A Methodology for the Design, Implementation and Evaluation of Intelligent Systems with an Application to Critical Care Medicine.

RONALD SUMMERS

Thesis submitted for the degree of Doctor of Philosophy

City University

Research Centre for Measurement and Information in Medicine

December1992

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 for study purposes, subject to normal conditions of acknowledgement.

ACKNOWLEDGEMENTS

Special thanks are extended to the following: Prof. Ewart Carson, research supervisor and mentor; to Prof. Derek Cramp for providing clinical support and introductions in the early stages of the work and guidance in the latter; and Dr. Mark Leaning for giving technical support during the critical first year of research. My clinical colleagues at the Royal Free Hospital, London, also deserve a special mention: Dr. Doreen Browne, for providing access to the Intensive Therapy Unit; Mr. Dominic Cox for his technical assistance; and the many clinical staff who contributed towards the project.

Thanks also to my fellow students within the the Research Centre for Measurement and Information in Medicine for creating a stimulating atmosphere from which I am sure we all have gained. My special thanks extend to Dr. John Chelsom for being a friend as well as a colleague.

Many thanks to Mr. Andy Morrison for drawing the diagrams, Mr. Paul Worthy for proof-reading, and to Mrs. Joyce Bernard for completing that unenvied task of typing the references.

ABSTRACT

This thesis illustrates the technology required to provide a new generation of clinical instrumentation systems for critical care medicine. This advance in measurement science is gained from the use of a knowledge-based component able to process information as well as data. To implement a clinical information system using knowledge-based technology requires prior knowledge of human and computer-based activity within the critical domain. A historical perspective is given to both of these topics which reflects the genesis of current practice. The application area is introduced by investigating a control system approach to managing patients who require ventilatory therapy.

It was found that no current methodology is wholly appropriate when a knowledge-based component is included in the technological paradigm. Therefore, a novel methodology for system design, implementation and evaluation is proposed, and its utility tested in the aforementioned application domain. The detailed processes involved in the evolution of a prototype system which aids the clinical user in the art of ventilatory therapy are shown. Three levels of machine intelligence are shown to be required, based on: context-sensitive deterministic mechanisms; pattern cognition; and decision support elements. A wider discussion of the important points raised in the practical use of the methodology focuses upon the philosophical basis of clinical information systems and the processes of knowledge elicitation, knowledge representation and intelligent system evaluation.

CONTENTS

PART III

LIST OF TABLES

LIST OF FIGURES

1 : INTRODUCTION

1.1 Background

This study focuses on the problems associated with introducing an intelligent measurement and information system into a medical environment. Clinicians use physiological measurement as a means of providing information about the well-being, or state, of a patient. If patient state is regarded as a dynamic entity then clinical measurement provides the basis from which patient state trajectory can be determined. Both patient state and its trajectory pattern have a direct bearing on clinical diagnosis and subsequently on the therapy or treatment regimes required for a successful outcome. This outcome may be defined as the return of the patient to normal physiological function, or alternatively may be more generally defined as the return of the patient to a less morbid state.

The medical environment of interest is the Critical Care Unit, an umbrella term which encompasses all high dependency environments, such as the Intensive Care Unit, Coronary Care Unit, Post-operative Recovery Ward, Spinal Injuries Unit, Burns Unit and Special Care Baby Unit. These environments are distinct units within a hospital system which cater for "...the care of patients who are deemed recoverable but who need continuous supervision and need or are likely to need the prompt use of specialised techniques operated by skilled personnel", (British Medical Association, 1967). Although this definition is over twenty years old, it is the one adopted by the Royal College of Nursing.

There have been many recent technological advances taking place within the Critical Care Unit which have mirrored those taking place in associated fields of medicine and bio-engineering. For instance, all of the following have influenced a change in patient management, (after Gregory, 1983) :-

i) development of new surgical procedures,
ii) development of new therapies and treatment regimes,
iii) an increase in general measurement technology,
iv) development of instrumentation systems to monitor the effects of the proposed therapy.

The advancement in surgical procedures, especially in cardiac and vascular surgery, has meant that a greater proportion of people can be treated for what were untreatable conditions a matter of years ago. This has had the effect of increasing the throughput of patients

who spend time in the Critical Care Unit as a normal part of their rehabilitation from surgery.

Progress in the use of new therapies and treatment regimes can be considered as having two distinct strands of development. The first are those developments which are necessary to cope with the new surgical procedures described above; the second are those instances where clinical research has indicated that an advance can be made on therapies and treatments in current use.

The increase which is evident in general measurement technology can be illustrated by observing the increase in sensitivity of sensors whilst their physical size and cost have been diminishing rapidly. Both the increase in sensitivity and decrease in physical size have contributed to improvements in the measurement process, for it is one of the corner-stones of measurement theory that a measuring system should not disturb the natural behaviour of the system of interest. In practice this is rarely possible, so one should always be aware of the sources of measurement error, a full account of which can be found in the literature, (eg. Barry, 1978; Hofmann, 1982). As advances are made in measurement technology, novel ways of measuring variables previously unmeasurable become apparent, which as a consequence increases the number of variables available to describe patient state. These, following appropriate processing, yield additional information for the clinician. However, more information about patient state does not necessarily lead to a more accurate diagnosis or a more beneficial treatment regime. Instead it may lead to a situation where extra information actually clouds the diagnostic process, and therefore has a resultant negative effect on patient management. Hence there is a need for concomitant development of instrumentation systems which can cope with the increased complexity of measurement.

Advances in instrumentation are closely related, and in most cases subservient, to increases in component technology. Emphasis has been placed on the development of electrical and electronic instruments where massive miniaturisation has become possible. Perhaps the best examples of this new generation of medical instrumentation are the computer-based or microprocessor-based instrumentation systems, developed in response to a general increase in the complexity of measurement. Not only can a computer-based system deal with the increased number of measured variables, but it can also process the

information at a rate of the order of magnitude of a million instructions per second. Thus, if necessary, quite complex processing functions can be dealt with in real-time. Another advantage of a computer-based instrumentation system is its capability to store and archive information by down-loading it to a permanent storage medium, (eg. magnetic diskette). These storage media are small, cheap, portable and information is easily transferred back to the working memory of the computer. With large increases in information handling capacity a need became apparent to assign a utility function which describes the pragmatic value of each piece of information. This function can then be used to find an automatic way of filtering out redundant data from those which are more useful. Such a function can be defined by increasing the knowledge content of the controlling software: intelligent instrumentation systems came about as a logical consequence of this requirement.

