blackboard(Blackboard,classified_data,Variable,_,_)),
retractall(executed_ks(Blackboard,classify_data,[Variable,_])),
check_derived_data(Blackboard,Variable),
check_relation_data(Blackboard,Variable),
retractall(blackboard(Blackboard,variable_hypothesis,_,_,Variable,_)),
retractall(executed_ks(Blackboard,variable_hypothesis,[Variable|_])).

/* checks for symptoms & history */

truth_maintenance_checks(Blackboard,Type,Facet):-
hypothesis_type(Type,Hypothesis),
retractall(blackboard(Blackboard,Hypothesis,_,_,Facet,_)),
retractall(executed_ks(Blackboard,evidence,[Type,Facet,_])).


/* checking derived data variables */

check_derived_data(Blackboard,Variable):-
executed_ks(Blackboard,derive_data,[V,Dependents_list,E]),
once(member(Variable,Dependents_list)),
retractall(current_data(variable,V,_)),
retractall(executed_ks(Blackboard,derive_data,[V,Dependents_list,E])),
fail.

check_derived_data(Blackboard,Variable).

/* find dependent data */

dependent_variable(Variable,Dependents_list):-
member(Variable,Dependents_list),!.

dependent_variable(Variable,Dependents_list):-
member(change(Variable),Dependents_list),!.

/* checking relation hypotheses */

check_relation_data(Blackboard,Variable):-
executed_ks(Blackboard,relation_evidence,[Root,Dependents_list,I]),
dependent_variable(Variable,Dependents_list),
retractall(blackboard(Blackboard,relation_hypothesis,Root,_,Dependents_l
ist,_)),
retractall(executed_ks(Blackboard,relation_evidence,[Root,Dependents_lis
t,I])), fail.

check_relation_data(Blackboard,Variable).


/* ROUTINES USED BY DERIVE DATA KS */

/* Check derived/default data and transfer to database */

transfer_raw_data(supposition,Variable,Value).

transfer_raw_data(Blackboard,Variable,Value):-
data_parameter(Variable,upper_limit,U),
data_parameter(Variable,lower_limit,L),
Value=<U,
Value>=L,
assert(current_data(variable,Variable,Value)).


/* derive data value */

derive_variable(Blackboard,Variable,Expression,Variable_value):-
evaluate_expression(Expression,Variable_value),
asserta(blackboard(Blackboard,raw_data,variable,Variable,Variable_value,
derivation)),
transfer_raw_data(Blackboard,Variable,Variable_value).


/* ROUTINES USED BY CLASSIFY DATA KS */

/* write level of classified data on blackboard */

write_level_entry(Blackboard,Variable,Level,Probability):-
entry_display(Variable,Level,Probability,Entry_display),
window(blackboard3,text(Entry_display)),
asserta(blackboard(Blackboard,classified_data,Variable,Level,Probability
)).


/* setting levels for supposition of specific level */

set_level(Blackboard,Variable,low):-
asserta(blackboard(Blackboard,classified_data,Variable,low,1.0)),
asserta(blackboard(Blackboard,classified_data,Variable,usual,0.0)),
asserta(blackboard(Blackboard,classified_data,Variable,high,0.0)).

set_level(Blackboard,Variable,usual):-
asserta(blackboard(Blackboard,classified_data,Variable,low,0.0)),
asserta(blackboard(Blackboard,classified_data,Variable,usual,1.0)),
asserta(blackboard(Blackboard,classified_data,Variable,high,0.0)).

set_level(Blackboard,Variable,high):-
asserta(blackboard(Blackboard,classified_data,Variable,low,0.0)),
asserta(blackboard(Blackboard,classified_data,Variable,usual,0.0)),
asserta(blackboard(Blackboard,classified_data,Variable,high,1.0)).


/* setting levels of data variables */

set_low_level(Blackboard,Variable,UX):-
assign_distribution_value(UX,D),
P is 1-D,
write_level_entry(Blackboard,Variable,low,P).

set_usual_level(Blackboard,Variable,UX,LX):-
assign_distribution_value(UX,D1),
assign_distribution_value(LX,D2),
P is D1-D2,
write_level_entry(Blackboard,Variable,usual,P).

set_high_level(Blackboard,Variable,LX):-
assign_distribution_value(LX,D),
P is D,
write_level_entry(Blackboard,Variable,high,P).


/* finding probability of classification from look-up table */

/* getting decimal place information */

get_decimal_places(X,XDP1,DP2,E):-
XDP1 is truncate(10*X)/10,          /* X to one decimal place unrounded
*/ DP2 is fix((X-XDP1)*100),              /* 2nd dp rounded as int 0-9
*/ E is (X-XDP1-(DP2/100))*100.           /* extrapolation factor */


/* getting probability distribution value for variable X */

distribution_value(X,D):-
XC is X+0.00001,                    /* prolog2 doesn't round properly */
get_decimal_places(XC,XDP1,DP2,E),
gaussian(XDP1,Distribution),              /* find distribution section */
DL1 is DP2+1,                             /* location in distribution */
DL2 is DP2+2,
arg(DL1,Distribution,D1),
arg(DL2,Distribution,D2),
D is D1+((D2-D1)*E).                      /* D is D1 plus extrapolation */


/* assign probabilty for value X */

assign_distribution_value(X,D):-          /* negative argument */
X<0,
NX is -X,
assign_distribution_value(NX,ND),
D is 1-ND.

assign_distribution_value(X,1.000):-      /* argument >= 3.400 */
X>=3.400.

assign_distribution_value(X,D):-          /* 0 =< argument < 3 */
distribution_value(X,D).


/* look-up table for gaussian distribution */

gaussian(0.0,distribution(0.5000,0.5040,0.5080,0.5120,0.5160,0.5199,0.52
39,0.5279,0.5319,0.5359,0.5398)).
gaussian(0.1,distribution(0.5398,0.5438,0.5478,0.5517,0.5557,0.5596,0.56
36,0.5675,0.5714,0.5753,0.5793)).
gaussian(0.2,distribution(0.5793,0.5832,0.5871,0.5910,0.5948,0.5987,0.60
26,0.6064,0.6103,0.6141,0.6179)).
gaussian(0.3,distribution(0.6179,0.6217,0.6255,0.6293,0.6331,0.6368,0.64
06,0.6443,0.6480,0.6517,0.6554)).
gaussian(0.4,distribution(0.6554,0.6591,0.6628,0.6664,0.6700,0.6736,0.67
72,0.6808,0.6844,0.6879,0.6915)).
gaussian(0.5,distribution(0.6915,0.6950,0.6985,0.7019,0.7054,0.7088,0.71
23,0.7157,0.7190,0.7224,0.7257)).
gaussian(0.6,distribution(0.7257,0.7291,0.7324,0.7357,0.7389,0.7422,0.74
54,0.7486,0.7517,0.7549,0.7580)).
gaussian(0.7,distribution(0.7580,0.7611,0.7642,0.7673,0.7704,0.7734,0.77
64,0.7794,0.7823,0.7852,0.7881)).
gaussian(0.8,distribution(0.7881,0.7910,0.7939,0.7967,0.7995,0.8023,0.80
51,0.8078,0.8106,0.8133,0.8159)).
gaussian(0.9,distribution(0.8159,0.8186,0.8212,0.8238,0.8264,0.8289,0.83
15,0.8340,0.8365,0.8389,0.8413)).
gaussian(1.0,distribution(0.8413,0.8438,0.8461,0.8485,0.8508,0.8531,0.85
```

```
54,0.8577,0.8599,0.8621,0.8643)).
gaussian(1.1,distribution(0.8643,0.8665,0.8686,0.8708,0.8729,0.8749,0.87
70,0.8790,0.8810,0.8830,0.8849)).
gaussian(1.2,distribution(0.8849,0.8869,0.8888,0.8907,0.8925,0.8944,0.89
62,0.8980,0.8997,0.9015,0.9032)).
gaussian(1.3,distribution(0.9032,0.9049,0.9066,0.9082,0.9099,0.9115,0.91
31,0.9147,0.9162,0.9177,0.9192)).
gaussian(1.4,distribution(0.9192,0.9207,0.9222,0.9236,0.9251,0.9265,0.92
79,0.9292,0.9306,0.9319,0.9332)).
gaussian(1.5,distribution(0.9332,0.9345,0.9357,0.9370,0.9382,0.9394,0.94
06,0.9418,0.9429,0.9441,0.9452)).
gaussian(1.6,distribution(0.9452,0.9463,0.9474,0.9484,0.9495,0.9505,0.95
15,0.9525,0.9535,0.9545,0.9554)).
gaussian(1.7,distribution(0.9554,0.9564,0.9573,0.9582,0.9591,0.9599,0.96
08,0.9616,0.9625,0.9633,0.9641)).
gaussian(1.8,distribution(0.9641,0.9649,0.9656,0.9664,0.9671,0.9678,0.96
86,0.9693,0.9699,0.9706,0.9713)).
gaussian(1.9,distribution(0.9713,0.9719,0.9726,0.9732,0.9738,0.9744,0.97
50,0.9756,0.9761,0.9767,0.9772)).
gaussian(2.0,distribution(0.97725,0.97778,0.97831,0.97882,0.97932,0.9798
2,0.98030,0.98077,0.98124,0.98169,0.98214)).
gaussian(2.1,distribution(0.98214,0.98257,0.98300,0.98341,0.98382,0.9842
2,0.98461,0.98500,0.98537,0.98574,0.98610)).
gaussian(2.2,distribution(0.98610,0.98645,0.98679,0.98713,0.98745,0.9877
8,0.98809,0.98840,0.98870,0.98899,0.98928)).
gaussian(2.3,distribution(0.98928,0.98956,0.98983,0.99010,0.99036,0.9906
1,0.99086,0.99111,0.99134,0.99158,0.99180)).
gaussian(2.4,distribution(0.99180,0.99202,0.99224,0.99245,0.99266,0.9928
6,0.99305,0.99324,0.99343,0.99361,0.99379)).
gaussian(2.5,distribution(0.99379,0.99396,0.99413,0.99430,0.99446,0.9946
1,0.99477,0.99492,0.99506,0.99520,0.99534)).
gaussian(2.6,distribution(0.99534,0.99547,0.99560,0.99573,0.99585,0.9959
8,0.99609,0.99621,0.99632,0.99643,0.99653)).
gaussian(2.7,distribution(0.99653,0.99664,0.99674,0.99683,0.99693,0.9970
2,0.99711,0.99720,0.99728,0.99736,0.99744)).
gaussian(2.8,distribution(0.99744,0.99752,0.99760,0.99767,0.99774,0.9978
1,0.99788,0.99795,0.99801,0.99807,0.99813)).
gaussian(2.9,distribution(0.99813,0.99819,0.99825,0.99831,0.99836,0.9984
1,0.99846,0.99851,0.99856,0.99861,0.99865)).
gaussian(3.0,distribution(0.99865,0.99869,0.99873,0.99877,0.99881,0.9988
5,0.99889,0.99893,0.99897,0.99900,0.99903)).
gaussian(3.1,distribution(0.99903,0.99906,0.99909,0.99912,0.99915,0.9991
8,0.99921,0.99924,0.99927,0.99929,0.99931)).
gaussian(3.2,distribution(0.99931,0.99934,0.99936,0.99938,0.99940,0.9994
2,0.99944,0.99946,0.99948,0.99950,0.99952)).
gaussian(3.3,distribution(0.99952,0.99954,0.99956,0.99958,0.99960,0.9996
1,0.99962,0.99963,0.99964,0.99965,0.99966)).


/* GENERAL ROUTINES FOR CONSIDERING EVIDENCE */


/* relation between type of evidence and asserted hypothesis */

hypothesis_type(symptom,symptom_hypothesis).
hypothesis_type(history,history_hypothesis).
hypothesis_type(variable,variable_hypothesis).
hypothesis_type(relation,relation_hypothesis).


/* find an existing hypothesis */

hypothesis_exists(Blackboard,Root,Hypothesis,Disorder,E,L):-
hypothesis_type(_,Hypothesis),
blackboard(Blackboard,Hypothesis,Root,Disorder,E,L).


/* summing apriori probabilities for a list of nodes */

                                        /* total certainty for frame list */
sum_frame_certainties(Root,[],Tfc,Tfc).

sum_frame_certainties(Root,[[Frame|_]|Occurrence_list],Tfc,Tfc_so_far):-
apriori(disorder,Root,Frame,A),
New_tfc_so_far is Tfc_so_far + A,!,
sum_frame_certainties(Root,Occurrence_list,Tfc,New_tfc_so_far).


/* get a piece of evidence from the knowledge base */

get_evidence(symptom,Frame,Root,Facet,Attribute,Certainty):-
frame_symptom(Frame,Root,Facet,Attribute,Certainty).

get_evidence(history,Frame,Root,Facet,Attribute,Certainty):-
frame_history(Frame,Root,Facet,Attribute,Certainty).

/* for diseases and disorders added as history Attribute is present or
absent */
get_evidence(history,Frame,Root,Facet,Attribute,Certainty):-
link(Evidence_root,Facet,Descendent,_),
get_evidence(history,Frame,Root,Descendent,Attribute,Certainty).


get_evidence(variable,Frame,Root,Facet,Attribute,Certainty):-
frame_variable(Frame,Root,Facet,Attribute,Certainty).

get_evidence(variable,Frame,Root,Facet,Attribute,0.0).


/* ROUTINES USED BY VARIABLE EVIDENCE KS */

/* summing the total weight of a piece of variable evidence. The sum is
of the product of (P(l|F)*P(l) + P(u|F)*P(u) + P(h|F)*P(h))*P(F) over
all F. */


sum_variable_evidence_certainties(Root,[],Cl,Cu,Ch,Tec,Tec).

sum_variable_evidence_certainties(Root,[[Frame,Cfl,Cfu,Cfh]|Incidence_li
st],Cl,Cu,Ch,Tec,T_so_far):-
apriori(disorder,Root,Frame,A),
C is A*((Cfl*Cl) + (Cfu*Cu) + (Cfh*Ch)),
New_T_so_far is T_so_far + C,!,
sum_variable_evidence_certainties(Root,Incidence_list,Cl,Cu,Ch,Tec,New_T
_so_far).


/* get the probabilities of low, usual or high levels for a frame */

get_variable_evidence_probability(Frame,Root,Variable,Cfl,Cfu,Cfh):-
once(get_evidence(variable,Frame,Root,Variable,low,Cfl)),
once(get_evidence(variable,Frame,Root,Variable,usual,Cfu)),
once(get_evidence(variable,Frame,Root,Variable,high,Cfh)).


/* find frames with variable evidence */

find_variable_incidence_list(Root,Variable,[],[]).

find_variable_incidence_list(Root,Variable,[Frame|List],[[Frame,Cfl,Cfu,
Cfh]|Incidence_list]):-
get_variable_evidence_probability(Frame,Root,Variable,Cfl,Cfu,Cfh),
find_variable_incidence_list(Root,Variable,List,Incidence_list).