Given that intelligent measurement and instrumentation systems are an emergent technology, there is still some confusion over what constitutes intelligence in measurement. One author has suggested that the term 'intelligent instrumentation' has come to mean "...the use of a measurement system to evaluate a physical variable employing usually a digital computer to perform all (or nearly all) the signal/information processing" (Barney, 1985). This definition is unsatisfactory as most measurement systems, whether or not they employ a computer to do some signal or information processing, would fall within its scope. Another study describes intelligent measurement as a three level hierarchy: inferential measurement; pattern cognition; and measurement as part of an integrated information system (Carson et al, 1986; Finkelstein and Carson, 1986). It is this description which is adopted for the purpose of this study. Intelligent instrumentation employs software techniques drawn from Artificial Intelligence. This includes various distinct processes which contribute to the evolving intelligent system: the elicitation of domain dependent human expertise; the representation of that expertise in a form which can be utilised directly by the computer; methods of obtaining intelligible input and meaningful output to the system; and a coherent software control strategy capable of being able to reproduce the same output from the same set of inputs.

1.2 Problem Definition and Objectives of the Research Programme

One specific situation where intelligent instrumentation is required is in the process of artificial ventilation. This requires the use of a ventilator, which is a medical machine that allows the mechanical ventilation of a patient who is unable to breathe spontaneously. Currently, the settings of the ventilator which control respiratory performance are defined heuristically. This requires the clinician to have prior knowledge of the factors involved in this process. To counter any vagueness in this respect, an intelligent module is required which makes the rules for changing the ventilator settings more explicit. The objective of the study described here is to develop and implement a prototype Artificial Intelligent Respirator System (AIRS) which interfaces such an intelligent module with a modern ventilator.

In normal clinical practice patients who undergo surgery where a general anaesthetic agent is administered require mechanical ventilation until normal respiratory function returns. This usually occurs whilst the patient is still in the recovery room, which is part of the operating theatre complex. However for some surgical procedures, such as prolonged cardiac surgery, a patient may be transferred to a Critical Care Unit to await recovery. A patient may also require ventilatory therapy as a consequence of some respiratory dysfunction. This is the main category of patient for which AIRS has been designed.

AIRS can be described as a management system for adult patients who require respiratory support. It is NOT a diagnostic system, indeed the diagnostic state of the patient forms an important information source. The system is centred on the Puritan-Bennett 7200a microprocessor-controlled servo-ventilator. This microprocessor fulfils the dual role of control and display of an extensive primary data-set. For the former function strategically placed sensors in the pneumatic sub-system of the ventilator yield control data on pressure, flow and temperature of the inspired gas; for the latter function the data-set used by the ventilator is organised into a form that can be interpreted readily by the end-user. AIRS adds a knowledge-based component on top of that system. This allows computer-based advice to be given in the form of a suggestion for an appropriate action to be taken, acting on the basis of incoming data and knowledge of the current state of the patient.

The top-goal of this study is to specify the problems and provide solutions for the successful introduction of an intelligent instrumentation system into a specialised clinical unit, with an emphasis on clinical applicability and acceptability. To reach this goal a number of contributary objectives become apparent. These include: an appreciation of current working practice in the Critical Care Unit; the development of a novel methodology which caters for the design, implementation and evaluation of technological change; and an illustrative application of this methodology to the domain of respiratory management of adult patients undergoing mechanical ventilatory therapy. These goals and sub-goals are reflected in the organisation of the thesis.

1.3 Outline Plan of this Thesis

This thesis consists of three parts. The first part, which encompasses Chapters 2, 3, and 4, contains the reviews necessary to place this study in historical context and provides a new methodological advance which is able to cope with the introduction of an intelligent system capability into what is already a highly technological environment. Chapter 2 contains a historical review of Artificial Intelligence and historical development of the Critical Care Unit. This makes the reader aware of the great progress made in both of these areas during the last thirty years. Chapter 3 contains a critical review of techniques used in both a control systems approach and Artificial Intelligence approach to respiratory management. This provides an appreciation of technology employed currently in this domain. The need for a novel methodology for technological change is described in Chapter 4. This methodology is created from previous work in Requirements Engineering, Design Engineering, and methodologies which are not wholly appropriate for dealing with change at the technological level.

The second part comprises Chapters 5 and 6. These chapters describe the domain dependent aspect of the study. Chapter 5 contains the system specification for the implementation of AIRS. This utilises one element of the design methodology presented in the previous chapter. This chapter also contains a brief introduction to PROLOG, the computer language used in the implementation. Chapter 6 includes specific details of the implementation and evaluation of AIRS, illustrated by extracts from the actual program.

The third part consists of the last two chapters - Chapters 7

and 8, which attempt to place AIRS in context with other intelligent measurement and instrumentation systems, from which the appropriate conclusions can be drawn. The discussion in Chapter 7 can be divided into two parts. It begins with a general discussion on intelligent systems, and then focuses upon topics which are relevant to this particular study. One of the central themes of this part of the discussion is the evaluation of intelligent knowledge-based systems. A systems framework is utilised which permits the study of existing evaluation strategies from other fields of interest. Those strategies which contribute towards the more meaningful evaluation of computer systems which exhibit some sort of intelligent behaviour can then be elicited. Chapter 8 contains the conclusions drawn from this work together with recommendations for the future. This last comment perhaps emphasises the fact that AIRS should not be considered as a static piece of research work, rather development of it will continue as knowledge-based systems, like their human counterparts, evolve as new knowledge is acquired.

There are two appendicies. A critical review of Artificial Intelligence, with applications in medicine as the dominant theme, is presented in Appendix I. This introduces the concepts and constructs used in the domain and is useful as background information for the implementation of AIRS. An annotated program listing of several aspects of the implementation is given in Appendix II.

PART I

2 : HISTORICAL REVIEW OF ARTIFICIAL INTELLIGENCE AND HISTORICAL DEVELOPMENT OF THE CRITICAL CARE UNIT

2.1 Introduction

Artificial intelligence and the environment of the critical care unit are the two distinct themes which are of concern in this study. Current perspectives in artificial intelligence have their origins in the theory of numbers and computability, which itself is based on the fundamental concepts of measurement. However, it would be pedantic to begin this part of the review with the measurement system devised by the Egyptians in 3,500 B.C.! An alternative criterion for the onset of the review is revealed when we consider the history of artificial intelligence as being intertwined with the development of the modern computer with which it is inextricably linked. Therefore, the chosen starting point is 1812, the year in which Charles Babbage first conceived a fully mechanised automatic system to compute numbers - the Difference Engine. Where applicable within this review, emphasis has been placed on developments in the medical domain, which provide the necessary clinical relevance.

The historical review of the development of the critical care unit proved less problematical in defining a suitable time origin. The need for a specialised area within a hospital for critically ill patients was perhaps first realised in 1863 by Florence Nightingale.