/* assert hypothesis & weight for each node affected by variable */

assert_variable_evidence(Blackboard,Root,Variable,[],Cl,Cu,Ch,Tfc,Tec).

assert_variable_evidence(Blackboard,Root,Variable,[[Frame,Cfl,Cfu,Cfh]|I
ncidence_list],Cl,Cu,Ch,Tfc,0.0):-
entry_display(Frame,Variable,0.0,Entry_display),
window(blackboard2,text(Entry_display)),
assert(blackboard(Blackboard,variable_hypothesis,Root,Frame,Variable,0.0
)),!,
assert_variable_evidence(Blackboard,Root,Variable,Incidence_list,Cl,Cu,C
h,Tfc,0.0).

assert_variable_evidence(Blackboard,Root,Variable,[[Frame,Cfl,Cfu,Cfh]|I
ncidence_list],Cl,Cu,Ch,Tfc,Tec):-
Cf is ((Cfl*Cl) + (Cfu*Cu) + (Cfh*Ch)),
Likelihood is (Cf*Tfc)/Tec,
entry_display(Frame,Variable,Likelihood,Entry_display),
window(blackboard2,text(Entry_display)),
assert(blackboard(Blackboard,variable_hypothesis,Root,Frame,Variable,Lik
elihood)),!,
assert_variable_evidence(Blackboard,Root,Variable,Incidence_list,Cl,Cu,C
h,Tfc,Tec).


/* impact the effect of a classified data variable on a tree */

impact_variable_evidence(Blackboard,[],Condition_list).

impact_variable_evidence(Blackboard,[Root|Root_list],[Variable,Cl,Cu,Ch]
):-
setof(Frame,(L,C)^frame_variable(Frame,Root,Variable,L,C),Occurrence_lis
t),
find_variable_incidence_list(Root,Variable,Occurrence_list,Incidence_lis
t),
sum_frame_certainties(Root,Incidence_list,Tfc,0),sum_variable_evidence_c
ertainties(Root,Incidence_list,Cl,Cu,Ch,Tec,0),
assert_variable_evidence(Blackboard,Root,Variable,Incidence_list,Cl,Cu,C
h,Tfc,Tec),!,
impact_variable_evidence(Blackboard,Root_list,[Variable,Cl,Cu,Ch]).


/* ROUTINES USED BY SIGNS&SYMPTOMS AND HISTORY EVIDENCE KS */


/* summing the total weight of a piece of evidence. The sum is of the
product of P(e|F)*P(F) over all F. */

sum_evidence_certainties(Root,[],Tec,Tec).

sum_evidence_certainties(Root,[(Frame,Ce)|Incidence_list],Tec,T_so_far):
- apriori(disorder,Root,Frame,A),
C is A*Ce,
New_T_so_far is T_so_far + C,!,
sum_evidence_certainties(Root,Incidence_list,Tec,New_T_so_far).


/* assert hypotheses for all nodes affected by evidence */

assert_evidence(Blackboard,Hypothesis_type,Root,Facet,[],Tfc,Tec).

assert_evidence(Blackboard,Hypothesis_type,Root,Facet,[(Frame,Pe_given_F
)|Incidence_list],Tfc,0.0):-
entry_display(Frame,Facet,0.0,Entry_display),
window(blackboard2,text(Entry_display)),
assert(blackboard(Blackboard,Hypothesis_type,Root,Frame,Facet,0.0)),!,
assert_evidence(Blackboard,Hypothesis_type,Root,Facet,Incidence_list,Tfc
,0.0).

assert_evidence(Blackboard,Hypothesis_type,Root,Facet,[(Frame,Pe_given_F
)|Incidence_list],Tfc,Tec):-
Likelihood is (Pe_given_F*Tfc)/Tec,
entry_display(Frame,Facet,Likelihood,Entry_display),
window(blackboard2,text(Entry_display)),
assert(blackboard(Blackboard,Hypothesis_type,Root,Frame,Facet,Likelihood
)),!,
assert_evidence(Blackboard,Hypothesis_type,Root,Facet,Incidence_list,Tfc
,Tec).


/* get the probability of evidence for a given tree node */

get_evidence_probability(history,Frame,Root,Facet,Attribute,Certainty):-
not(history(Facet,_)),
once(bagof(C,get_evidence(history,Frame,Root,Facet,Attribute,C),C_list))
, once(sum_list(C_list,0,Certainty)).

/* if another node in the same class is present/absent */
get_evidence_probability(history,Frame,Root,Facet,Attribute,0.0):-
not(history(Facet,_)),
link(Evidence_root,_,Facet,_),
link(Evidence_root,_,Other_node,_),
Other_node\==Facet,
get_evidence(history,Frame,Root,Other_node,Attribute,Certainty).

get_evidence_probability(Evidence_type,Frame,Root,Facet,Attribute,Certai
nty):-
```

```
get_evidence(Evidence_type,Frame,Root,Facet,Attribute,Certainty).

get_evidence_probability(Evidence_type,Frame,Root,Facet,Attribute,0.0):-
get_evidence(Evidence_type,Frame,Root,Facet,_,_),
not get_evidence(Evidence_type,Frame,Root,Facet,Attribute,_).


/* Find the incidence list of evidence with probabilities */

find_evidence_incidence_list(Root,Evidence_type,Facet,Attribute,[],[]).

find_evidence_incidence_list(Root,Evidence_type,Facet,Attribute,[[Frame]
|Occurrence_list],[(Frame,Certainty)|Incidence_list]):-
once(get_evidence_probability(Evidence_type,Frame,Root,Facet,Attribute,C
ertainty)),
find_evidence_incidence_list(Root,Evidence_type,Facet,Attribute,Occurren
ce_list,Incidence_list).

/* Get a root disorder affected by a piece of evidence */

get_affected_root(Evidence_type,Facet,Root):-
disorder(Root),
once(get_evidence_probability(Evidence_type,_,Root,Facet,_,_)).

/* impact the effect of a piece of evidence on a tree */

impact_evidence(Blackboard,[],Condition_list).

impact_evidence(Blackboard,[Root|Root_list],[Evidence_type,Facet,Attribu
te]):-
setof([Frame],C^get_evidence_probability(Evidence_type,Frame,Root,Facet,
Attribute,C),Occurrence_list),
find_evidence_incidence_list(Root,Evidence_type,Facet,Attribute,Occurren
ce_list,Incidence_list),
sum_frame_certainties(Root,Occurrence_list,Tfc,0),
sum_evidence_certainties(Root,Incidence_list,Tec,0),
hypothesis_type(Evidence_type,Hypothesis_type),
assert_evidence(Blackboard,Hypothesis_type,Root,Facet,Incidence_list,Tfc
,Tec),!,
impact_evidence(Blackboard,Root_list,[Evidence_type,Facet,Attribute]).


/* ROUTINES USED BY RELATIONSHIPS EVIDENCE KS */


/* Test a relationship: fails if relationship does not hold */

test_relationship(Comparator,Left,Right):-
evaluate_expression(Left,Left_value),
evaluate_expression(Right,Right_value),
Test=..[Comparator,Left_value,Right_value],!,
call(Test).


/* find the probability of a relationship holding between variables for
a specific node. P=0 if any relationships for that node do not hold,
otherwise P=1 */


find_relationship_certainty(Blackboard,Dependents_list,[],1.0).

find_relationship_certainty(Blackboard,Dependents_list,[(Left,Right,Comp
)|Occurrence_list],P):-
once(substitute_values(Blackboard,Left,Dependents_list,Instantiated_left
)),
once(substitute_values(Blackboard,Right,Dependents_list,Instantiated_rig
ht)),
test_relationship(Comp,Instantiated_left,Instantiated_right),!,
find_relationship_certainty(Blackboard,Dependents_list,Occurrence_list,P
).

find_relationship_certainty(Blackboard,Dependents_list,Occurrence_list,0
.0).


/* summing the total weight of a piece of evidence. The sum is of the
product of P(e|F)*P(F) over all F. P(e|F)=1 if the relationship holds
for F, otherwise  P(e|F)=0 */

sum_relation_certainties(Blackboard,Root,Dependents_list,[],Tec,Tec).

sum_relation_certainties(Blackboard,Root,Dependents_list,[[Frame]|Occurr
ence_list],Tec,T_so_far):-
bagof((Left,Right,Comp),R^relation_evidence(Frame,Root,R,Left,Right,Comp
,Dependents_list),Frame_occurrence_list),
find_relationship_certainty(Blackboard,Dependents_list,Frame_occurrence_
list,Ce),
apriori(disorder,Root,Frame,A),
C is A*Ce,
New_T_so_far is T_so_far + C,!,
sum_relation_certainties(Blackboard,Root,Dependents_list,Occurrence_list
,Tec,New_T_so_far).


/* impact the effect of a relationship as evidence on a tree */

impact_relation_evidence(Blackboard,Root,Dependents_list,Occurrence_list
):-
sum_frame_certainties(Root,Occurrence_list,Tfc,0),
sum_relation_certainties(Blackboard,Root,Dependents_list,Occurrence_list
,Tec,0),  /* total evidence certainty */
assert_relation_evidence(Blackboard,Root,Dependents_list,Occurrence_list
,Tfc,Tec).


/* assert hypotheses based on relationship evidence */

assert_relation_evidence(Blackboard,Root,Dependents_list,[],Tfc,Tec).

assert_relation_evidence(Blackboard,Root,Dependents_list,[[Frame]|Occurr
ence_list],Tfc,0.0):- entry_display(Frame,"relation",0.0,Entry_display),
window(blackboard2,text(Entry_display)),
assert(blackboard(Blackboard,relation_hypothesis,Root,Frame,Dependents_l
ist,0.0)),!,
assert_relation_evidence(Blackboard,Root,Dependents_list,Occurrence_list
,Tfc,0.0).

assert_relation_evidence(Blackboard,Root,Dependents_list,[[Frame]|Occurr
ence_list],Tfc,Tec):-
bagof((Left,Right,Comp),R^relation_evidence(Frame,Root,R,Left,Right,Comp
,Dependents_list),Frame_occurrence_list),
find_relationship_certainty(Blackboard,Dependents_list,Frame_occurrence_
list,Pe_given_F),
Likelihood is (Pe_given_F*Tfc)/Tec,
entry_display(Frame,"relation",Likelihood,Entry_display),
window(blackboard2,text(Entry_display)),
assert(blackboard(Blackboard,relation_hypothesis,Root,Frame,Dependents_l
ist,Likelihood)),!,
assert_relation_evidence(Blackboard,Root,Dependents_list,Occurrence_list
,Tfc,Tec).


/* ROUTINES USED BY SUM HYPOTHESIS  KS */
/* propagates effects of evidence on single nodes throughout a tree */

/* get the current degree of belief in a hypothesis: L = 1 -> hypothesis
neither confirmaed or disconfirmed L > 1 -> hypothesis confirmed
L < 1 -> hypothesis disconfirmed   */

get_current_hypothesis(Blackboard,Root,Frame,L):-
retract(blackboard(Blackboard,hypothesis,Root,Frame,L)).

get_current_hypothesis(Blackboard,Root,Frame,L):-
apriori(disorder,Root,Frame,L).

/* update the belief in a hypothesis given new impacted evidence */

update_hypotheses(Blackboard,Root,[],Likelihood).

update_hypotheses(Blackboard,Root,[Frame|Singleton_list],Likelihood):-
get_current_hypothesis(Blackboard,Root,Frame,L),
NL is Likelihood*L,
assert(blackboard(Blackboard,hypothesis,Root,Frame,NL)),
update_hypotheses(Blackboard,Root,Singleton_list,Likelihood).


/* propagate effect of evidence to all singleton descendents */

combine_evidence(Blackboard,Root,[]).

combine_evidence(Blackboard,Root,[(Frame,L)|Frame_list]):-
singleton_descendents(disorder,Root,Frame,Singleton_list),
update_hypotheses(Blackboard,Root,Singleton_list,L),
combine_evidence(Blackboard,Root,Frame_list).


/* ROUTINES USED BY RANK HYPOTHESIS  KS */

/* find normalization factor for leaf nodes of tree */

normalize_likelihoods([],0.0,NF):-!,fail.

normalize_likelihoods([],NF,NF).

normalize_likelihoods([(_,L)|Hypothesis_list],NF_so_far,NF):-
New_NF_so_far is NF_so_far + L,
normalize_likelihoods(Hypothesis_list,New_NF_so_far,NF).

/* find the rank order of a diagnosis of strength L */

get_rank(Blackboard,Root,P,C,Rank):-
clause(blackboard/5,blackboard(Blackboard,sub_diagnosis,Root,Disorder,L)
,C), P=<L,
NC is C+1,
get_rank(Blackboard,Root,P,NC,Rank).

get_rank(Blackboard,Root,P,C,Rank):-
Rank is C-1.

/* assert leaf node diagnoses */

set_diagnosis(Blackboard,Root,NF,[]).

set_diagnosis(Blackboard,Root,NF,[(Disorder,L)|Hypothesis_list]):-
P is L/NF,
get_rank(Blackboard,Root,P,1,Rank),
assert((blackboard(Blackboard,sub_diagnosis,Root,Disorder,P)),Rank),
entry_display(Disorder,P,"",Entry_display),
window(blackboard1,text(Entry_display)),
set_diagnosis(Blackboard,Root,NF,Hypothesis_list).


/* A diagnosis is confirmed if its probaility is >= initial value */

confirmed_diagnosis(Blackboard,Root,Disorder):-
blackboard(Blackboard,sub_diagnosis,Root,Disorder,L),
once(apriori(disorder,Root,Disorder,A)),
L>=A.

/* Compound diagnosis is confirmed if all is singletons are confirmed */

confirmed_compound_diagnosis(Blackboard,Root,[]).

confirmed_compound_diagnosis(Blackboard,Root,[Node|Singleton_list]):-
confirmed_diagnosis(Blackboard,Root,Node),
confirmed_compound_diagnosis(Blackboard,Root,Singleton_list).

/* Compounding the diagnosis: breadth first search of tree looking for
nodes whose singleton descendents are all confirmed as diagnoses.
Because the search starts from the root, it is ensured that the maximum
compounding occurs. Once a compound diagnosis has been found, there is
no need to search further down that branch */

compound_diagnosis(Blackboard,Root,[],[]).

/* swap lists */
compound_diagnosis(Blackboard,Root,[],Next_search_list):-
compound_diagnosis(Blackboard,Root,Next_search_list,[]).
```

```
/* compound diagnosis found */
compound_diagnosis(Blackboard,Root,[Node|Search_list],New_search_list):-
singleton_descendents(disorder,Root,Node,Singleton_list),
confirmed_compound_diagnosis(Blackboard,Root,Singleton_list),
sum_compound_evidence(Root,Singleton_list,0,P),
get_rank(Blackboard,Root,P,1,Rank),
assert((blackboard(Blackboard,sub_diagnosis,Root,Node,P)),Rank),!,
compound_diagnosis(Blackboard,Root,Search_list,New_search_list).

/* add descendents to next list */
compound_diagnosis(Blackboard,Root,[Node|Search_list],Next_search_list):
- disorder(Root,Node,Descendents),
append(Descendents,Next_search_list,New_next_search_list),!,
compound_diagnosis(Blackboard,Root,Search_list,New_next_search_list).

/* no descendents */
compound_diagnosis(Blackboard,Root,[Node|Search_list],Next_search_list):
- compound_diagnosis(Blackboard,Root,Search_list,Next_search_list).


/* summing probabilities for a compound diagnosis */

sum_compound_evidence(Root,[],P,P).

sum_compound_evidence(Root,[Diagnosis|List],P_so_far,P):-
retract(blackboard(Blackboard,sub_diagnosis,Root,Diagnosis,L)),
New_P_so_far is P_so_far + L,!,
sum_compound_evidence(Root,List,New_P_so_far,P).