2.2 Historical Review of Artificial Intelligence

Today the accuracy of tables which describe certain mathematical functions is taken for granted. However, in the early ninteenth century most mathematical tables contained numerous errors, much to the chagrin of the mathematicians, scientists, and engineers of that time. Therefore there was a real need to check the values contained within these tables; if done by hand this process would have been both time consuming and laborious and as a result may well have introduced further errors. In 1812 Charles Babbage attempted to solve these problems by his conception of the 'Difference Engine'. This was a mechanical device to compute and print tables of mathematical functions automatically. The principle on which the machine worked is that any mathematical function can be approximated to sufficient accuracy by means of a polynomial 'fit'. This polynomial can be inferred by using simple additions, that is, by constructing a difference table. It was this process that Babbage proposed to mechanise. The specification for his device was to compute tables of

sixth degree polynomials to twenty decimal places. An example of a difference table of a second degree polynomial can be found in Figure 2.1. Technology of the time was typified by Arkwright's steam mill, developed thirty years previously, a product of the industrial revolution used in the textile industry. A working prototype of the Difference Engine was capable of tabulating quadratic functions to eight decimal places. However, to progress to a fully mechanised system and as a result of the high precision required, all of the components were individually manufactured. This was a contributing factor for the renunciation of any further work on the Difference Engine in 1842. However some scientific profit was made from this project. In 1854 a Swedish engineer, Georg Scheutz, built a machine using Babbage's design which could tabulate fourth degree polynomials to fourteen decimal places.

The major reason Babbage abandoned work on his machine was that the Government of the day refused a request for further funding. He then became involved in a project to design a more universal mechanical machine which he called the 'Analytical Engine'. Included in the design specification were devices for input and output, a memory store (which Babbage described as a mathematical 'mill') and a control unit. If we substitute 'arithmetic logical unit' for 'mathematical mill' then we have all the components of a modern electronic computer! A comparison can be made between the architecture of the Analytical Engine and the more modern von Neumann computer architecture, (Figures 2.2 and 2.3). The memory store of the Analytical Engine was conceived as a device to hold numbers, whether in raw data format or as intermediate data values in the course of a numerical calculation. The mathematical mill performed the arithmetic operations on the numbers from the store. The design of the control unit, which ensured that the machine performed the desired operations in the correct sequence, was an example of the transformation of a design concept from the textile industry, as both processes exhibited their control by the use of punched cards. In the Analytical Engine another set of cards constituted the input device, each card representing the value of an input variable. The output device displayed the result of the calculation in the form of yet another punched card, or could print the result directly on to paper.

x	y	First Difference	Second Difference
		(1)	
0	6		(2)
		3	
1	9		2
		5	
2	14		2
		7	
3	21		2
		9	
4	30		2
		11	
5	41		2
		13	
6	54		

As the second difference is constant, the function $y=x^2+2x+6$ is an example of a *second* degree polynomial

FIGURE 2.1 AN EXAMPLE OF A DIFFERENCE TABLE FOR THE FUNCTION $y = x^2 + 2x + 6$

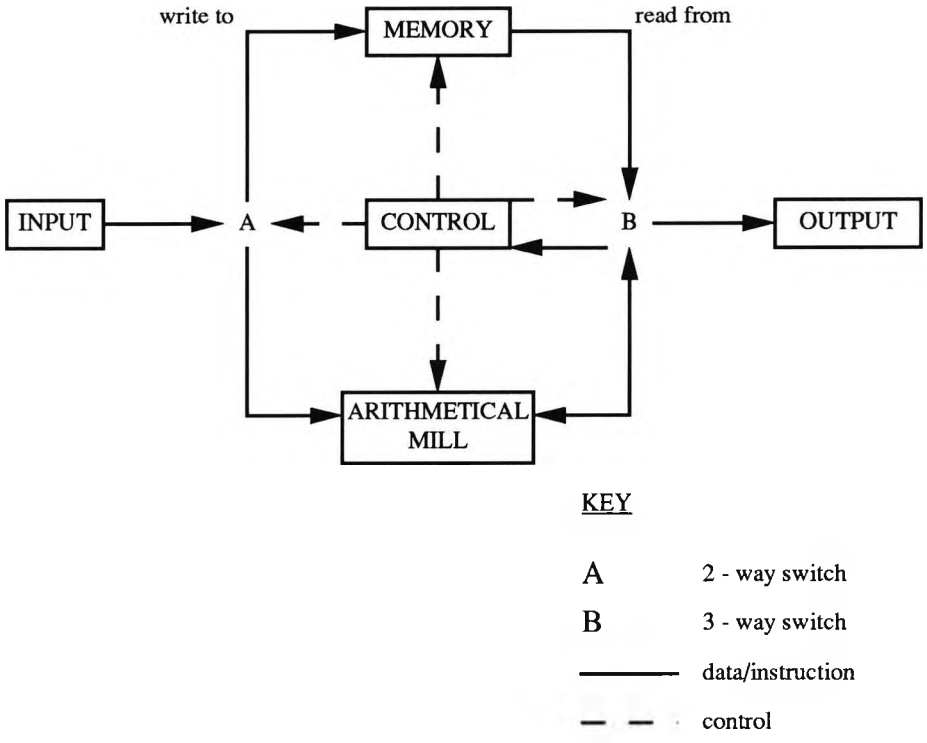


FIGURE 2.2 ARCHITECTURE OF THE ANALYTICAL ENGINE

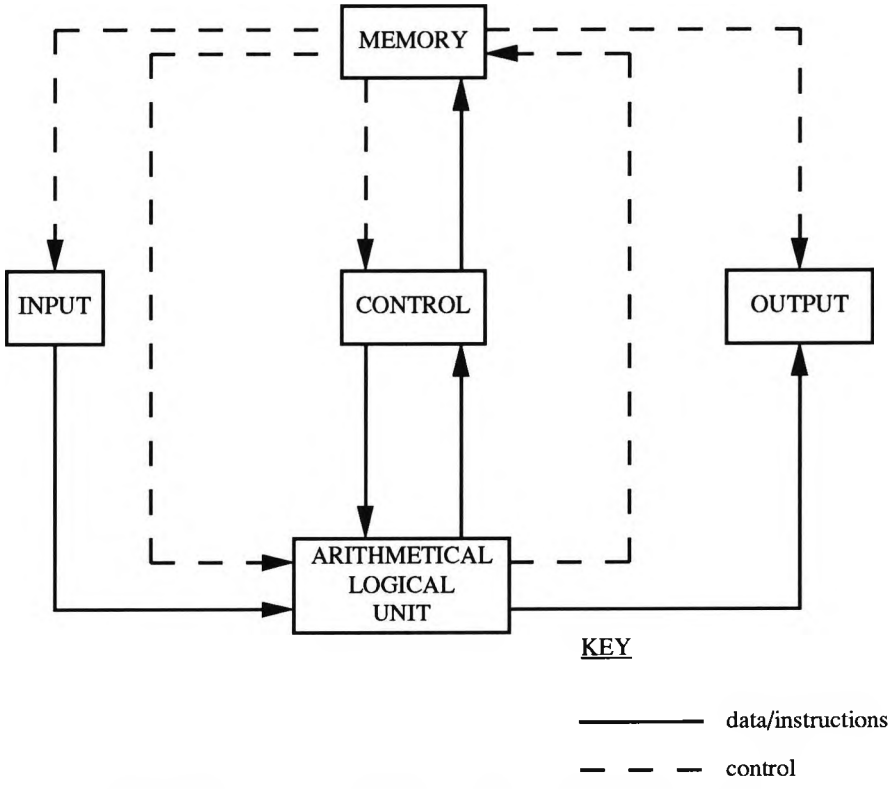
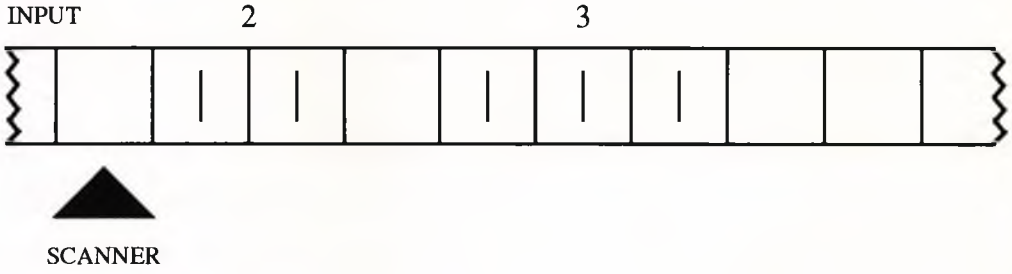