/* ROUTINES USED BY TRANSFER DATA KS */

/* remove disorders from manifestations and current data or raw_data
level */

retract_current_disorders(Blackboard):-
retract(blackboard(Blackboard,manifestations,_,Disorder)),
retractall(current_data(history,Disorder,"present")), fail.

retract_current_disorders(Blackboard).


/* assert disorders as current data */

assert_current_disorder(Disorder,L):-
L>=0.95,
assert(current_data(history,Disorder,"present")).

assert_current_disorder(Disorder,L).


/* transfer diagnoses from sub_diagnosis level to current data and
manifestations */

transfer_diagnosis_data(Blackboard,Condition_list):-
retract_current_disorders(Blackboard),
transfer_data(Blackboard,Condition_list).


transfer_data(Blackboard,[]).

transfer_data(Blackboard,[(Root,Disorder,L)|List]):-
assert_current_disorder(Disorder,L),
assert(blackboard(Blackboard,manifestations,Root,Disorder)),
transfer_data(Blackboard,List).


/* ROUTINES USED BY WRITE DISEASE DIAGNOSIS KS */


write_disease_diagnosis(Blackboard,[]).

write_disease_diagnosis(Blackboard,[(Root,Disease)|List]):-
entry_display(Root,Disease,"",Entry_display),
window(blackboard1,text(Entry_display)),
asserta(blackboard(Blackboard,diagnosis,Root,Disease)),
write_disease_diagnosis(Blackboard,List).


/* ROUTINES USED BY PREDICT DISORDER KS */

/* assert all disorders predicted by a list of diseases */

assert_disorder_hypotheses(Blackboard,[]).

assert_disorder_hypotheses(Blackboard,[(Root,Disease)|Diagnosis_list]):-
bagof(Super_disease,descendent(disease,Root,Super_disease,Disease),Disea
se_list),
find_predicted_disorders(Blackboard,Disease,[Disease|Disease_list]),
assert_disorder_hypotheses(Blackboard,Diagnosis_list).


/* assert all disorders predicted by a disease */

find_predicted_disorders(Blackboard,Disease,[]).

find_predicted_disorders(Blackboard,Disease,[D|Disease_list]):-
frame_history(D,_,Disorder,"present",Probability),
once(entry_display(Disease,Disorder,"",Entry_display)),
once(window(blackboard2,text(Entry_display))),
once(asserta(blackboard(Blackboard,prediction,Disease,Disorder))), fail.

find_predicted_disorders(Blackboard,Disease,[D|Disease_list]):-
find_predicted_disorders(Blackboard,Disease,Disease_list).


/* ROUTINES USED BY CRITIQUE DIAGNOSIS KS */


/* find the type of critique needed */

type_of_critique(Blackboard,[],expected):-!.

type_of_critique(Blackboard,[Disorder|Manifestation_list],consistent):-
blackboard(Blackboard,prediction,Disease,Disorder),!.

type_of_critique(Blackboard,[Disorder|Manifestation_list],Type):-
type_of_critique(Blackboard,Manifestation_list,Type).


/* assert the critique */

assert_critique(Blackboard,Root,[],Type).

assert_critique(Blackboard,Root,[(Disease,Disorder)|List],Type):-
assert(blackboard(Blackboard,critique,Root,Disease,Disorder,Type)),
entry_display(Disease,Disorder,Type,Entry_display),
window(blackboard3,text(Entry_display)),
assert_critique(Blackboard,Root,List,Type).


/* Form the list of critiques */

form_critique_list(Root,Manifestation_list,[],[],Type).

form_critique_list(Root,Manifestation_list,[(Disease,Disorder)|Predictio
n_list],[(Disease,Disorder)|List],consistent):-
member(Disorder,Manifestation_list),
form_critique_list(Root,Manifestation_list,Prediction_list,List,consiste
nt).

form_critique_list(Root,Manifestation_list,[(Disease,Disorder)|Predictio
n_list],[(Disease,Disorder)|List],expected):-
descendent(disorder,Root,Root,Disorder),
form_critique_list(Root,Manifestation_list,Prediction_list,List,expected
).

form_critique_list(Root,Manifestation_list,[_|Prediction_list],List,Type
):-
form_critique_list(Root,Manifestation_list,Prediction_list,List,Type).
```

```
/* das.ipt data input/output facitilies */

/* Predicates defined in this file:

entry_box/5
define_input_window/3
change_stream/2
string_input/1
get_rest_of_string/3
in_string/1
enter_string/4
convert_value/2
check_string/1
check_list/1
in_number/1
enter_data/4
get_slot_type/2
enter_attribute/2
update_data/3
update_data/2
value_display/2
get_value/2
get_units/2
create_data_display/2
get_attribute/3
create_slot_display/3
draw_backdrop/0
remove_backdrop/0
set_master_menu/0
remove_master_menu/0
display_master/0
master_action/1
set_data_time/0
get_data_time/2
personal_data/0
define_personal_data_window/0
display_name/0
display_age/0
display_sex/0
display_occupation/0
draw_personal_data_window/0
toggle_sex/2
personal_data_command_loop/0
personal_command/3
refine_diagnosis_input/0
define_refining_backdrops/0
draw_refining_backdrop/0
reveal_display/0
remove_refining_windows/0
define_symptom_window/0
slot_display/2
display_slots/1
draw_symptom_window/0
define_variable_window/0
display_variables/0
draw_variable_window/0
refine_diagnosis_command_loop/0
refine_diagnosis_action/3
input_lab_data/0
define_lab_windows/0
remove_lab_windows/0
draw_lab_window/0
lab_data_command_loop/0
lab_data_action/3
select_patient/0
define_dbase_windows/0
remove_dbase_windows/0
draw_dbase_head/0
write_display/2
dbase_display/0
draw_dbase_window/0
draw_dbase_foot/1
dbase_command_loop/0
select_identity/1
add_patient/1
delete_patient/1
dbase_command/2
display_selected_patient/2
patient_selection/1
load_patient/0
set_current_diseases/0
create_archive_data/0
archive_patient/1
set_patient/0
set_personal_data/1
remove_personal_data/0
remove_current_data/0
archive_database/0
view_diagnosis/1
define_diagnosis_windows/0
close_diagnosis_windows/0
write_diagnoses/1
display_diagnoses/1
draw_diagnosis_backdrop/1
view_diagnosis_command_loop/1
view_diagnosis_action/3
patient_diseases/0
display_current_diseases/0
define_disease_foot/0
draw_disease_foot/1
define_disease_window/0
remove_disease_window/0
display_diseases/0
draw_disease_window/0
disease_command_loop/0
add_disease/2
check_diseases/2
disease_display_command/2
disease_selection/2
define_selection_window/0
define_selection_command_window/0
close_selection_windows/0
draw_selection_header/0
draw_selection_window/1
form_selection_list/2
display_selection_list/1
selection_command_loop/2
selection_action/3                          */


/* ------------------------------------------------------------------------ */
/* ----------------------DATA INPUT ROUTINES----------------------------- */

/* enter string in a box */

entry_box(Y,X,L,Instructions,Entry):-
fedit(Y,X,L,Instructions," ","",red on cyan,Input),
truncate_string(Input,Entry).


/* routines to enter string */

define_input_window(Y,X,L):-
create_stream(input,readwrite,byte,window(1,L,bright red on cyan)),
open(input,readwrite),
screen(input,create(Y,X,input,0,0,0,none,None,1,L,hidden)),
window(input,cursor_home),
screen(input,unhide).

change_stream(Old_stream,New_stream):-        /* change current stream */
see(New_stream).

change_stream(Old_stream,New_stream):-      /* reverts on backtracking */
see(Old_stream).


/* routine to input string from current stream */

string_input(String):-
get0(C),
get_rest_of_string(C,"",String).

get_rest_of_string(13,String,String).

get_rest_of_string(C,S,String):-
in_string(C),
S1 is string S & [C],
get0(C1),
get_rest_of_string(C1,S1,String).

get_rest_of_string(C,S,String):-
get0(C1),
get_rest_of_string(C1,S,String).

in_string(C):-C>31,C<123.


/* enter string of a specified length */

enter_string(Y,X,L,Entry):-
define_input_window(Y,X,L),
seeing(Stream),
change_stream(Stream,input),
once(string_input(Input)),
once(truncate_string(Input,Entry)),
see(Stream),!,
close(input),
delete_stream(input).

enter_string(Y,X,L,""):-!.


/* convert number to real */

convert_value(Value,Value):-
real(Value).

convert_value(Value,V):-
V is float(Value).

/* check that a string contains only digits '-' and dp */

check_string(""):-!,fail.

check_string(String):-
list(List,String),
check_list(List).

check_list([]).

check_list([C|List]):-
in_number(C),
check_list(List).

in_number(C):-C>47,C<58.        /* digit */
in_number(45).                  /* -ve */
in_number(46).                  /* dp */


/* enter 7 digit number */

enter_data(Y,X,L,Value):-
enter_string(Y,X,L,Entry),
check_string(Entry),
V is value(Entry,ir),
convert_value(V,Value).

enter_data(Y,X,L,none).


/* enter a frame attribute */

get_slot_type(Facet,symptom):-
symptom(Facet,_).

get_slot_type(Facet,history):-
history(Facet,_).


enter_attribute(Slot,Facet):-
get_attribute(Slot,Facet,Attribute),
Slot_archive=..[Slot,Facet,Attribute_list],
call(Slot_archive),
```

```
next_element(Attribute,New_attribute,Attribute_list),
update_data(Slot,Facet,New_attribute).


/* check input values and update database */

/* signs&symptoms or history */

update_data(Slot,Facet,"unknown"):-
retractall(current_data(Slot,Facet,_)).

update_data(Slot,Facet,Attribute):-
retractall(current_data(Slot,Facet,_)),
assert(current_data(Slot,Facet,Attribute)).


/* variables */

update_data(Variable,none):-
retractall(current_data(variable,Variable,_)).

update_data(Variable,Value):-
data_parameter(Variable,lower_limit,none),
Warning is_string "Lower limit undefined",
warning_box(8,30,Warning).

update_data(Variable,Value):-
data_parameter(Variable,lower_limit,Lower_limit),
Value<Lower_limit,                          /* check lower limit */
Warning is_string "Lower limit " & string(Lower_limit,ops),
warning_box(8,30,Warning).

update_data(Variable,Value):-
data_parameter(Variable,upper_limit,none),
Warning is_string "Upper limit undefined",
warning_box(8,30,Warning).

update_data(Variable,Value):-
data_parameter(Variable,upper_limit,Upper_limit),
Value>Upper_limit,                          /* check upper limit */
Warning is_string "Upper limit " & string(Upper_limit,ops),
warning_box(8,30,Warning).

update_data(Variable,Value):-
retractall(current_data(variable,Variable,_)),
asserta(current_data(variable,Variable,Value)).

update_data(Variable,Value).



/* ------------------------------------------------------------------ */
/* ----------------------DATA DISPLAY ROUTINES---------------------- */

/* displaying values */

value_display(Value,"        "):-
Value=unknown.

value_display(Value,Display):-
Value=< -10.0,
Value> -100.0,
Display is_string string(Value,ops).

value_display(Value,Display):-
Value< 0.0,
Value> -10.0,
Display is_string " " & string(Value,ops).

value_display(Value,Display):-
Value>=0.0,
Value<10.0,
Display is_string "  " & string(Value,ops).

value_display(Value,Display):-
Value>=10.0,
Value<100.0,
Display is_string " " & string(Value,ops).

value_display(Value,Display):-
Value>=100.0,
Value<1000.0,
Display is_string string(Value,ops).


value_display(Value,"        ").


/* displaying variables */

get_value(Variable,Value):-
current_data(variable,Variable,Value).

get_value(Variable,unknown).


get_units(Variable,Units):-
data_parameter(Variable,units,Units).

get_units(Variable,"").


create_data_display(Variable,Data_display):-
get_value(Variable,Value),
get_units(Variable,Units),
value_display(Value,Value_display),
fill_out(Variable,Variable_display,23),
fill_out(Units,Units_display,7),
Data_display is_string Variable_display & Value_display & " " &
Units_display & "\n\r ".


/* displaying slots */

get_attribute(Slot,Facet,Attribute):-
current_data(Slot,Facet,Attribute).

get_attribute(Slot,Facet,"unknown").

create_slot_display(Facet,Attribute,Display):-
fill_out(Facet,Facet_display,23),
fill_out(Attribute,Attribute_display,15),
Display is_string Facet_display & Attribute_display & "\n\r ".



/* ------------------------------------------------------------------ */
/* ------------------------BACKDROP ROUTINES------------------------ */

draw_backdrop:-
create_stream(backdrop,readwrite,byte,window(25,80,red on black)),
open(backdrop,readwrite),
screen(backdrop,create(0,0,backdrop,0,0,0,none,None,25,80,revealed)),
create_stream(header,readwrite,byte,window(1,80,bright white on red)),
open(header,readwrite),
screen(header,create(0,0,header,0,0,0,none,None,1,80,hidden)),
window(header,cursor_address(0,0)),
window(header,text(" ICU Data Analysis System  V1.2 ")),
screen(header,unhide).


remove_backdrop:-
close(header),
delete_stream(header),
close(backdrop),
delete_stream(backdrop).


/* ------------------------------------------------------------------ */
/* --------------------DEEFINE THE MASTER MENU--------------------- */

set_master_menu:-
create_stream(master,readwrite,byte,window(10,16,white on black)),
open(master,readwrite),
screen(master,create(3,1,master,0,0,0,all,black on red,10,16,hidden)),
retractall(menu_selection(master,_)),
assert(menu_selection(master,0)).

remove_master_menu:-
close(master),
delete_stream(master).


display_master:-
set_master_menu,
repeat,
once(menu_selection(master,Selection)),
once(menu(master,"BGAS MASTER",
["Dialogue"-"@"-true-0-help, "Blackboard"-"@"-true-1-help, "New
Patient"-"@"-true-2-help, "Personal Data"-"@"-true-3-help, "Lab Data"-
"@"-true-4-help, "View Diagnosis"-"@"-true-5-help, "Make Diagnosis"-"@"-
true-6-help, "Set Diseases"-"@"-true-7-help, ""-"@"-true-8-help,
"Quit"-"@"-true-9-help],Option,Selection)), once(screen(master,unhide)),
once(retract(menu_selection(master,Selection))),
once(assert(menu_selection(master,Option))),
once(master_action(Option)),
Option==9,                          /* fails until Exit selected */
remove_master_menu.


master_action(0):-
dialogue.

master_action(1):-
display_blackboard.

master_action(2):-
clear_blackboard(disorder),
clear_blackboard_display,
select_patient.

master_action(3):-
personal_data.

master_action(4):-
input_lab_data.

master_action(5):-
view_diagnosis(disorder).

master_action(6):-
make_diagnosis(disorder),
view_diagnosis(disorder).

master_action(7):-
patient_diseases.

master_action(9):-
current_data(demographic,"identity",Id),
Display is_string "Archiving " & Id,
decision_box(3,55,Display,Result),
archive_patient(Result).

master_action(Selection).              /* catches failures */


/* ------------------------------------------------------------------ */
/* -------------------- TIME OF DATA ENTRY ------------------------ */

/* The time is set when variable data is input */

/* set the time of data input */

set_data_time:-
retractall(current_data_time/2),
date(D,M,Y),
Date=D-M-Y,
time(Hour,Min,Sec),
Time=Hour-Min,
assert(current_data_time(Date,Time)).
```

```
/* get the time of data input */

get_data_time(Date,Time):-
current_data_time(Date,Time).

get_data_time(Date,Time):-
date(D,M,Y),
Date=D-M-Y,
time(Hour,Min,Sec),
Time=Hour-Min.


/* ------------------------------------------------------------------ */
/* ----------------------INPUT OF PERSONAL DATA---------------------- */


/* routines to display personal data */

personal_data:-
define_personal_data_window,
draw_personal_data_window,
personal_data_command_loop,
close(personal),
delete_stream(personal).

personal_data:-
close(personal),
delete_stream(personal).

define_personal_data_window:-
create_stream(personal,readwrite,byte,window(6,39,red on cyan)),
open(personal,readwrite),
screen(personal,create(3,22,personal,0,0,0,all,red on
cyan,6,39,hidden)).

display_name:-
current_data(demographic,"identity",Id),
personal_data(Id,name,Name),
fill_out(Name,Display,20),
window(personal,attribute(bright red on cyan)),
window(personal,cursor_address(2,7)),
window(personal,text(Display)).

display_age:-
current_data(variable,"age",Age_no),
Age is_string string(Age_no,ops),
fill_out(Age,Display,3),
window(personal,attribute(bright red on cyan)),
window(personal,cursor_address(2,31)),
window(personal,text(Display)).

display_sex:-
current_data(history,"sex",Sex),
fill_out(Sex,Display,6),
window(personal,attribute(bright red on cyan)),
window(personal,cursor_address(4,31)),
window(personal,text(Display)).

display_occupation:-
current_data(history,"occupation",Occupation),
fill_out(Occupation,Display,15),
window(personal,attribute(bright red on cyan)),
window(personal,cursor_address(4,12)),
window(personal,text(Display)).


draw_personal_data_window:-
current_data(demographic,"identity",Id),
window(personal,attribute(red on cyan)),
window(personal,cursor_address(0,1)),
window(personal,text("Personal Data     Patient Id ")),
fill_out(Id,Id_display,6),
window(personal,text(Id_display)),
window(personal,cursor_address(2,1)),
window(personal,text("Name                    Age")),
window(personal,cursor_address(4,1)),
window(personal,text("Occupation              Sex")),
display_name,
display_age,
display_occupation,
display_sex,
screen(personal,unhide).


/* routines to edit personal data */

toggle_sex("male","female").
toggle_sex("female","male").

personal_data_command_loop:-
retractall(cursor_location(personal,_,_)),
assert(cursor_location(personal,3,22)),
repeat,
once(cursor_location(personal,SY,SX)),
once(locator(SY,SX,Y,X,3,22,8,61)),
once(retract(cursor_location(personal,_,_))),
once(assert(cursor_location(personal,Y,X))),
once(personal_command(Y,X,Result)),
Result=exit.


personal_command(Y,X,name):-
Y=5,
X>22,
X<48,
enter_string(5,29,20,Name),
current_data(demographic,"identity",Id),
retractall(personal_data(Id,name,_)),
assert(personal_data(Id,name,Name)),
display_name.

personal_command(Y,X,age):-
Y=5,
X>47,
X<55,
enter_data(5,53,3,Entry),
update_data("age",Entry),
display_age.

personal_command(Y,X,occupation):-
Y=7,
X>22,
X<48,
enter_attribute(history,"occupation"),
display_occupation.

personal_command(Y,X,sex):-
Y=7,
X>47,
X<58,
retract(current_data(history,"sex",Sex)),
toggle_sex(Sex,New_sex),
assert(current_data(history,"sex",New_sex)),
display_sex.


personal_command(Y,X,exit).


/* ------------------------------------------------------------------ */
/* -----------------INPUT FOR REFINING DIAGNOSIS-------------------- */

/* main loop for input of information */

refine_diagnosis_input:-
define_refining_backdrops,
draw_refining_backdrop,
draw_symptom_window,
draw_variable_window,
reveal_display,
refine_diagnosis_command_loop,
remove_refining_windows.


/* backdrops */

define_refining_backdrops:-
create_stream(refine_backdrop,readwrite,byte,window(24,80,black on
black)),
open(refine_backdrop,readwrite),
screen(refine_backdrop,create(1,0,refine_backdrop,0,0,0,none,None,24,80,
hidden)),
create_stream(symptom_backdrop,readwrite,byte,window(21,39,black on
red)), open(symptom_backdrop,readwrite),
screen(symptom_backdrop,create(2,0,symptom_backdrop,0,0,0,none,None,21,3
9,hidden)),
create_stream(variable_backdrop,readwrite,byte,window(21,39,black on
red)),
open(variable_backdrop,readwrite),
screen(variable_backdrop,create(2,41,variable_backdrop,0,0,0,none,None,2
1,39,hidden)).

draw_refining_backdrop:-
window(refine_backdrop,attribute(black on red)),
window(refine_backdrop,cursor_address(23,0)),
window(refine_backdrop,text("                                    REFINE
DIAGNOSIS                                  ")).


/* window handling */

reveal_display:-
screen(refine_backdrop,unhide),
screen(symptom_backdrop,unhide),
screen(symptom,unhide),
screen(variable_backdrop,unhide),
screen(variable,unhide).

remove_refining_windows:-
close(refine_backdrop),
close(symptom_backdrop),
close(variable_backdrop),
close(symptom),
close(variable),
delete_stream(refine_backdrop),
delete_stream(symptom_backdrop),
delete_stream(variable_backdrop),
delete_stream(symptom),
delete_stream(variable).


/* display signs,symptoms,history */

/* define window */

define_symptom_window:-
create_stream(symptom,readwrite,byte,window(80,40,red on cyan)),
open(symptom,readwrite),
screen(symptom,create(3,0,symptom,0,0,0,none,None,19,39,hidden)).


/* display a slot in the window */

slot_display(history,"disorder").

slot_display(Slot,Facet):-
get_attribute(Slot,Facet,Attribute),
create_slot_display(Facet,Attribute,Display),
window(symptom,text(Display)).


/* display all slots */

display_slots(Slot):-
once(Archive=..[Slot,Facet,List]),
call(Archive),
once(slot_display(Slot,Facet)),
fail.

display_slots(Slot).


/*  draw the signs/symptom/history window */
```

```
draw_symptom_window:-
define_symptom_window,
window(symptom_backdrop,cursor_address(0,0)),
window(symptom_backdrop,text("    SIGNS    SYMPTOMS    HISTORY      ")),
window(symptom_backdrop,cursor_address(20,0)),
window(symptom_backdrop,text("\24
\25")), window(symptom,cursor_address(0,1)),
display_slots(symptom),
display_slots(history).


/* display data variables */

/* define window */

define_variable_window:-
create_stream(variable,readwrite,byte,window(80,40,red on cyan)),
open(variable,readwrite),
screen(variable,create(3,41,variable,0,0,0,none,None,19,39,hidden)).


/* display all variables */

display_variables:-
variable(Variable),
once(create_data_display(Variable,Display)),
once(window(variable,text(Display))),
fail.

display_variables.


draw_variable_window:-
define_variable_window,
window(variable_backdrop,cursor_address(0,0)),
window(variable_backdrop,text(" DATA VARIABLES")),
window(variable_backdrop,cursor_address(20,0)),
window(variable_backdrop,text("\24
\25")), window(variable,cursor_address(0,1)),
display_variables.


/* command loop for input of data */

refine_diagnosis_command_loop:-
retractall(cursor_location(refine,_,_)),
assert(cursor_location(refine,24,4)),
repeat,
once(cursor_location(refine,SY,SX)),
once(locator(SY,SX,Y,X,3,0,24,79)),
once(retractall(cursor_location(refine,_,_))),
once(assert(cursor_location(refine,Y,X))),
once(refine_diagnosis_action(Y,X,Action)),
Action=exit.


refine_diagnosis_action(22,0,continue):-          /* window scrolls */
scroll_window_up(symptom).

refine_diagnosis_action(22,38,continue):-
scroll_window_down(symptom).

refine_diagnosis_action(22,41,continue):-
scroll_window_up(variable).

refine_diagnosis_action(22,79,continue):-
scroll_window_down(variable).


refine_diagnosis_action(Y,X,continue):-                /* enter symptom */
X<39,
Y<22,
screen(symptom,info(_,_,symptom,OY,_,_,_,_,_,_,_,_)),  /*get scroll info*/
WY is Y+OY-3,
get_facet_entry(symptom,WY,Facet),                        /* find facet */
Facet \= "",
get_slot_type(Facet,Slot),
enter_attribute(Slot,Facet),                          /* enter attribute */
window(symptom,cursor_address(WY,1)),                 /* position cursor */
slot_display(Slot,Facet).                             /* update display */


refine_diagnosis_action(Y,X,continue):-            /* enter variable */
X>40,
Y<22,
screen(variable,info(_,_,variable,OY,_,_,_,_,_,_,_,_)),  /* scroll info */
WY is Y+OY-3,
get_facet_entry(variable,WY,Variable),                    /* find facet */
Variable \= "",
enter_data(Y,65,7,Entry),
update_data(Variable,Entry),
create_data_display(Variable,Data_display),
window(variable,cursor_address(WY,1)),                /* position cursor */
window(variable,text(Data_display)).                  /* update display */


refine_diagnosis_action(Y,X,exit):-
Y=24.

refine_diagnosis_action(Y,X,continue).                /* catch failures */


/* ------------------------------------------------------------------ */
/* -------------------- INPUT OF LABORATORY DATA -------------------- */

/* main loop for input of data */

input_lab_data:-
define_lab_windows,
draw_lab_window,
set_data_time,
lab_data_command_loop,
remove_lab_windows.


/* define the windows */

define_lab_windows:-
create_stream(variable_backdrop,readwrite,byte,window(22,40,black on
red)),
open(variable_backdrop,readwrite),
screen(variable_backdrop,create(2,20,variable_backdrop,0,0,0,lr,black on
red,22,40,hidden)),
create_stream(variable,readwrite,byte,window(80,40,red on cyan)),
open(variable,readwrite),
screen(variable,create(3,20,variable,0,0,0,none,None,20,40,hidden)).

remove_lab_windows:-
close(variable),
delete_stream(variable),
close(variable_backdrop),
delete_stream(variable_backdrop).


/* display lab data variables */

draw_lab_window:-
window(variable_backdrop,cursor_address(0,0)),
window(variable_backdrop,text(" LABORATORY DATA        VALUE   UNITS")),
window(variable_backdrop,cursor_address(21,0)),
window(variable_backdrop,text("\24                    EXIT
\25")),
window(variable,cursor_address(0,1)),
display_variables,
screen(variable_backdrop,unhide),
screen(variable,unhide).


/* command loop for data input */

lab_data_command_loop:-
retractall(cursor_location(lab_data,_,_)),
assert(cursor_location(lab_data,1,1)),
repeat,
once(cursor_location(lab_data,SY,SX)),
once(locator(SY,SX,Y,X,3,20,23,59)),
once(retractall(cursor_location(lab_data,_,_))),
once(assert(cursor_location(lab_data,Y,X))),
once(lab_data_action(Y,X,Action)),
Action=exit.


lab_data_action(23,X,exit):-
X>37,
X<42.

lab_data_action(23,20,continue):-
scroll_window_up(variable).

lab_data_action(23,59,continue):-
scroll_window_down(variable).

lab_data_action(Y,X,continue):-                        /* enter variable */
Y<23,
screen(variable,info(_,_,variable,OY,_,_,_,_,_,_,_,_)),/*get scroll info*/
WY is Y+OY-3,
once(get_facet_entry(variable,WY,Variable)),           /* find variable */
Variable \= "",
enter_data(Y,44,7,Entry),
update_data(Variable,Entry),
create_data_display(Variable,Data_display),
window(variable,cursor_address(WY,1)),              /* position cursor */
window(variable,text(Data_display)).                 /* update display */

lab_data_action(Y,X,continue).


/* ------------------------------------------------------------------ */
/* ---------------------PATIENT DATABASE HANDLING-------------------- */

select_patient:-
define_dbase_windows,
draw_dbase_head,
draw_dbase_window,
dbase_command_loop,
remove_dbase_windows.

define_dbase_windows:-
create_stream(dbase,readwrite,byte,window(80,29,red on cyan)),
open(dbase,readwrite),
screen(dbase,create(5,22,dbase,0,0,0,lrb,red on cyan,19,29,hidden)),
create_stream(dbase_head,readwrite,byte,window(2,29,red on cyan)),
open(dbase_head,readwrite),
screen(dbase_head,create(3,22,dbase_head,0,0,0,lrt,red on
cyan,2,29,hidden)),
create_stream(dbase_foot,readwrite,byte,window(1,29,white on red)),
open(dbase_foot,readwrite),
screen(dbase_foot,create(1,22,dbase_foot,0,0,0,none,None,1,29,hidden)).

remove_dbase_windows:-
close(dbase_head),
delete_stream(dbase_head),
close(dbase),
delete_stream(dbase),
close(dbase_foot),
delete_stream(dbase_foot).


draw_dbase_head:-
window(dbase_head,cursor_address(0,1)),
window(dbase_head,text("Patient Name          Id no.")),
window(dbase_head,text("\r\n
\196\196\196\196\196\196\196\196\196\196\196\196\196\196\196\196\196\196\196\196
\196\196\196\196\196\196\196\196\196\196")).

write_display(Name,Id):-
fill_out(Name,Name_display,21),
window(dbase,text(Name_display)),
fill_out(Id,Id_display,6),
window(dbase,text(Id_display)),
window(dbase,text("\n\r ")).
```

```
dbase_display:-
once(window(dbase,clear)),
once(window(dbase,cursor_address(0,1))),
personal_data(Id,name,Name),
once(write_display(Name,Id)),
fail.

dbase_display.


draw_dbase_window:-
number_of_clauses(personal_data/3,personal_data(_,name,_),N),
((N<19,Drop is N+1);(Drop is 19)),
retractall(window_drop(dbase,_)),
assert(window_drop(dbase,Drop)),
screen(dbase,change(5,22,dbase,0,0,0,lrb,red on cyan,Drop,29,hidden)),
dbase_display,
scroll_to_foot(dbase),
screen(dbase_head,unhide),
screen(dbase,unhide).


/* routines for dbase commands */

draw_dbase_foot(Drop):-
Y is Drop+4,
screen(dbase_foot,change(Y,22,dbase_foot,0,0,0,none,None,1,29,hidden)),
window(dbase_foot,cursor_address(0,0)),
window(dbase_foot,text("\24 Add  Delete  Select  Exit \25")),
screen(dbase_foot,unhide).


dbase_command_loop:-
retractall(cursor_location(dbase_foot,_,_)),
assert(cursor_location(dbase_foot,0,22)),
repeat,
once(window_drop(dbase,Drop)),
once(CY is Drop+4),
once(draw_dbase_foot(Drop)),
once(cursor_location(dbase_foot,_,SX)),
once(locator(CY,SX,Y,X,CY,22,CY,50)),
once(retract(cursor_location(dbase_foot,_,_))),
once(assert(cursor_location(dbase_foot,0,X))),
once(dbase_command(X,Result)),
Result=exit.


select_identity(Id):-
repeat(X),
Idx is_string "PT" & string(X,ops),
not personal_data(Idx,_,_),
Id=Idx.

add_patient(Name):-
select_identity(Id),
assert(personal_data(Id,name,Name)),
assert(personal_data(Id,age,25)),
assert(personal_data(Id,sex,"female")),
assert(personal_data(Id,occupation,"unknown")).


delete_patient(Id):-                          /* delete current patient */
retract(current_data(demographic,"identity",Id)),
display_selected_patient("",""),
retractall(personal_data(Id,_,_)).

delete_patient(Id):-
retractall(personal_data(Id,_,_)).


dbase_command(X,scroll_up):-
X=22,
scroll_window_up(dbase).

dbase_command(X,scroll_down):-
X=50,
scroll_window_down(dbase).


dbase_command(X,add):-
X>23,
X<27,
entry_box(3,56,20,"Enter new patient",Patient),
add_patient(Patient),
draw_dbase_window.

dbase_command(X,delete):-
X>28,
X<35,
entry_box(3,58,6,"Id",Identity),
delete_patient(Identity),
draw_dbase_window.

dbase_command(X,select):-
X>36,
X<43,
get_window_entry(dbase,Entry),
patient_selection(Entry).

dbase_command(X,exit):-
X>44,
X<49,
set_patient.

dbase_command(X,continue).