FIGURE 2.3 von NEUMANN ARCHITECTURE

In 1886 Hollerith extended the idea of using punched cards as devices to carry data. His system used an electrical device rather than a mechanical means for carrying data values. That is, the presence of a hole in the punched card would allow current to flow whereas the absence of a hole would stop it. Thus a binary code was used to carry data, the two states being current 'off' and current 'on'.

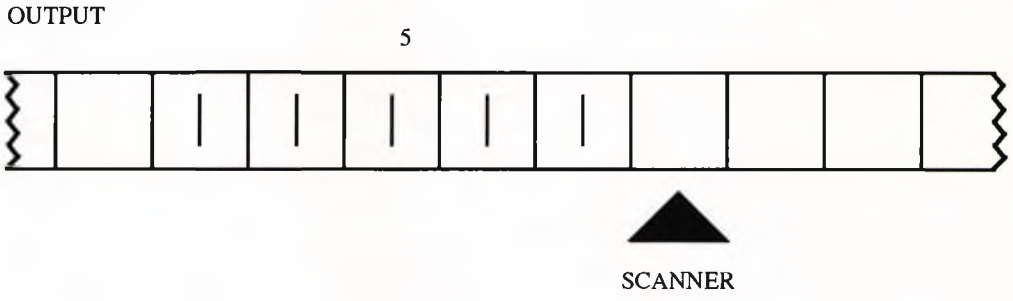
The binary encoding of data and instructions was exploited further by Turing in 1936. However, he anticipated the use of continuous paper tape divided into unit squares as a form of input and output, each unit square being binary encoded as a '1' or a '0', (Turing, 1937). His machine, soon to be termed the 'Turing Machine', could erase symbols as well as reading and writing them. The Turing machine was in fact a collection of machines, each one defined by its own decision table, (the decision table for adding two numbers is shown in Figure 2.4). The Turing machine was completely deterministic and could be described by its configuration. It worked automatically and can be considered as an early pattern recognition device. Thus, as well as mathematical functions, the Turing machine could also act as an automatic decision device, for example, in deciding whether or not one number is divisable by another. Indeed, Turing constructed his machine for the purpose of automatic decision-making. Turing's work of 1937 is of significance for two reasons; he was concerned with 'computable numbers' which almost coins the word 'computer' as the word to describe his device; and he realised the scope of his work, even at that early stage, by comparing the workings of his machine to the human brain.

The decade of the 1940s was an important one in the development of the computer. In 1944, at Harvard University U.S.A., H.H. Aiken led a research team responsible for the Automatic Sequence Controlled Calculator (ASCC), which can be described as a realisation of the 'paper' machine developed by Turing. The numbers used in the calculations were stored in 'registers' which consisted of sets of wheels. Each wheel could be in one of ten states and could therefore store the representation of a decimal digit. The storage register was made up of a group of 24 wheels which could store a 23 digit number plus its sign. There were 72 registers in all. The instructions for the ASCC were fed in via paper tape which was read by an electromechanical method. The complete machine contained approximately 750,000 parts and used more than 500 miles of wire. However, when the



DECISION TABLE

CONFIGURATION	SYMBOL SCANNED	
	BLANK	1
1	move R; config 1	move R; config 2
2	write '1'; move R; config 3	move R; config 2
3	move L; config 4	move R; config 3
4	no move config 4	erase; no move; config 4



The task, defined by the addition decision table, is to fill in the blank space with a '1', then to erase the last '1'. A four configuration decision table is sufficient to define the machine: configuration 1 moves along the tape until the first '1' is encountered and then moves into configuration 2; when it meets the blank separator, configuration 3 is reached; this then moves the the scanner along until it reaches the blank after the second group of '1"s, which acts as the signal to move back one space; configuration 4 is then reached, which erases the last '1' and remains as a stopping state.

FIGURE 2.4 A TURING ADDING MACHINE
(from Hodges, 1983)

ASCC was ready to be evaluated it was already out of date, superseded by the world's first electronic computer. The Electronic Numerical Integrator and Calculator (ENIAC) was designed by a research team at the University of Pennsylvania, U.S.A., led by J.P. Echert and J.W. Maunchly. The ENIAC consisted of 18,000 values, the majority of which were double triodes, and 1,500 relays.

The ENIAC machine could only store 20 numbers, so a research team led by J. von Neumann designed an improved version, termed the Electronic Discrete Variable Calculator (EDVAC). An important attribute of this machine was that the available memory capacity was increased considerably . However, the first operational stored program computer was a prototype machine designed by F.C. Williams at Manchester University, U.K., which was demonstrated in 1948. Indeed two English groups competed with each other to produce such a machine. The successful Manchester group used a Cathode Ray Oscilloscope as the storage medium, whereas a group at Cambridge University, under the direction of M.V. Wilkes, preferred the use of mercury delay-lines in their Electronic Delay Storage Automatic Computer (EDSAC).

By 1950, emphasis changed from hardware development to the development of programmable software which could be used to make computers do something. Activities on both sides of the Atlantic concentrated on constructing programs which would allow computers to make an intelligent decision or take an intelligent action. In the U.K. Turing's conjecture "...can machines think?" was a particular driving force which stimulated research activity.