/* routines to set the current patient */

display_selected_patient(Name,Id):-
window(header,cursor_address(0,41)),
window(header,text("Patient : ")),
fill_out(Name,Name_display,22),
window(header,text(Name_display)),
fill_out(Id,Id_display,6),
window(header,text(Id_display)).


patient_selection(Entry):-
Id_display is_string substring(Entry,21,6),   /* get patient identity */
truncate_string(Id_display,Id),
personal_data(Id,name,Name),
retractall(new_patient/1),
assert(new_patient(Id)),
display_selected_patient(Name,Id).


/* load a patient data file */


load_patient:-                                /* archive file exists */
retractall(current_data/3),
new_patient(Id),
Filename is_string string(Id) & ".ARC",
exists_file(Filename),
reconsult(Filename),
set_current_diseases,
set_personal_data(Id).

load_patient:-                                /* no archive file */
retractall(current_data/3),
new_patient(Id),
set_personal_data(Id).


set_current_diseases:-
archive_data_time(Date,Time),
archive_data(Date,Time,disease,Root,Disease),
assert(current_data(disease,Root,Disease)),
fail.

set_current_diseases:-
remove_disease_window,
display_current_diseases.


/* archive patient data to file */

create_archive_data:-
once(remove_personal_data),
once(get_data_time(Date,Time)),
retract(current_data(Data_type,Facet,Attribute)),
asserta(archive_data(Date,Time,Data_type,Facet,Attribute)), fail.

create_archive_data:-
get_data_time(Date,Time),
retractall(archive_data_time/2),
assert(archive_data_time(Date,Time)).

create_archive_data.


archive_patient(ok):-                                /* save current data */
current_data(demographic,"identity",Id),
create_archive_data,
(delete_file("CURRENT.ARC");true),
create("CURRENT.ARC"),
create_stream(archive,readwrite,ascii,file("CURRENT.ARC")),
open(archive,readwrite),
state(output,_,archive),
write_clauses(archive_data/5),
write_clauses(archive_data_time/2),
close(archive),
delete_stream(archive),
Filename is_string Id & ".ARC",          /* if archive successful ... */
(delete_file(Filename);true),            /* ...overwrite old data */
rename_file("CURRENT.ARC",Filename).

archive_patient(Result).                  /* dont save new data */


/* set the current patient */

set_patient:-                            /* new patient same as old one */
current_data(demographic,"identity",Id),
new_patient(Id).

set_patient:-                            /* old patient must be removed */
current_data(demographic,"identity",Id),
remove_personal_data,
Display is_string "Archiving " & Id,
decision_box(3,55,Display,Result),
archive_patient(Result),
remove_current_data,
load_patient.

set_patient:-                             /* no patient previously */
load_patient.

set_patient.                              /* catch failures */


/* set current data from personal info */

set_personal_data(Id):-
personal_data(Id,name,Name),
personal_data(Id,age,Age),
personal_data(Id,sex,Sex),
personal_data(Id,occupation,Occupation),
assert(current_data(demographic,"identity",Id)),
assert(current_data(variable,"age",Age)),
assert(current_data(history,"sex",Sex)),
assert(current_data(history,"occupation",Occupation)).


/* remove personal info from current data */

remove_personal_data:-
retract(current_data(demographic,"identity",Id)),
retract(current_data(variable,"age",Age)),
retract(current_data(history,"sex",Sex)),
retract(current_data(history,"occupation",Occupation)),
retract(personal_data(Id,age,_)),
retract(personal_data(Id,sex,_)),
```

```
retract(personal_data(Id,occupation,_)),
assert(personal_data(Id,age,Age)),
assert(personal_data(Id,sex,Sex)),
assert(personal_data(Id,occupation,Occupation)).

remove_personal_data.


/* remove current data */

remove_current_data:-
retractall(current_data/3),
retractall(current_data_time/2).



/* archive the patient database */

archive_database:-
(delete_file("PATIENTS.TMP");true),
create("PATIENTS.TMP"),
create_stream(archive,readwrite,ascii,file("PATIENTS.TMP")),
open(archive,readwrite),
state(output,_,archive),
write_clauses(personal_data/3),
close(archive),
delete_stream(archive),
(delete_file("PATIENTS.DAT");true),        /* overwrite old data...... */
rename_file("PATIENTS.TMP","PATIENTS.DAT").  /*..if archive successful*/



/* ---------------------------------------------------------------- */
/* ----------------DISPLAY THE DIAGNOSIS RESULTS-------------------- */

view_diagnosis(Panel):-
define_diagnosis_windows,
draw_diagnosis_backdrop(Panel),
display_diagnoses(Panel),
view_diagnosis_command_loop(Panel),
close_diagnosis_windows.

define_diagnosis_windows:-
create_stream(diagnosis_backdrop,readwrite,byte,window(21,36,black on
red)),
open(diagnosis_backdrop,readwrite),
screen(diagnosis_backdrop,create(2,26,diagnosis_backdrop,0,0,0,lr,black
on red,21,36,hidden)),
create_stream(view_diagnosis,readwrite,byte,window(80,36,red on cyan)),
open(view_diagnosis,readwrite),
screen(view_diagnosis,create(3,26,view_diagnosis,0,0,0,none,None,19,36,h
idden)).

close_diagnosis_windows:-
close(diagnosis_backdrop),
close(view_diagnosis),
delete_stream(diagnosis_backdrop),
delete_stream(view_diagnosis).

write_diagnoses([]):-
window(view_diagnosis,cursor_down).

write_diagnoses([(Diagnosis,Probability)|Diagnosis_list]):-
value_display(Probability,P_display),
fill_out(Diagnosis,D_display,22),
Display is_string "\r\n " & D_display & P_display,
window(view_diagnosis,text(Display)),
write_diagnoses(Diagnosis_list).


display_diagnoses(Panel):-
Root_archive=..[Panel,Root],
call(Root_archive),
once(bagof((Disorder,Probability),diagnosis_exists(Panel,Root,Disorder,P
robability),Diagnosis_list)),
once(write_diagnoses(Diagnosis_list)),
fail.

display_diagnoses(Panel):-
screen(diagnosis_backdrop,unhide),
screen(view_diagnosis,unhide).

draw_diagnosis_backdrop(Panel):-
window(diagnosis_backdrop,cursor_address(0,1)),
window(diagnosis_backdrop,text("DISORDER                BELIEF")),
window(diagnosis_backdrop,cursor_address(20,0)),
window(diagnosis_backdrop,text("\24   Refine Diagnosis        Exit
\25")).


view_diagnosis_command_loop(Panel):-
retractall(cursor_location(view,_,_)),
assert(cursor_location(view,24,26)),
repeat,
once(cursor_location(view,SY,SX)),
once(locator(SY,SX,Y,X,22,26,22,61)),
once(retractall(cursor_location(view,_,_))),
once(assert(cursor_location(view,24,X))),
once(view_diagnosis_action(X,Panel,Action)),
Action=exit.


view_diagnosis_action(26,Panel,continue):-
scroll_window_up(view_diagnosis).

view_diagnosis_action(61,Panel,continue):-
scroll_window_down(view_diagnosis).

view_diagnosis_action(X,Panel,continue):-
X>29,
X<46,
refine_diagnosis_input,
make_diagnosis(disorder),
window(view_diagnosis,clear),
display_diagnoses(Panel).

view_diagnosis_action(X,Panel,exit):-
X>53,
X<58.


/* ---------------------------------------------------------------- */
/* ---------------------- INPUT PATIENT DISEASE -------------------- */

patient_diseases:-
define_disease_window,
draw_disease_window,
define_disease_foot,
disease_command_loop.


/* initial set up of disease display */

display_current_diseases:-
number_of_clauses(current_data/3,current_data(disease,_,_),N),
N>0,
define_disease_window,
draw_disease_window.

display_current_diseases.


/* display the command line */

define_disease_foot:-
create_stream(disease_foot,readwrite,byte,window(1,22,white on red)),
open(disease_foot,readwrite),
screen(disease_foot,create(15,1,disease_foot,0,0,0,none,None,1,22,hidden
)).

draw_disease_foot(Drop):-
Y is Drop+15,
screen(disease_foot,change(Y,1,disease_foot,0,0,0,none,None,1,22,hidden)
), window(disease_foot,cursor_address(0,0)),
window(disease_foot,text(" \24  Add  Del  Exit  \25 ")),
screen(disease_foot,unhide).


/* define the window for disease display */

define_disease_window:-
stream(disease_display,A,B,D,S).

define_disease_window:-
create_stream(disease_display,readwrite,byte,window(80,22,white on
black)),
open(disease_display,readwrite),
screen(disease_display,create(16,1,disease_display,0,0,0,all,black on
red,8,22,hidden)),
create_stream(disease_header,readwrite,byte,window(1,22,black on red)),
open(disease_header,readwrite),
screen(disease_header,create(15,1,disease_header,0,0,0,none,black on
red,1,22,hidden)), window(disease_header,text(" CURRENT DISEASES
\205\205\205\205")).

/* remove the window for disease display */

remove_disease_window:-
stream(disease_display,A,B,D,S),
close(disease_display),
delete_stream(disease_display),
close(disease_header),
delete_stream(disease_header).

remove_disease_window.


/* displaying the current diseases */

display_diseases:-
once(window(disease_display,clear)),
once(window(disease_display,cursor_address(0,1))),
current_data(disease,Root,Disease),
once(write_window_entry(disease_display,Disease)),
fail.

display_diseases.


/* draw the window for current diseases */

draw_disease_window:-
number_of_clauses(current_data/3,current_data(disease,_,_),N),
((N<8,Drop is N+1);(Drop is 8)),
retractall(window_drop(disease_display,_)),
assert(window_drop(disease_display,Drop)),
screen(disease_display,change(16,1,disease_display,0,0,0,all,black on
red,Drop,22,hidden)),
display_diseases,
scroll_to_foot(disease_display),
screen(disease_display,unhide),
screen(disease_header,pull_up),
screen(disease_header,unhide).


/* routines for display disease commands */

disease_command_loop:-
retractall(cursor_location(disease_foot,_,_)),
assert(cursor_location(disease_foot,0,18)),
repeat,
once(window_drop(disease_display,Drop)),
once(CY is Drop+15),
once(draw_disease_foot(Drop)),
once(cursor_location(disease_foot,_,SX)),
once(locator(CY,SX,Y,X,CY,2,CY,21)),
once(retract(cursor_location(disease_foot,_,_))),
once(assert(cursor_location(disease_foot,0,X))),
once(disease_display_command(X,Result)),
Result=exit.

/* adding a new disease */
```

```
add_disease(Root,"EXIT").

add_disease(Root,Disease):-
check_diseases(Root,Disease).

add_disease(Root,Disease):-
assert(current_data(disease,Root,Disease)),
assert(current_data(history,Disease,"present")).

/* check the input disease */

check_diseases(Root,Disease):-          /* fails if ok */
current_data(disease,Root,D),
related_to(disease,Root,D,Disease),!,
warning_box(8,30,"CANNOT ADD THIS").

check_diseases(Root,Disease):-          /* fails if ok */
current_data(disease,Root,Disease),!,
warning_box(8,30,"CANNOT ADD THIS").


disease_display_command(X,scroll_up):-
X=2,
scroll_window_up(disease_display).

disease_display_command(X,scroll_down):-
X=21,
scroll_window_down(disease_display).

disease_display_command(X,add):-
X>4,
X<8,
disease_selection(Root,Disease),
add_disease(Root,Disease),
draw_disease_window.

disease_display_command(X,delete):-
X>9,
X<13,
entry_box(16,26,24,"Delete disease",Disease),
retractall(current_data(disease,Root,Disease)),
retractall(current_data(history,Disease,_)),
draw_disease_window.

disease_display_command(X,exit):-
X>14,
X<19,
window_drop(disease_display,1),          /* If no current diseases */
remove_disease_window,
close(disease_foot),
delete_stream(disease_foot).

disease_display_command(X,exit):-
X>14,
X<19,
close(disease_foot),
delete_stream(disease_foot).


disease_display_command(X,continue).

/* ----------------------------------------------------------------*/
/* DISEASE SELECTION ROUTINES */

/* input of current disease */

disease_selection(Root,Disease):-
once(define_selection_window),
once(define_selection_command_window),
once(draw_selection_header),
once(draw_selection_window(roots)),
once(selection_command_loop(roots,Root)),
once(draw_selection_window(Root)),
once(selection_command_loop(Root,Disease)),
close_selection_windows.

disease_selection(Root,"EXIT"):-                /* catches failures */
close_selection_windows,
warning_box(8,30,"Cannot add disease").


/* Define the selection window */

define_selection_window:-
create_stream(selection_background,readwrite,byte,window(22,55,white on
black)),
open(selection_background,readwrite),
screen(selection_background,create(2,25,selection_background,0,0,0,none,
None,22,55,revealed)),
create_stream(selection,readwrite,byte,window(60,24,white on black)),
open(selection,readwrite),
screen(selection,create(1,1,selection,0,0,0,lrb,black on
red,23,24,hidden)).


/* Define the command line */

define_selection_command_window:-
create_stream(selection_command,readwrite,byte,window(1,24,white on
red)), open(selection_command,readwrite),
screen(selection_command,create(0,0,selection_command,0,0,0,none,None,1,
24,hidden)),
window(selection_command,text("\24          EXIT          \25")).


/* Close selection windows */

close_selection_windows:-
close(selection_command),
delete_stream(selection_command),
close(selection_header),
delete_stream(selection_header),
close(selection),
delete_stream(selection),
close(selection_background),
delete_stream(selection_background).


/* Draw the header */

draw_selection_header:-
create_stream(selection_header,readwrite,byte,window(1,24,white on
red)), open(selection_header,readwrite),
screen(selection_header,create(3,30,selection_header,0,0,0,lrt,black on
red,1,24,hidden)),
window(selection_header,cursor_address(0,1)),
window(selection_header,text("INPUT DISEASE")),
screen(selection_header,unhide).


/* Draw the selection window */

draw_selection_window("EXIT").

draw_selection_window(Selector):-
form_selection_list(Selector,List),
length(List,N),
((N<17,Drop is N+1);(Drop is 19)),
retractall(window_drop(selection,_)),
assert(window_drop(selection,Drop)),
screen(selection,chang
e(4,30,selection,0,0,0,lrb,black on red,Drop,24,hidden)),
window(selection,clear),
window(selection,cursor_address(0,1)),
display_selection_list(List),
screen(selection,unhide).


/* Form the list for display */

form_selection_list(roots,List):-!,
bagof(Root,disease(Root),List).

form_selection_list(Root,List):-
bagof(D,descendent(disease,Root,Root,D),List).


/* Display the list of diseases */

display_selection_list([]).

display_selection_list([Entry|List]):-
write_window_entry(selection,Entry),
display_selection_list(List).


/* The loop for selecting a disease */

selection_command_loop("EXIT","EXIT").

selection_command_loop(Root,Selection):-
window_drop(selection,Drop),
BY is Drop+3,
screen(selection_command,change(BY,30,selection_command,0,0,0,none,None,
1,24,hidden)),
screen(selection_command,unhide),
retractall(cursor_location(selection,_,_)),
assert(cursor_location(selection,BY,30)),
repeat,
once(cursor_location(selection,SY,SX)),
once(locator(SY,SX,Y,X,4,30,BY,53)),
once(WY is Y-4),
once(WX is X-30),
once(retractall(cursor_location(selection,_,_))),
once(assert(cursor_location(selection,Y,X))),
once(selection_action(WY,WX,Result)),
Result=exit,
screen(selection,info(_,_,selection,OY,_,_,_,_,_,_,_)),  /*scroll info*/
AY is WY+OY,                                             /* absolute Y pos */
window(selection,cursor_address(AY,1)),
window(selection,inquire_text(22,Entry)),
truncate_string(Entry,Selection).


selection_action(WY,WX,scroll_up):-
window_drop(selection,Drop),
WY is Drop-1,
WX=0,
scroll_window_up(selection).

selection_action(WY,WX,scroll_down):-
window_drop(selection,Drop),
WY is Drop-1,
WX=23,
scroll_window_down(selection).

selection_action(WY,WX,exit):-
window_drop(selection,Drop),
WY is Drop-1,
WX>9,
WX<14.

selection_action(WY,WX,exit):-
window_drop(selection,Drop),
WY<Drop-1.

selection_action(WY,WX,continue).
```

```
/* das.dia dialogue control */

/* Predicates defined in this file:

dialogue/0
set_dialogue_context/1
store_data/0
restore_data/0
define_dialogue_window/0
define_query_window/0
prepare_for_output/0
enter_dialogue/1
getsentence/1
getrest/2
getletters/3
in_word/2
upcase/2
string_to_atoms/2
form_atom/2
list_to_atoms/3
convert_string_list/2
capitalize/2
standardize_output/2
replace_synonym/2
synonym_combination/2
find_synonym/2
display_query/2
display_dialogue_output/0
dialogue_text/1
capitalize_output/0
output_text/1
output_spaceless_text/1
output_list/2
output_patient_name/0
output_personal_pronoun/0
get_person/2
output_diagnoses/2
output_diagnosis_expression/1
create_diagnosis_list/2
find_diagnosed_roots/2
find_undiagnosed_roots/3
produce_diagnosis_output/4
generate_diagnosis_output/5
generate_diagnosis_output/2
output_critique/2
find_consistent_diagnoses/2
form_consistent_list/3
find_inconsistent_diagnoses/2
form_inconsistent_list/3
output_consistencies/2
output_inconsistencies/3
basic_command/3
information_request/7
supposition/5
explanation_request/4
explanation_request/5
impact_request/5
impact_request/6
process_input/2
exit/0
misunderstood/0
control/1
output_diagnosis_explanation/2
output_specific_explanation/3
check_justification/4
counter_justification/2
output_justification_violation/2
produce_diagnosis_explanation/3
justification_introduction/2
justify_diagnosis/5
generate_justification_output/6
sum_evidence_effect/7
get_evidence_weight/6
output_justification/7
output_relation_description/3
relevant_evidence/5
indicate_support_strength/1
output_evidence_impact/3
output_evidence_impact/4
evidence_impact/5
variable_impact/4
relation_impact/4
total_relation_impact/4
evidence_impact_introduction/2
generate_evidence_impact_output/1
impact_description/2
make_supposition/3
make_supposition_diagnosis/3
pose_context_question/1
set_supposition/3
output_information/5
retrieve_current_data/4
variable_level_description/3            */

/* ----------------------ACCEPT USER INPUT------------------------- */
/* ---------------------------------------------------------------- */

/* kick off */

dialogue:-
define_dialogue_window,
define_query_window,
store_data,
prepare_for_output,
output_diagnoses(disorder,present),
assert(output(new_paragraph)),
output_critique(disorder,present),
display_dialogue_output,
repeat,
once(enter_dialogue(Input)),
once(process_input(Input,Action_goal)),
once(display_query(Input,Action_goal)),
once(call(Action_goal)),
once(display_dialogue_output),
Action_goal=exit,
restore_data,
close(dialogue),
delete_stream(dialogue),
close(query),
delete_stream(query).


dialogue:-
restore_data,
close(dialogue),
delete_stream(dialogue),
close(query),
delete_stream(query).


/* Setting the context for the dialogue */

set_dialogue_context(Context):-
retractall(dialogue_context/1),
assert(dialogue_context(Context)).

/* Store the current data before a dialogue session */

store_data:-
once(retractall(stored_data/3)),
current_data(Type,Entity,Value),
assert(stored_data(Type,Entity,Value)),
fail.

store_data:-
set_dialogue_context(diagnosis(disorder)).


/* Restore current data from stored data */

restore_data:-
dialogue_context(diagnosis(disorder)).

restore_data:-
once(retractall(current_data/3)),
retract(stored_data(Type,Entity,Value)),
assert(current_data(Type,Entity,Value)),
fail.

restore_data:-
set_dialogue_context(diagnosis(disorder)).


/* ---------------------------------------------------------------- */
/* ------------- DIALOGUE SCREEN & I/O OPERATIONS ----------------- */

/* define screen */

define_dialogue_window:-
create_stream(dialogue,readwrite,byte,window(80,53,black on red)),
open(dialogue,readwrite),
screen(dialogue,create(3,26,dialogue,30,0,0,lrt,black on
red,16,53,revealed)), window(dialogue,clear),
window(dialogue,cursor_address(63,0)).

define_query_window:-
create_stream(query,readwrite,byte,window(3,53,white on red)),
open(query,readwrite),
screen(query,create(19,26,query,0,0,0,lrb,black on red,3,53,revealed)),
window(query,clear).


/* prepare for dialogue output */

prepare_for_output:-
retractall(output/1),
assert(output(new_paragraph)),
screen(dialogue,change(3,26,dialogue,64,0,0,lrt,black on
red,16,53,revealed)).


/* type in dialogue */

enter_dialogue(List):-
window(query,clear),
seeing(Stream),
see(query),
getsentence(List),
see(Stream),
retractall(output/1).


/* construct a list of atoms from user input */

getsentence(Wordlist):-
get0(C),
getrest(C,Wordlist).

getrest(L,[Word|Wordlist]):-
in_word(L,Lc),
getletters(L,Letters,C),
list(Letters,S),
name(W,S),
replace_synonym(W,Word),
getrest(C,Wordlist).

getrest(13,[]).

getrest(L,Wordlist):-
getsentence(Wordlist).


getletters(L,[Lc|Letters],Last_letter):-
in_word(L,Lc),
get0(N1),
getletters(N1,Letters,Last_letter).

getletters(13,[],13).

getletters(L,[],L).

/* characters that can appear in a word */
```

```
in_word(C,C) :- C>96, C<123.                    /* a b ..... z */
in_word(C,L) :- C>64, C<91, L is C+32.          /* A B ..... Z */
in_word(C,C) :- C>47, C<58.                     /* 0 1 ......9 */
in_word(39,39).                                 /* ' */
in_word(45,45).                                 /* - */
in_word(38,38).                                 /* & */
in_word(46,46).                                 /* . must have dp */


/* convert characters to upper case */

upcase(C,U) :- C>96, C<123, U is C-32.          /* a b ..... z */
upcase(C,C).                                   /* everything else */



/* some list utilities */

string_to_atoms(String,Atom_list) :-
list(List,String),
list_to_atoms(List,"",Atom_list).


form_atom(Input,Atom) :-
name(Atom1,Input),
replace_synonym(Atom1,Atom).


form_atom(Input,Atom) :-
Atom is_string string(Input,ops).

list_to_atoms([],String,[Atom]) :-
form_atom(String,Atom).

list_to_atoms([32|List],String,[Atom|Atom_list]) :-
form_atom(String,Atom),
list_to_atoms(List,"",Atom_list).

list_to_atoms([C|List],String,Atom_list) :-
in_word(C,L),
New_string is_string String & [L],
list_to_atoms(List,New_string,Atom_list).


/* Convert list of strings to list of lists */

convert_string_list([],[]).

convert_string_list([String|String_list],[Atom_list|List]) :-
string_to_atoms(String,Atom_list),
convert_string_list(String_list,List).


/* puts capital letter on string */

capitalize(Text,New_text) :-
S is_string substring(Text,0,1),
list([C],S),
upcase(C,U),
New_text is_string [U] & delete(Text,0,1).


/* standardize the output */

standardize_output([],[]).

standardize_output([Atom|List],[New_atom|New_list]) :-
replace_synonym(Atom,New_atom),
standardize_output(List,New_list).


/* Replace an atom by synonym */

replace_synonym(Atom,New_atom) :-
synonym(New_atom,Atom).

replace_synonym(Atom,Atom).


/* find a combination of synonyms */

synonym_combination([],[]).

synonym_combination([Atom|List],[New_atom|New_list]) :-
find_synonym(Atom,New_atom),
synonym_combination(List,New_list).


/* Replace an atom by synonym */

find_synonym(Atom,New_atom) :-
synonym(Atom,New_atom).

find_synonym(Atom,Atom).


/* ------------------------------------------------------------------ */
/* ---------------------- DIALOGUE OUTPUT ---------------------------- */


/* display the query in the dialogue window */

display_query(Input,exit).

display_query(Input,control(_)).

display_query(Input,Result) :-
prepare_for_output,
assert(output(new_paragraph)),
assert(output(Input)),
assert(output(query)),
assert(output(new_line)),
assert(output(new_paragraph)).

/* display all output */

display_dialogue_output :-
retract(output(Text)),
dialogue_text(Text),!,
display_dialogue_output.

display_dialogue_output.


/* output text to dialogue window */

dialogue_text(new_paragraph) :-            /* remove any new sentences */
retract(output/1,output(new_sentence),1),
dialogue_text(new_paragraph).

dialogue_text(new_paragraph) :-
capitalize_output,
window(dialogue,text("\r\n  ")).

dialogue_text(new_sentence) :-       /* remove any other new sentences */
retract(output/1,output(new_sentence),1),
dialogue_text(new_sentence).

dialogue_text(new_sentence) :-
capitalize_output,
window(dialogue,text(". ")).

dialogue_text(end_text) :-
window(dialogue,text(".\r\n")).

dialogue_text(new_line) :-
window(dialogue,text("\r\n")).

dialogue_text(comma) :-
window(dialogue,text(",")).

dialogue_text(colon) :-
window(dialogue,text(":")).

dialogue_text(semi_colon) :-
window(dialogue,text(";")).

dialogue_text(query) :-
window(dialogue,text("?")).

dialogue_text(space) :-
window(dialogue,inquire_cursor_address(_,0)).

dialogue_text(space) :-
window(dialogue,text(" ")).

dialogue_text(space(N)) :-
repeat(X),
dialogue_text(space),
X is N-1.

dialogue_text(list(Type,List)) :-
output_list(Type,List).

dialogue_text(spaceless(List)) :-
output_spaceless_text(List).

dialogue_text(List) :-
output_text(List).


/* Capitalize output */

capitalize_output :-
retract(output/1,output([First_atom|List]),1),
First_word is_string string(First_atom,ops),
capitalize(First_word,Text),
asserta(output([First_word|List])).

capitalize_output :-
retract(output/1,
output(list(Type,[First_atom|List])),1),
First_word is_string string(First_atom,ops),
capitalize(First_word,Text),
asserta(output(list(Type,[First_word|List]))).

capitalize_output.


/* outputs a simple list of atoms with spaces between each atom */

output_text([]).

output_text([Atom|List]) :-
Text is_string string(Atom,ops),
dialogue_text(space),
window(dialogue,text(Text)),
output_text(List).


/* outputs a simple list of atoms without spaces */

output_spaceless_text([]).

output_spaceless_text([Atom|List]) :-
Text is_string string(Atom,ops),
window(dialogue,text(Text)),
output_spaceless_text(List).


/* outputs a compound list of atoms */

output_list(Type,[Element]) :-
output_text(Element).

output_list(Type,[Element1,Element2]) :-
output_text(Element1),
output_text([Type|Element2]).

output_list(Type,[Element|List]) :-
output_text(Element),
dialogue_text(comma),
```

```
output_list(Type,List).


/* generate output for patient */

output_patient_name:-
current_data(demographic,identity,Id),
patient_data(Id,name,Name),
string_to_atoms(Name,Atom_list),
assert(output(Atom_list)).

output_patient_name:-
assert(output([this,patient])).


/* generate he or she */

output_personal_pronoun:-
current_data(history,sex,"male"),
assert(output([he])).

output_personal_pronoun:-
assert(output([she])).

/* get plurality of list */

get_person([A],2).        /* singular */

get_person(L,3).          /* plural */


/* ---------------------------------------------------------------- */
/* ------------------------ DIAGNOSIS OUTPUT ------------------------ */

/* output the diagnosis */

output_diagnoses(Blackboard,Tense):-
find_diagnosed_roots(Blackboard,Diagnosed_list),
find_undiagnosed_roots(Blackboard,Undiagnosed_list,Diagnosed_list),
produce_diagnosis_output(Blackboard,Diagnosed_list,Tense,start),
generate_diagnosis_output(Undiagnosed_list,Tense),
assert(output(new_sentence)),
assert(output(new_paragraph)).

/* output list of diagnoses */

output_diagnosis_expression([(Disorder,P)]):-
string_to_atoms(Disorder,Disorder_output),
assert(output(Disorder_output)).

output_diagnosis_expression(Disorder_list):-
create_diagnosis_list(Disorder_list,Disorder_output),
assert(output(list(or,Disorder_output))).


/* create a list of atomized diagnoses */

create_diagnosis_list([],[]).

create_diagnosis_list([(Disorder,P)|List],[Disorder_atoms|New_list]):-
string_to_atoms(Disorder,Disorder_atoms),
create_diagnosis_list(List,New_list).


/* find list of diagnosed/undiagnosed roots */

find_diagnosed_roots(Blackboard,Root_list):-
setof(Root,(D,P)^diagnosis_exists(Blackboard,Root,D,P),Root_list).

find_diagnosed_roots(Blackboard,[]).

find_undiagnosed_roots(Blackboard,Root_list,Diagnosed_list):-
setof([Root],no_diagnosis_exists(Blackboard,Root,Diagnosed_list),Root_li
st).

find_undiagnosed_roots(Blackboard,[],Diagnosed_list).


produce_diagnosis_output(Blackboard,[],Tense,Context).

produce_diagnosis_output(Blackboard,[Root|List],Tense,Context):-
bagof((Disorder,P),diagnosis_exists(Blackboard,Root,Disorder,P),Diagnosi
s_list),
generate_diagnosis_output(Root,Diagnosis_list,Tense,Context,New_context)
, produce_diagnosis_output(Blackboard,List,Tense,New_context).

produce_diagnosis_output(Blackboard,[Root|List],Tense,Context):-
generate_diagnosis_output(Root,[],Tense,Context,New_context),
produce_diagnosis_output(Blackboard,List,Tense,New_context).