In the U.S.A., Shannon described computers as not only being able to carry out numerical calculations, but were so general and flexible that they could "be adapted to work symbolically with elements representing words, propositions or other conceptual entities" (Shannon, 1950). Shannon illustrated these concepts with the game of chess. This problem domain was chosen because it is well-defined, both in terms of its final goal (checkmate) and in operational terms (a change of state occurs after each move in the game). The computer was programmed to find a solution from a reduced set of the whole, that is, situations where only a few of the defined pieces were in play at any one time (to simulate chess endgames). Thus, the problem to be solved was neither too simple nor too difficult. Evaluation of the chess-playing program was made by playing the computer against human chess-players of various ability.

The work of Turing and Shannon serves to illustrate the two schools of thought which constituted computer-based intelligence in the early 1950s. The view of the psychologists (Turing) was that computer-based intelligence should be considered as analagous to human intelligence, that is, the computer and its controlling software should imitate the human brain and thought processes respectively. Alternatively, Shannon champions the perspective taken by engineers, and viewed computer intelligence as mechanistic in nature, controlled by computer algorithms which imitate what humans do rather than how they do it. To illustrate the currency of this divide the debate surfaced again some thirty years later (Kolata, 1982; Waldrop, 1984).

In 1954 two precursors of artificial intelligence led to the field of Medicine and Bio-engineering being one of the major domains for its application. First, Meehl argued that many clinical predictions could be made by statistical rather than intuitive means (Meehl, 1954). Second, Savage re-introduced Bayesian statistical decision theory (Savage, 1954), which was later to provide the basis for value judgements in several 'intelligent' computer-based systems.

One of the most notable landmarks in the study of computer-based intelligence was a Summer School in 1956, held at Dartmouth, U.S.A. and organised by J. McCarthy. The 'Dartmouth Summer Research Project on Artificial Intelligence' was a resounded success, as it achieved its main aim of unifying the efforts of research groups in the U.S.A. Along with McCarthy, present at some time during the six weeks were, for example, Simon, Newell, Minsky and Shannon. Newell and Simon presented a logic theorist model to their colleagues which was a forerunner to their General Problem Solver program (Newell and Simon, 1957). The General Problem Solver was based upon a model of human activity. That is, humans going about their everyday tasks bring some general processes to bear for their successful completion. The General Problem Solver was designed to separate the problem solving strategy from the task-specific knowledge by employing a 'means-end analysis'. This analysis considered the problem solving process as a series of states, with a trajectory of discrete steps from 'current state' to 'goal state'. For a successful outcome each state in the trajectory is nearer to the goal state than its predecessor. This iterative procedure used appropriate operators to move from state to state, with a full list of operators used to reach the goal state termed the 'solution plan'. These operators often took the form of heuristic rules, which meant usually that there was more than one solution plan

for each problem considered. This ambitious project was hampered by several inadequacies in its implementation. For instance, the problem of choosing which attributes to use to classify the differences between successive states; finding appropriate operators (heuristic rules) to act on the attributes; and only trivial problems were ever considered, mainly because of excess computer times and memory allocation required to compute the solution plan (Newell et al., 1960).

In 1959 McCarthy described his programming language for non-numeric computation. LISP was developed as a practical LISt Processing language with a recursive function capability. Using LISP, iterative processes such as the ones used by Newell and Simon in their program for General Problem Solving, could be implemented more directly. The year of 1959 was also a landmark for research in artificial intelligence in medicine, as Ledley and Lusted published their seminal paper on the reasoning foundations of medical diagnosis (Ledley and Lusted, 1959). This paper described how statistics and Bayesian analysis could be used to enhance the diagnostic process.

Entering the decade of the 1960s the first generation of computers, which had the valve as its most fundamental component, were being superseded by transistor-based machines. Several advantages ensued because of this change in technology, these included: various cost-type benefits, the most obvious being the decrease in financial outlay for hardware purchase; the integral components were smaller, decreasing the size of the computer; transistors dissipated less heat than valves, so although ventilation systems were still required they were not as specialised; and as a consequence of their size and decrease in heat dissipated, transistor-based computer systems could have more components built into them, thus increasing computing power. This latter advantage was to have far reaching effects, as the scope of the problems for which a computer-based implementation was sought diversified.

Work began in the early 1960s to develop computer-based tools useful for mathematicians. A heuristic-based computer program to perform symbolic integration at University entrance level was developed (Slagle, 1961). This proved to be the forerunner of MACSYMA, a well known mathematical symbolic programming language developed during the 1970s at the Massachusetts Institute of Technology, U.S.A. Game-playing exercises were still popular to those working in artificial intelligence, as well as chess, the strategies used in

draughts (American checkers) were also studied (Samuel, 1963). These should not be considered as recreational activities as the benefits to emerge from this work were computer-based search strategies used to obtain successful solution paths in directed graphs and hierarchies.

Computer vision systems were pioneered in the mid-1960s (Roberts, 1965), with the computer component of the system being implemented to understand simple polyhedral block scenes. Much preprocessing was required before input into the computer-based part of the system. A pattern matching algorithm was used to compare the current blocks scene to the one stored in computer memory.

Perhaps the most notorious program of the 1960s was ELIZA, which simulated a non-directive psychotherapist (Weizenbaum, 1966). ELIZA was conceived as a parody to machine understanding. The natural language interface used in its implementation was neverthless an indication of what would be acceptable to the future users of intelligent systems. There is no doubt that the domain chosen helped in the overall success of the program in achieving its objective of demonstrating the ridiculousness of a machine being able to understand true natural language. If a print-out of a machine - subject interaction were studied, the level of machine understanding may at first sight seem impressive. However, on closer examination the deterministic nature of the program could be deduced. The program was implemented in such a way that a deterministic reply was given on the identification of a key word in the subjects' answers. Likely key words in the domain were, for example, 'mother', 'family' and 'sex'. Each word would have their own predetermined answer composed in such a way as to continue the interrogation. In the event of a reply by the subject which contained no recognisable key words, an ambiguous answer was given, such as 'Please continue'. This reply was chosen until the next key word was recognised, thus extending the interview.

The period between the end of the 1960s and the start of the 1970s can be viewed as a watershed in the field of artificial intelligence research. Groups on both sides of the Atlantic consolidated their positions, guided by results of research in this young science. Artificial intelligence was found to be much more exacting than researchers first realised, where success lay not in 'general problem' solving as at first thought, but rather in computer-based implementations of knowledge from narrow domains of interest. Heuristic rules were necessary to limit the number of possible

solutions found by computer search techniques, otherwise valuable computer time and memory were taken up computing and storing non-optimal pathways. There was also a further increase in computer technology. The change of technology to this third generation of computer hardware had all the advantages of the previous change; that is, there were advantages of decreased financial outlay to install a new computer, integral computer components were smaller (based on integrated circuit technology), computing power increased while both size and heat dissipation decreased. Two other factors could also be identified for the necessary transition into the third generation of machines: the standardisation of operating systems as the interface between hardware and software; and a standardisation of component technology (imposed and driven by the market leader in computer manufacture).

Up to the early 1970s the LISP programming environment was the only one specialised for research into artificial intelligence. In 1972 at the University of Marseille, France, a new programming language was developed for PROgramming in LOGic (Roussel, 1975). PROLOG became more popular as it became more generally available, partly because it is a relational language based on first predicate calculus, but also because it subsumed most of the list processing functions performed by LISP.

Research in artificial intelligence was set to expand using the third generation computer hardware when a set-back to its progress occurred in the U.K. The Government of the day commissioned a report into artificial intelligence research from the Science Research Council (now the Science and Engineering Research Council). Its author, the physicist Sir James Lighthill, saw no need for a separate field for artificial intelligence and found no organised body of techniques that represented such a field (Lighthill, 1972). Lighthill saw the fields of automation and computer science coming together to fill whatever research gap ensued. As a result of the report there was an immediate cessation of work in artificial intelligence in the U.K. Affected personnel were dispersed to other research centres, mainly in the U.S.A., where their contributions were received gladly.

However, perhaps as a response to the Lighthill report which had a knock-on effect in the U.S.A., research was initiated in development of the techniques used in artificial intelligence. For example, new knowledge representation schemes appeared, computer search techniques began to mature and inter-domain research activity

became apparent, (where the medical domain was an early application for artificial intelligence). During the 1970s feasible approaches were demonstrated for speech understanding, language processing, and computer vision. A sub-set of artificial intelligence called 'expert systems' also began to emerge.

From 1971 to 1976 the HEARSAY-II speech understanding system was developed at Carnegie-Mellon University, U.S.A., (Erman et al., 1980). One of the first tasks of this research was to define when understanding (rather than recognition) of the spoken word had taken place. A behaviourist view was employed which allowed understanding to take place at several levels. The HEARSAY-II program was said to understand speech if it could perform one of the following tasks : give a correct answer to a question; paraphrase a paragraph; take inferences from a paragraph; translate a paragraph into another language; or predict what might be said next. In its summative evaluation the program could understand sentences with 90% accuracy from continuous speech, based on a thousand word vocabulary. The HEARSAY project used a 'Blackboard System Architecture', which allows and controls information flow from multiple knowledge sources. In this way knowledge from different levels in the task-hierarchy can be considered at the same time.

A natural language program, SHRDLU, was developed at the Massachusetts Institute of Technology, U.S.A., (Winograd, 1972), which was designed to interface with an 'artificial blocks world' (c.f. the work by Roberts discussed earlier). The significance of SHRDLU, although limited in extent, was that it was the first program to integrate successfully the syntax and semantics of natural language with a knowledge base.

Continuation of work started in the 1960s on computer vision systems resulted in an industrial prototype (Gleason and Agin, 1979). The purpose of this system was the assessment of quality assurance of industrial workpieces. A special lighting system was used to illuminate the workpiece whose edges were extracted using a continuous scan process. Any workpiece which did not match with the representations held in the memory of the computer were rejected.

Expert systems began to emerge in the 1970s, with the medical domain being a keen user of the technology. (The definition of an expert system has been given in Chapter 1, and medical expert systems are discussed in more depth in Appendix I). The genesis of medical

expert systems perhaps stems from three fields of interest: the clinical algorithm; medical databases; and clinical decision making using decision theoretic techniques.

The clinical algorithm can be considered as a simple decision-making tool. It has been successfully applied to the encoding of triage protocols for use by nurses (Perlman et al. 1974), and giving therapeutic advice for acid-base disorders (Bleich, 1972). Deficiencies in this method are well recognised, such as its unwieldiness in large domains and the difficulty of maintaining the medical knowledge contained in the algorithm. The lack of any explicit model also makes justification of a course of action difficult to explain.

The hospital-kept patient notes form an original 'paper' database. With the advent of advanced instrumentation systems not only did it become possible to computerise these notes, but they could also be cross-referenced with domain-specific databases. Indeed, large domain-specific databases are being gathered for various clinical problems (Weyl et al., 1975; Mabry et al., 1977). There are drawbacks to the use of clinical database systems, which include the need for a standard approach to nomenclature and interpretation for comparison between similar systems, and rare disorders may have so few references in the database that any inferences made may be statistically inappropriate.

One statistical method employed in clinical decision-making which provides for uncertainty is the use of Bayes theorem. For this to be consistent, the theory requires quantitative values for the a priori and conditional likelihoods for each disease state and constituent manifestations under consideration. Applications which use this method are typically from small domains, as otherwise the time required to carry out the a posteriori mathematical calculations required for a set of possible solutions becomes untenable. Examples of the use of Bayes theorem in clinical decision-making are in the management of patients with acute renal failure (Gorry et al., 1973), and from the clinical response to digitalis therapy (Gorry et al., 1978). The main disadvantage of this method is the difficulty in obtaining a reasonable estimate of the a priori probabilities for disease states and their manifestations in each domain. It follows from this observation that large medical domains and multiple disorder protocols are not good examples of potential areas of application. However, there is one notable exception. A system for acute abdominal

pain has been in the process of development since 1972 at the University of Leeds, U.K. (de Dombal et al., 1972; Horrocks et al., 1972). An evaluation study has been performed which assessed the impact of the system on clinical practice. This study involved 8 hospitals, 250 clinicians and 16,737 patients at both national and international sites (Adams et al., 1986).

When research from all three of the above approaches are combined, any resultant expert system would benefit from the advantages offered by each individual approach whilst each of their individual disadvantages are minimised. For example, a clinical algorithm could provide the procedural knowledge required in an expert system, while data from a domain-specific clinical database could provide the statistical insight required to obtain the a priori probabilities for the disease states used in a Bayesian analysis of clinical prediction.

As research entered the 1980s it became clear that the encapsulation of knowledge into computer-based systems was necessary and central to the field of artificial intelligence. Suddenly 'expert systems' and 'intelligent knowledge-based systems' became synonyms for research into artificially intelligent systems. Although one definition of 'expert system' has already been given (Chapter 1), it is pertinent to add a second more comprehensive definition at this stage :

> "An expert system is an intelligent computer program that uses knowledge and inference procedures to solve problems that are difficult enough to require significant human expertise for their solution. The knowledge necessary to perform at such a level, plus the inference procedures used, can be thought of as a model of the expertise of the best practitioners of the field.
>
> The knowledge of an expert system consists of facts and heurisitics. The 'facts' constitute a body of information that is widely shared, publically available, and generally agreed upon by experts in a field. The 'heuristics' are mostly private, little-discussed rules of good judgement (rules of plausible reasoning, rules of good guessing) that characterise expert-level decision-making in the field. The performance level of an expert system is primarily a function of the size and quality of the knowledge base that it possesses."
>
> (Feigenbaum, 1982).