/* type 1 */

generate_diagnosis_output(Root,[(Disorder,P)|_],Tense,start,1):-
P>0.95,
assert(output([my,diagnosis,for])),
output_patient_name,
verb(be,Tense,2,V,[]),
assert(output(V)),
output_diagnosis_expression([(Disorder,P)]).

generate_diagnosis_output(Root,[(Disorder,P)],Tense,Context,new):-
P>0.95,
member(Context,[1,2,3]),
assert(output([with])),
output_diagnosis_expression([(Disorder,P)]).

generate_diagnosis_output(Root,[(Disorder,P)],Tense,Context,1):-
P>0.95,
assert(output(new_sentence)),
output_personal_pronoun,
verb(have,Tense,2,V,[]),
assert(output(V)),
output_diagnosis_expression([(Disorder,P)]).


/* type 2 */

generate_diagnosis_output(Root,[(Disorder,P)|_],Tense,start,2):-
P>0.7,
assert(output([the,most,likely,diagnosis,for])),
output_patient_name,
verb(be,Tense,2,V,[]),
assert(output(V)),
output_diagnosis_expression([(Disorder,P)]).

generate_diagnosis_output(Root,[(Disorder,P)|_],Tense,1,new):-
P>0.7,
assert(output([and,probably])),
output_diagnosis_expression([(Disorder,P)]).

generate_diagnosis_output(Root,[(Disorder,P)|_],Tense,2,new):-
P>0.7,
assert(output([and])),
output_diagnosis_expression([(Disorder,P)]).

generate_diagnosis_output(Root,[(Disorder,P)|_],Tense,Context,2):-
P>0.7,
assert(output(new_sentence)),
output_personal_pronoun,
assert(output([probably])),
verb(have,Tense,2,V,[]),
assert(output(V)),
output_diagnosis_expression([(Disorder,P)]).


/* type 3 */

generate_diagnosis_output(Root,[(Disorder1,P1),(Disorder2,P2)|_],Tense,s
tart,3):-
P1>0.35,
P2>0.35,
assert(output([my,diagnosis,for])),
output_patient_name,
verb(be,Tense,2,V,[]),
assert(output(V)),
assert(output([either])),
output_diagnosis_expression([(Disorder1,P1),(Disorder2,P2)]).

generate_diagnosis_output(Root,[(Disorder1,P1),(Disorder2,P2)|_],Tense,C
ontext,new):-
P1>0.35,
P2>0.35,
member(Context,[1,2,3]),
assert(output([and,either])),
output_diagnosis_expression([(Disorder1,P1),(Disorder2,P2)]).

generate_diagnosis_output(Root,[(Disorder1,P1),(Disorder2,P2)|_],Tense,C
ontext,3):-
P1>0.35,
P2>0.35,
assert(output(new_sentence)),
output_personal_pronoun,
verb(have,Tense,2,V,[]),
assert(output(V)),
assert(output([either])),
output_diagnosis_expression([(Disorder1,P1),(Disorder2,P2)]).


/* Type 4 */

generate_diagnosis_output(Root,[(Disorder,P)|Disorder_list],Tense,start,
new):-
P>0.35,
assert(output(new_sentence)),
output_patient_name,
assert(output([probably])),
verb(have,Tense,2,V,[]),
assert(output(V)),
output_diagnosis_expression([(Disorder,P)]),
assert(output(comma)),
assert(output([possibly])),
output_diagnosis_expression(Disorder_list).

generate_diagnosis_output(Root,[(Disorder,P)|Disorder_list],Tense,Contex
t,new):-
P>0.35,
assert(output(new_sentence)),
output_personal_pronoun,
assert(output([probably])),
verb(have,Tense,2,V,[]),
assert(output(V)),
output_diagnosis_expression([(Disorder,P)]),
assert(output(comma)),
assert(output([possibly])),
output_diagnosis_expression(Disorder_list).


/* type 5 */

generate_diagnosis_output([],Tense).

generate_diagnosis_output(Root_list,Tense):-
assert(output(new_sentence)),
assert(output([i])),
verb(be,Tense,1,V,[]),
assert(output(V)),
assert(output([unable,to,diagnose])),
assert(output(list(or,Root_list))),
assert(output([using,the,available,information])).


/* ---------------------------------------------------------------- */
/* ------------------------ CRITIQUE OUTPUT ------------------------ */

/* Main routine to output the critique */

output_critique(Blackboard,Tense):-
bagof(Disease,R^blackboard(Blackboard,diagnosis,R,Disease),Disease_list)
, find_consistent_diagnoses(Blackboard,Consistent_diagnoses),
find_inconsistent_diagnoses(Blackboard,Inconsistent_diagnoses),
output_consistencies(Tense,Consistent_diagnoses),
```

```
output_inconsistencies(Tense,Disease_list,Inconsistent_diagnoses).

output_critique(Blackboard,Tense).


/* Find the consistent diagnoses */

find_consistent_diagnoses(Blackboard,List):-
setof(Disease,(R,D)^blackboard(Blackboard,critique,R,Disease,D,consisten
t),Disease_list),
form_consistent_list(Blackboard,Disease_list,List).

find_consistent_diagnoses(Blackboard,[]).


form_consistent_list(Blackboard,[],[]).

form_consistent_list(Blackboard,[Disease|Disease_list],[(Disease,Disorde
r_list)|List]):-
bagof(Disorder,R^blackboard(Blackboard,critique,R,Disease,Disorder,consi
stent),Disorder_list),
form_consistent_list(Blackboard,Disease_list,List).


/* Find the inconsistent diagnoses */

find_inconsistent_diagnoses(Blackboard,List):-
setof(Root,(Disease,Disorder)^blackboard(Blackboard,critique,Root,Diseas
e,Disorder,expected),Root_list),
form_inconsistent_list(Blackboard,Root_list,List).

find_inconsistent_diagnoses(Blackboard,[]).


form_inconsistent_list(Blackboard,[],[]).

form_inconsistent_list(Blackboard,[Root|Root_list],[(Root,Disorder_list)
|List]):-
setof(Disorder,Disease^blackboard(Blackboard,critique,Root,Disease,Disor
der,expected),Disorder_list),
form_inconsistent_list(Blackboard,Root_list,List).


/* output for the consistent diagnoses */

output_consistencies(Tense,[]).

output_consistencies(Tense,[(Disease,Disorder_list)|List]):-
get_person(Disorder_list,P),
convert_string_list(Disorder_list,Disorders),
assert(output(list(and,Disorders))),
verb(be,Tense,P,V,[]),
assert(output(V)),
assert(output([consistent,with])),
string_to_atoms(Disease,D),
assert(output(D)),
assert(output(new_sentence)),
output_consistencies(Tense,List).


/* output for the inconsistent diagnoses */

output_inconsistencies(Tense,Disease_list,[]).

output_inconsistencies(Tense,Disease_list,[(Root,Disorder_list)|List]):-
assert(output([my,diagnosis,for])),
string_to_atoms(Root,R),
assert(output(R)),
verb(be,Tense,2,V,[]),
assert(output(V)),
assert(output([not,consistent,with])),
convert_string_list(Disease_list,Diseases),
assert(output(list(or,Diseases))),
assert(output(semi_colon)),
assert(output(["I",would,expect,to,see])),
convert_string_list(Disorder_list,Disorders),
assert(output(list(or,Disorders))),
assert(output(new_sentence)),
output_inconsistencies(Tense,Disease_list,List).


/* ------------------------------------------------------------------ */
/* -------------- IDENTIFY ACTIONS FROM INPUT SENTENCES ------------ */

/* basic commands */

basic_command(exit) --> verb(stop,present,1).
basic_command(scroll_up) --> [scroll,up].
basic_command(scroll_down) --> [scroll,down].
basic_command(scroll_down) --> [scroll].


/* information requests */

information_request(value,diagnosis,Type,the,past) -->
[original,diagnosis].
information_request(value,diagnosis,Type,the,past) --> [diagnosis].

information_request(value,N,Type,the,Tense) -->
interrogative_phrase(instantiate,Tense),noun_phrase(N,Type).
information_request(value,N,Type,P,Tense) -->
interrogative_phrase(instantiate,Tense),possessive(P),noun(N,Type).
information_request(value,N,Type,the,Tense) -->
interrogative_phrase(instantiate,Tense),[the,value,of],noun_phrase(N,Typ
e).
information_request(value,N,Type,P,Tense) -->
interrogative_phrase(instantiate,Tense),[the,value,of],possessive(P),nou
n_phrase(N,Type).

information_request(level,N,Type,the,Tense) -->
interrogative_phrase(instantiate,Tense),[the,level,of],noun_phrase(N,Typ
e).
information_request(level,N,Type,the,Tense) -->
interrogative_phrase(instantiate,Tense),[the,level,of],possessive(P),nou
n_phrase(N,Type).

/* suppositions */

supposition(variable,N,V) -->
supposition_phrase,noun_phrase(N,variable),verb(be,Tense,2),num(V).
supposition(variable,N,V) -->
supposition_phrase,noun_phrase(N,variable),verb(be,Tense,2),num(V),[U],
{(data_parameter(N,units,Units),string_to_atoms(Units,T),!,T=U)}.

supposition(Type,N,V) -->
supposition_phrase,noun_phrase(N,Type),verb(be,Tense,2),noun(V,attribute
(Type,N)). supposition(variable,N,L) -->
supposition_phrase,noun_phrase(N,variable),verb(be,Tense,2),adjective(L,
level).


/* explanation requests */

explanation_request(diagnosis,Blackboard) -->
interrogative(reason),{dialogue_context(diagnosis(Blackboard))}.
explanation_request(diagnosis,Blackboard,Disorder) -->
interrogative(reason),noun_phrase(Disorder,disorder),{dialogue_context(d
iagnosis(Blackboard))}.
explanation_request(diagnosis,Blackboard,Disorder) -->
interrogative(reason),[did,you,diagnose],noun_phrase(Disorder,disorder),
{dialogue_context(diagnosis(Blackboard))}.
explanation_request(diagnosis,Blackboard,Disorder) -->
interrogative_phrase(reason,Tense),noun_phrase(Disorder,disorder),[diagn
osed],{dialogue_context(diagnosis(Blackboard))}.
explanation_request(non-diagnosis,Blackboard,Disorder) -->
interrogative(reason),[not],noun_phrase(Disorder,disorder),{dialogue_con
text(diagnosis(Blackboard))}.
explanation_request(non-diagnosis,Blackboard,Disorder) -->
interrogative(reason),['didn''t',you,diagnose],noun_phrase(Disorder,diso
rder),{dialogue_context(diagnosis(Blackboard))}.
explanation_request(non-diagnosis,Blackboard,Disorder) -->
interrogative(reason),['wasn''t'],noun_phrase(Disorder,disorder),[diagno
sed],{dialogue_context(diagnosis(Blackboard))}.
explanation_request(non-diagnosis,Blackboard,Disorder) -->
interrogative_phrase(reason,Tense),noun_phrase(Disorder,disorder),[not,d
iagnosed],{dialogue_context(diagnosis(Blackboard))}.


/* request for impact of evidence */

impact_request(Entity,Type,Tense) -->
interrogative(process),verb(do,Tense,2),noun_phrase(Entity,Type),[affect
,the,diagnosis].
impact_request(Entity,Type,Tense) -->
interrogative(process),verb(do,Tense,2),noun_phrase(Entity,Type),[affect
,your,diagnosis].
impact_request(Entity,Type,Tense) -->
interrogative(instantiate),verb(be,Tense,2),noun_phrase(_,influence),[of
],noun_phrase(Entity,Type).

impact_request(Entity,Type,Disorder,Tense) -->
interrogative(process),verb(do,Tense,2),noun_phrase(Entity,Type),[affect
],noun_phrase(Disorder,disorder).
impact_request(Entity,Type,Disorder,Tense) -->
interrogative(process),verb(do,Tense,2),noun_phrase(Entity,Type),[affect
,your,diagnosis,of],noun_phrase(Disorder,disorder).
impact_request(Entity,Type,Disorder,Tense) -->
interrogative(process),verb(do,Tense,2),noun_phrase(Entity,Type),[affect
,the,diagnosis,of],noun_phrase(Disorder,disorder).
impact_request(Entity,Type,Disorder,Tense) -->
interrogative(process),verb(be,Tense,2),noun_phrase(Disorder,disorder),[
affected,by],noun_phrase(Entity,Type).


/* Process user input */

/* takes a users input sentence, parses it, and decides action goal */

process_input(Input,exit):-
basic_command(exit,Input,[]).

process_input(Input,control(scroll_down)):-
basic_command(scroll_down,Input,[]).

process_input(Input,control(scroll_up)):-
basic_command(scroll_up,Input,[]).

process_input(Input,output_diagnosis_explanation(Blackboard,indicate)):-
explanation_request(diagnosis,Blackboard,Input,[]),!.

process_input(Input,output_specific_explanation(Blackboard,Disorder,indi
cate)):-
explanation_request(diagnosis,Blackboard,Disorder,Input,[]),!.

process_input(Input,output_specific_explanation(Blackboard,Disorder,cont
ra-indicate)):-
explanation_request(non-diagnosis,Blackboard,Disorder,Input,[]),!.

process_input(Input,make_supposition(Attribute,Entity,Value)):-
supposition(Attribute,Entity,Value,Input,[]),!.

process_input(Input,output_evidence_impact(Entity,Type,Tense)):-
impact_request(Entity,Type,Tense,Input,[]),!.

process_input(Input,output_evidence_impact(Entity,Type,Disorder,Tense)):
- impact_request(Entity,Type,Disorder,Tense,Input,[]),!.

process_input(Input,output_information(Information,Entity,Type,P,Tense))
:- information_request(Information,Entity,Type,P,Tense,Input,[]),!.

process_input(Input,misunderstood).


/* ------------------------------------------------------------------ */
/* ------------------------- BASIC ACTIONS ------------------------- */

/* finish dialogue */

exit:-retractall(dialogue_context/1).


/* Didn't understand the query */
```

```
misunderstood:-
assert(output([i,"don't",understand])),
assert(output(end_text)).

/* control functions */

control(scroll_down):-
repeat(X),
scroll_window_down(dialogue),
X=7.

control(scroll_up):-
repeat(X),
scroll_window_up(dialogue),
X=7.



/* ------------------------------------------------------------------ */
/* ------------------------ EXPLANATION OUTPUT ---------------------- */

/* output an explanation for the diagnosis */

output_diagnosis_explanation(Blackboard,Justification):-
setof((Root,Diagnosis),P^diagnosis_exists(Blackboard,Root,Diagnosis,P),D
iagnosis_list),
produce_diagnosis_explanation(Blackboard,Justification,Diagnosis_list),
assert(output(end_text)),
retractall(dialogue_context/1),
assert(dialogue_context(diagnosis(Blackboard))).

/* output an explanation for a specific diagnosis */

output_specific_explanation(Blackboard,Disorder,Justification):-
link(Root,_,Disorder,_),
check_justification(Blackboard,Disorder,Justification,Checked_justificat
ion),
produce_diagnosis_explanation(Blackboard,Checked_justification,[(Root,Di
sorder)]).


/* Check that the correct justification is being sought for a diagnosis
*/

check_justification(Blackboard,Disorder,indicate,indicate):-
diagnosis_exists(Blackboard,Root,Disorder,P).

check_justification(Blackboard,Disorder,contra-indicate,contra-
indicate):- not diagnosis_exists(Blackboard,Root,Disorder,P).

check_justification(Blackboard,Disorder,Justification,Counter_justificat
ion):-
counter_justification(Justification,Counter_justification),
output_justification_violation(Disorder,Justification).


/* Get the opposite justification */

counter_justification(indicate,contra-indicate).
counter_justification(contra-indicate,indicate).


/* Output warning of justification violation */

output_justification_violation(Disorder,contra-indicate):-
assert(output([i,did,diagnose,Disorder])),
assert(output(new_paragraph)).

output_justification_violation(Disorder,indicate):-
assert(output([i,'didn''t',diagnose,Disorder])),
assert(output(new_paragraph)).


/* Produce the explanation for a list of diagnoses. For each diagnosis,
display an introduction, then output the supporting evidence of each
type */

produce_diagnosis_explanation(Blackboard,Justification,[]).

produce_diagnosis_explanation(Blackboard,Justification,[(Root,Diagnosis)
|Diagnosis_list]):-
justification_introduction(Diagnosis,Justification),
justify_diagnosis(Blackboard,Root,Diagnosis,variable,Justification),
justify_diagnosis(Blackboard,Root,Diagnosis,symptom,Justification),
justify_diagnosis(Blackboard,Root,Diagnosis,history,Justification),
justify_diagnosis(Blackboard,Root,Diagnosis,relation,Justification),
produce_diagnosis_explanation(Blackboard,Justification,Diagnosis_list).

produce_diagnosis_explanation(Blackboard,Justification,[(Root,Diagnosis)
|Diagnosis_list]):-
assert(output(new_paragraph)),
assert(output([my,belief,in])),
string_to_atoms(Diagnosis,Diagnosis_output),
assert(output(Diagnosis_output)),
assert(output([was,not,affected,by,any,evidence])),
produce_diagnosis_explanation(Blackboard,Justification,Diagnosis_list).


justification_introduction(Diagnosis,Justification):-
relevant_evidence(Blackboard,Root,Diagnosis,Hypothesis_type,Evidence),
assert(output(new_paragraph)),
string_to_atoms(Diagnosis,Diagnosis_output),
assert(output(Diagnosis_output)),
assert(output([was])),
verb(Justification,past,2,V,[]),
assert(output(V)),
assert(output([by])),
assert(output(colon)).


/* Justify the diagnosis for a particular type of evidence */

justify_diagnosis(Blackboard,Root,Diagnosis,Evidence_type,Justification)
:- hypothesis_type(Evidence_type,Hypothesis_type),
setof(Evidence,relevant_evidence(Blackboard,Root,Diagnosis,Hypothesis_ty
pe,Evidence),Evidence_list),
generate_justification_output(Blackboard,Root,Diagnosis,Hypothesis_type,
Justification,Evidence_list).

justify_diagnosis(Blackboard,Root,Diagnosis,Evidence_type,Justification)
:-


/* Generate the justification output for one type of evidence */

generate_justification_output(Blackboard,Root,Node,Hypothesis_type,Justi
fication,[]).

generate_justification_output(Blackboard,Root,Node,Hypothesis_type,Justi
fication,[Evidence|Evidence_list]):-
singleton_descendents(disorder,Root,Node,Node_descendents),
sum_evidence_effect(Blackboard,Root,Node_descendents,Hypothesis_type,Evi
dence,0,Sum),
apriori(disorder,Root,Node,A),
W is Sum/A,
output_justification(Blackboard,Root,Node,Hypothesis_type,Evidence,Justi
fication,W),
generate_justification_output(Blackboard,Root,Node,Hypothesis_type,Justi
fication,Evidence_list).


/* Sum the weight of evidence for a node */

sum_evidence_effect(Blackboard,Root,[],Hypothesis_type,Evidence,Sum,Sum)
:-

sum_evidence_effect(Blackboard,Root,[Node|List],Hypothesis_type,Evidence
,Sum_so_far,Sum):-
get_evidence_weight(Blackboard,Root,Hypothesis_type,Evidence,Node,W),
apriori(disorder,Root,Node,A),
New_sum_so_far is Sum_so_far+(W*A),
sum_evidence_effect(Blackboard,Root,List,Hypothesis_type,Evidence,New_su
m_so_far,Sum).


/* Get the weight of evidence for a node */

get_evidence_weight(Blackboard,Root,Hypothesis_type,Evidence,Leaf_node,W
):-
blackboard(Blackboard,Hypothesis_type,Root,Node,Evidence,W),
singleton_descendents(disorder,Root,Node,Node_list),
member(Leaf_node,Node_list).

get_evidence_weight(Blackboard,Root,Hypothesis_type,Evidence,Leaf_node,1
.0).


/* Produce the justification output for an item of evidence */

output_justification(Blackboard,Root,Node,Hypothesis_type,Evidence,indic
ate,W):- W<1.

output_justification(Blackboard,Root,Node,Hypothesis_type,Evidence,contr
a-indicate,W):- W>1.

output_justification(Blackboard,Root,Node,variable_hypothesis,Evidence,J
,W):-
variable_level_description(Blackboard,Evidence,Level_description),
assert(output(new_line)),
assert(output(space(4))),
assert(output(Level_description)),
string_to_atoms(Evidence,Atom_list),
assert(output(Atom_list)).

output_justification(Blackboard,Root,Node,relation_hypothesis,Evidence,J
,W):-
output_relation_description(Root,Node,Evidence).

output_justification(Blackboard,Root,Node,Hypothesis_type,Evidence,J,W):
- hypothesis_type(Evidence_type,Hypothesis_type),
current_data(Evidence_type,Evidence,Attribute),
assert(output(new_line)),
assert(output(space(4))),
Output_string is_string Attribute & " " & Evidence,
string_to_atoms(Output_string,Atom_list),
assert(output(Atom_list)),
indicate_support_strength(W).


/* Output the description of a relationship */

output_relation_description(Root,Node,Dependents):-
relation_evidence(Node,Root,R,Left,Right,Comp,Dependents),
assert(output(new_line)),
assert(output(space(4))),
assert(output(spaceless(Left))),
assert(output(spaceless([Comp|Right]))),
fail.

output_relation_description(Root,Node,Dependents):-
relation_evidence(Compound_node,Root,R,Left,Right,Comp,Dependents),
descendent(disorder,Root,Compound_node,Node),
assert(output(new_line)),
assert(output(space(4))),
assert(output(spaceless(Left))),
assert(output(spaceless([Comp|Right]))),
fail.

output_relation_description(Root,Node,Dependents).


/* Finding evidence for confirmation/disconfirmation. Evidence is
relevant to a node if it affects any related node */

relevant_evidence(Blackboard,Root,Diagnosis,Hypothesis_type,Evidence):-
blackboard(Blackboard,Hypothesis_type,Root,Diagnosis,Evidence,P).

relevant_evidence(Blackboard,Root,Diagnosis,Hypothesis_type,Evidence):-
blackboard(Blackboard,Hypothesis_type,Root,Node,Evidence,P),
related_to(disorder,Root,Diagnosis,Node).

/* Indicate the strength of support of evidence */
```

```
indicate_support_strength(0.0):-
assert(output(["(",ruled,out,")"])).

indicate_support_strength(W).

/* ---------------------------------------------------------------- */
/* ---------------------- EFFECT OF EVIDENCE ---------------------- */


/* output the effect of a piece of evidence */

output_evidence_impact(Entity,Type,Tense):-
dialogue_context(diagnosis(Blackboard)),
hypothesis_type(Type,Hypothesis_type),
bagof((Frame,P),evidence_impact(Blackboard,Hypothesis_type,Frame,Entity,
P),Evidence_list), evidence_impact_introduction(Entity,Tense),
generate_evidence_impact_output(Evidence_list),
assert(output(end_text)).

output_evidence_impact(Entity,Type,Tense):-  /* if there is no impact */
verb(have,Tense,2,V,[]),
assert(output([Entity])),
assert(output(V)),
assert(output([no,effect,on,my,diagnosis])),
assert(output(end_text)).


/* effect of evidence on specific diagnosis */

output_evidence_impact(Entity,Type,Disorder,Tense):-
dialogue_context(diagnosis(Blackboard)),
hypothesis_type(Type,Hypothesis_type),
evidence_impact(Blackboard,Hypothesis_type,Frame,Entity,P),
impact_description(P,Description),
string_to_atoms(Frame,Frame_output),
assert(output([observation,of,Entity])),
assert(output(Description)),
assert(output([belief,in])),
assert(output(Frame_output)),
assert(output(end_text)).
                                        /* when there is no impact */
output_evidence_impact(Entity,Type,Disorder,Tense):-
verb(have,Tense,2,V,[]),
assert(output([Entity])),
assert(output(V)),
assert(output([no,effect,on,my,diagnosis,of])),
string_to_atoms(Frame,Frame_output),
assert(output(Frame_output)),
assert(output(end_text)).


/* Get evidence impact */

evidence_impact(Blackboard,variable_hypothesis,Frame,Variable,P):-
variable_impact(Blackboard,Frame,Variable,VP),
total_relation_impact(Blackboard,Frame,Variable,TRP), P is VP*TRP,
P \= 1.0.

evidence_impact(Blackboard,Hypothesis_type,Frame,Entity,P):-
blackboard(Blackboard,Hypothesis_type,Root,Frame,Entity,P).


/* Get impact of variable evidence (is 1 if no blackboard entry) */

variable_impact(Blackboard,Frame,Variable,VP):-
blackboard(Blackboard,variable_hypothesis,Root,Frame,Variable,VP),!.

variable_impact(Blackboard,Frame,Variable,1.0).


/* Get impact of variable as it figures in relationships */

relation_impact(Blackboard,Frame,Variable,RP):-
blackboard(Blackboard,relation_hypothesis,Root,Frame,Dependents,RP),
member(Variable,Dependents).

total_relation_impact(Blackboard,Frame,Variable,TRP):-
bagof(RP,relation_impact(Blackboard,Frame,Variable,RP),RP_list),
product_list(RP_list,1,TRP),!.

total_relation_impact(Blackboard,Frame,Variable,1).


/* Generate the introduction */

evidence_impact_introduction(Entity,Tense):-
verb(be,Tense,2,V,[]),
assert(output([the,effect,of,Entity,on,my,belief,in,possible,diagnoses])
), assert(output(V)),
assert(output([as,follows])),
assert(output(colon)).


/* Generate the output from list of effects */

generate_evidence_impact_output([]).

generate_evidence_impact_output([(Frame,W)|List]):-
impact_description(W,Description),
string_to_atoms(Frame,Frame_output),
assert(output(new_paragraph)),
assert(output(Frame_output)),
assert(output(Description)),
generate_evidence_impact_output(List).


/* Verbal interpretation of impact of evidence */

impact_description(0.0,[ruled,out]).

impact_description(W,[increased]):-
W>1.0.

impact_description(W,[decreased]):-
W<1.0.
```

```
/* ---------------------------------------------------------------- */
/* ------------------------ MAKING SUPPOSITIONS ------------------- */

/* Make a supposition */

make_supposition(Type,Entity,Value):-  /* already supposed for entity */
dialogue_context(diagnosis(supposition)),
retract(dialogue_context(supposition(Entity,_))),
make_supposition(Type,Entity,Value).

make_supposition(Type,Entity,Value):-    /* not the first supposition */
dialogue_context(supposition(_,_)),
pose_context_question(Response),
Response=[yes],
make_supposition_diagnosis(Type,Entity,Value).

make_supposition(Type,Entity,Value):-              /* fresh supposition */
restore_data,
set_dialogue_context(diagnosis(supposition)),
copy_blackboard(disorder,supposition),
make_supposition_diagnosis(Type,Entity,Value).

make_supposition(Type,Entity,Value):-                /* couldn't do it */
assert(output([i,cannot,do,this])),
assert(output(end_text)).


/* Get the diagnosis with the input suppositions */

make_supposition_diagnosis(Type,Entity,Value):-
set_supposition(Type,Entity,Value),
make_diagnosis(supposition),
prepare_for_output,
output_diagnoses(supposition,conditional),
assert(dialogue_context(supposition(Entity,Value))).


/* Pose question regarding the context of the supposition */

pose_context_question(Response):-
setof([Entity,is,Value],dialogue_context(supposition(Entity,Value)),Supp
osition_list),
assert(output(new_sentence)),
assert(output([should,"I",still,assume])),
assert(output(list(and,Supposition_list))),
assert(output(query)),
assert(output(end_text)),
display_dialogue_output,
enter_dialogue(Response),!.


/* setting suppositions for what if.... */

set_supposition(Type,Entity,Value):-
retractall(current_data(Type,Entity,_)),
assert(current_data(Type,Entity,Value)).


/* ---------------------------------------------------------------- */
/* ---------------------- INFORMATION REQUESTS ------------------- */

/* output information */

output_information(value,diagnosis,Type,P,Tense):-
set_dialogue_context(diagnosis(disorder)),
output_diagnoses(disorder,Tense).

output_information(value,Entity,Type,P,Tense):-
retrieve_current_data(Entity,Type,Value,Status),
verb(be,Tense,2,Verb,[]),
assert(output([P])),
assert(output([Entity])),
assert(output(Verb)),
assert(output(Value)),
assert(output(Status)),
assert(output(end_text)).

output_information(level,Entity,variable,P,Tense):-
dialogue_context(diagnosis(Blackboard)),
variable_level_description(Blackboard,Entity,Level_description),
verb(be,Tense,2,Verb,[]),
assert(output([P,Entity])),
assert(output(Verb)),
assert(output(Level_description)),
assert(output(end_text)).

output_information(level,Entity,Type,P,Tense):-
assert(output([Entity])),
assert(output([is,not,a,variable])),
assert(output(end_text)).


/* getting the current value of data */

retrieve_current_data(Variable,variable,[Value,Units],[by,supposition]):
- dialogue_context(supposition(Variable,V)),
T is value(V,ir),
Value is_string string(V,ops),
data_parameter(Variable,units,Units).

retrieve_current_data(Variable,variable,[Value],[by,supposition]):-
dialogue_context(supposition(Variable,V)),
Value is_string string(V,ops).

retrieve_current_data(Variable,variable,[Value,Units],[by,Status]):-
dialogue_context(diagnosis(Blackboard)),
blackboard(Blackboard,raw_data,variable,Variable,V,Status),
Value is_string string(V,ops),
data_parameter(Variable,units,Units).

retrieve_current_data(Entity,variable,[unknown],[]):-
variable(Entity).

retrieve_current_data(Entity,Type,[Value],[by,supposition]):-
dialogue_context(supposition(Entity,Value)).
```

```
retrieve_current_data(Entity,Type,[Value],[by,Status]):-
dialogue_context(diagnosis(Blackboard)),
blackboard(Blackboard,raw_data,Type,Entity,Value,Status).

retrieve_current_data(Entity,Type,[unknown],[]):-
Archive=..[Type,Entity,_],
call(Archive).


/* getting the level of a variable */

/* returns level if P>0.95, "fairly low" or "fairly high" if P>0.5
otherwise usual */

variable_level_description(Blackboard,Variable,Level_description):-
blackboard(Blackboard,classified_data,Variable,Level,P),
P>0.95,
Level_description=[Level].

variable_level_description(Blackboard,Variable,Level_description):-
blackboard(Blackboard,classified_data,Variable,Level,P),
P>0.5,
member(Level,[low,high]),
Level_description=[quite,Level].

variable_level_description(Blackboard,Variable,[normal]):-
blackboard(Blackboard,classified_data,Variable,_,_).

variable_level_description(Blackboard,Variable,[not,classified]):-
data_parameter(Variable,standard_deviation,none).

variable_level_description(Blackboard,Variable,[unknown]).
```