This definition emphasises the requirement for an expert system, as

well as dividing it into its constituent parts of facts and rules. Facts are universally known whereas rules usually employ 'local' knowledge, which could be one reason why expert systems are generally not very portable.

The early 1980s also saw the wholesale manufacture of microprocessors, which brought the cost of buying a computer down dramatically. With the advent of the microcomputer a potential home-computing market was established. Versions of LISP, PROLOG and other software packages became available for use with these microcomputers, which is perhaps one reason why expert system technology has proliferated. In 1982 the Japanese officially began a well-funded ten year research project to create a further generation in computer technology. This generation of machinery will be capable of computing in parallel, that is, perform many computational tasks concurrently. The significance of this project to this review is twofold : first, PROLOG has been chosen as the implementation language; and second, computers operating in a parallel mode have a great research potential in the field of artificial intelligence. These facts, taken in conjunction with the fact that intelligent computer-based systems are now found in commercial use, ensure that research in artificial intelligence will play an ever-increasing role in the future development of intelligent devices.

2.3 Historical Development of the Critical Care Unit

The critical care unit can be described as a specialised and confined area within a hospital unit where critically ill patients are gathered together. The creation of such an environment has various consequences. For instance: the highly trained nurses required to staff the unit can be deployed in an efficient way, more often than not in a ratio greater than unity between nurse and patient; the sophisticated instrumentation systems needed for enhancing patient care are utilised for a high proportion of their operational lives, thus expensive and specialised equipment does not sit idle for very long; and the critical care unit provides an ideal setting for medical training and research. All of these factors contribute to the finding that the establishment of a critical care unit within a hospital has the effect of reducing overall patient mortality and morbidity, (McCleave et al, 1977). As an illustration of the popularity of this method of dealing with critically ill patients, an American study has shown that over 80% of all short-term General Hospitals in the U.S.A.

which have more than 200 beds have a critical care unit (Snyder et al., 1981). However, this type of care is a phenomenon of the latter half of the twentieth century, which owes much to the establishment of the post-operative recovery room, perhaps first chronicled by Florence Nightingale, who observed,

> "It is not uncommon, in small country hospitals, to have a recess or small room leading from the operating theatre in which the patients remain until they have recovered from the immediate effects of the operation. "
>
> (Nightingale, 1863).

One of the problems of managing critically ill patients is that although initially there may be only one pathological condition which accounts for the prime complaint, in a relatively short period of time many physiological disturbances can contribute to overall patient state. Therefore there is potential for a number of clinical specialists to assume charge of the care of the patient. This can lead to a most unsatisfactory situation where a conflict of management approaches can be adopted by the different specialists. Therefore there is a need for a holistic approach to the management of the critically ill patient, which was first perceived by Kirschner, a surgeon from the hospital of the University of Tuebingen, Germany, (Kirschner, 1930). He designed and had built a dual purpose unit which catered for post-surgical patients as well as for the critically ill.

It was the advent of World War II which brought about further developments in critical care medicine. Due to the number of casualties involved, a systems-type approach to the management of the battle wounded was adopted in many places. For example, in July, 1943, a thoracic surgical tent was established in Bizerte, North Africa, to deal with incoming wounded. This enabled the limited number of specialised medical personnel to be deployed in the most efficient way. Further examples of this intensive approach to patient care in World War II (from an American perspective) are well documented in a series of volumes published by the Office of the Surgeon General, Department of the Army, Washington, D.C. (U.S. Army Medical Services, 1955; 1963; 1964).

The management of civilian crises has also been responsible for the evolution of the critical care unit. Indeed, it was one such crisis, namely the poliomyelitis epidemic of 1952 in Scandinavia, which ultimately provided the impetus for the creation of purpose-designed specialist units for the management of the critically ill.

However in 1952 no such unit existed, so in Copenhagen, Denmark, a hospital ward was temporarily seconded to cater for victims of the poliomyelitis epidemic who had severe respiratory problems caused by their primary complaint. This ward was supervised by anaesthetists, and as technology had yet to improve on the cuirass ventilator, intermittent positive pressure ventilation was applied manually for prolonged periods of time on the tracheotomised patients, (Ibsen, 1954). Clinical acceptance of this treatment regime was enhanced because evaluation of the methodology could be undertaken. Epidemiologists compared the mortality rates from respiratory paralysis during the poliomyelitis epidemic of 1952 to that of a similar epidemic in Scandinavia which took place three years earlier. Results showed that throughout Scandinavia the mortality rate during the 1949-1950 epidemic was 85%; prior to the anaesthetists' intervention in 1952 the mortality rate in Denmark was 87%; however after intervention by the anaesthetists in the way previously described, the mortality rate fell by more than half to 40%, (Lassen, 1953).

The clinical knowledge gained from these experiences had at least two consequences. First, a technological advance was made in the design of mechanical ventilators, which began to include an automatic way of delivering intermittent positive pressure ventilation. Second, a methodological advance was made in the design of a special unit which catered specifically for the critically ill.

The first purpose-built civilian multidisciplinary critical care units opened almost simultaneously in 1958, in Baltimore, U.S.A., and Uppsala, Sweden, (Safar et al, 1961; Holmdahl, 1962). Taking the latter unit as an example of the methodological advance, the 24 bed unit was split into two wards. The larger of the two wards contained 13 beds and was administered by anaesthetists, and could be considered as an extension to the post-operative recovery room. This can be visualised as the forerunner of the medico-surgical critical care units found in most large hospitals. The other ward, which contained 9 beds, was administered by cardio-thoracic surgeons and was perhaps the originator of the coronary care unit.

Other specialist units that can be considered under the umbrella term of critical care include the neurological intensive care unit and the special care baby unit. These have a more chequered historical development, owing much to the personality of the clinicians involved rather than any structured advancement in

methodology for patient care. A small three bed intensive care unit for the post-operative management of the neurosurgical patient was in existence in 1923 in Baltimore, U.S.A. This unit was directed by W.E. Dandy, who realised the efficacy of this type of patient care, (Harvey, 1974). A special care baby unit for neonatal intensive care was first established in Chicago, U.S.A., in 1927, (Klaus and Kennell, 1970). However, treatment and methods were still crude, often using medical instruments designed for adults on neonates. The creation of the unit in Chicago provided an environment which allowed clinicians to design proper sized instruments for use on neonates, (Flagg, 1928). An increase in general instrument technology has also contributed to the care of the critically ill. As clinical instrumentation has become more sensitive the amount of test substance required to obtain a reliable reading has become less. For instance, for paediatric blood-gas analysis a blood sample of 40 ul is all that is required to obtain a reliable measure from most modern machines. This has allowed serial analyses of variables to be undertaken in even the smallest of neonates. Patient management can then be optimised at regular intervals, which increases the chance of patient survival. Further examples of how other 'specialist' critical care units have developed from a clinical viewpoint can be found in an article by Hilberman (1975).

With the establishment of the critical care unit as a distinct entity within a hospital system, emphasis on its development has changed from a clinical perspective to a technological one. The use of computers to monitor continuous physiological signals first became a tool for patient management in 1964, in Los Angeles, U.S.A. (Jensen et al, 1966). Since then many more patient data items have become available on-line, and to counteract a 'data-explosion' these must be made available to a computer-based patient data management system (Booth, 1983). Such a system has existed in Kuopio, Finland, since 1986 (Kari, 1988), where the entire patient data set is captured either on-line or manually entered via a keyboard. A problem encumbent with any system of this type is its universal appeal, as the patient data set differs from hospital to hospital. To overcome this problem a European initiative is underway to determine what constitutes the minimal data set. Once this has been defined, commercial patient data management systems should be forthcoming. The future development of the critical care unit will then probably concentrate on interfacing these systems with intelligent software modules capable of deciding

automatically when to update data for the optimal management of the patient.

2.4 Summary

It has been shown how the history of research into artificial intelligence is intertwined with that of the development of the modern computer. The historical development of computer technology has been summarised previously (Harrison, 1986), see Figure 2.5. The Analytical Engine of 1842 comprised tonnes of brass and steel and was driven by a steam engine. Two 50-digit numbers could be multiplied together, the answer taking one minute to compute; addition and subtraction could be performed at a rate of 60 operations per second. The ENIAC machine of 1946 comprised 18,000 vacuum tubes, had dimensions of 100*10*3 feet, consumed 140kW of energy and was capable of 5,000 operations per second. The modern computer of 1988 is much smaller, uses much less power, costs much less and is capable of over 1,000,000 operations per second. It can be said with confidence that a computer revolution has taken place over the last forty years.

The intention of this chapter was to perform a historical review of artificial intelligence and show a historical development of the critical care unit. From these descriptions it can be seen how the development of the former has had an application in the latter. The use of 'high technology medicine' has had a beneficial effect on patient diagnosis, monitoring and therapy. One area of critical care which would benefit from the use of advanced instrumentation is the respiratory management of patients who require ventilatory therapy. The modern ventilator is often controlled by a micro-processor and is therefore capable of a 'smart' functionality. In the next chapter a critical review is performed on ventilatory devices which use feedback-control and techniques from the domain of artificial intelligence to demonstrate novel ways of increasing the 'smartness' or intelligence in patient management processes.

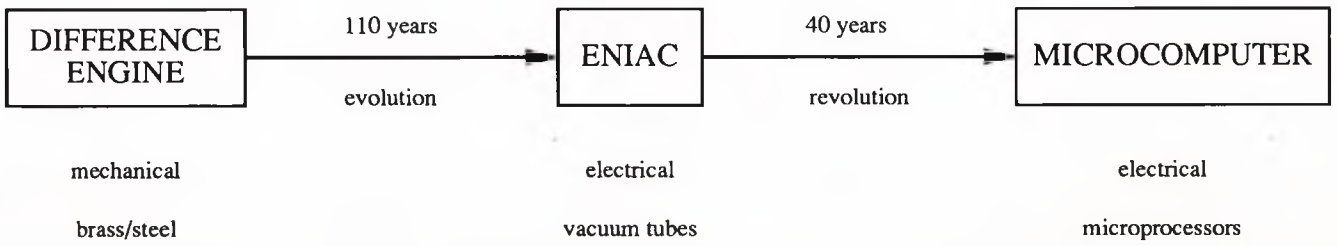


FIGURE 2.5 SUMMARY OF HISTORICAL DEVELOPMENT OF COMPUTING

3 : CRITICAL REVIEW OF A CONTROL SYSTEMS APPROACH AND AN ARTIFICIAL INTELLIGENCE APPROACH TO VENTILATORY MANAGEMENT

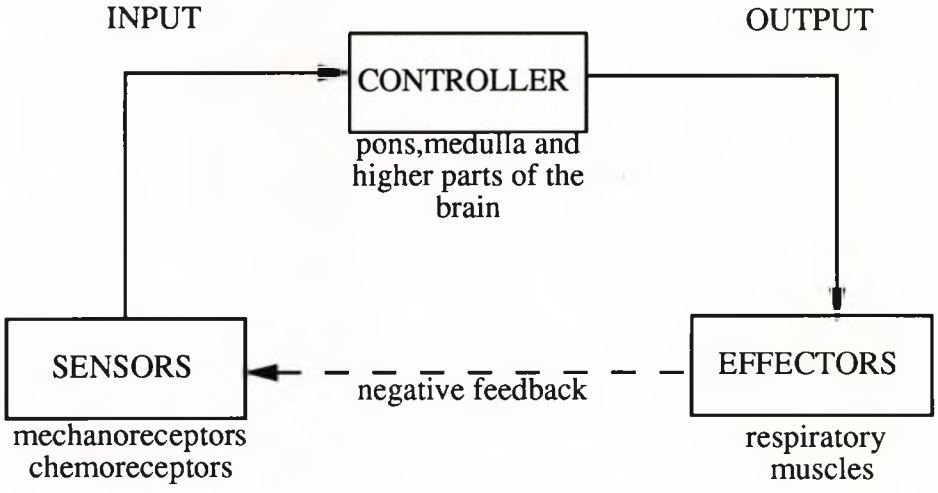
3.1 Introduction

In the previous chapter the historical aspect of artificial intelligence was given. Here the investigation of an application of this technology to the management of patients who require mechanical ventilatory support is considered. To achieve this objective this chapter critically reviews external control of the human respiratory system by use of an artificial ventilator. Two approaches to external control of ventilation are considered: the first from a traditional control systems perspective; and the second using the tools and techniques of an artificial intelligence approach.

A classical control system consists of three sub-units; the transducing element(s) from which the variables of the system are derived; the controller, (e.g. P, P+D, P+I, PID), which computes the degree of control required; and the actuator, which supplies the means by which the control action is carried out on the system. The purpose of a control system is to keep designated variables within pre-set desirable limits. In normal physiology the respiratory system has its own internal control mechanisms, the basic system elements are shown in Figure 3.1 a), with a more detailed account which includes neural respiratory control shown in Figure 3.1 b).

In situations where the physiological system which controls respiration is not intact, an alternative external controller is required. This is the role of an artificial ventilator; it takes over from the respiratory control system so as to sustain life. Such situations occur, for example, when the respiratory centre of the brain is deranged due to supression (as in drug overdose), or compression (as with a tumour). Many more examples exist which explain the dysfunction of the respiratory control system.

In Section 3.2 servo-controlled ventilator systems are reviewed. The purpose of a controlled ventilator system is to keep designated respiratory variables within pre-set desirable limits. This enables the production of a patient-specific management plan, the desired end-point of which is to return the patient to a state where spontaneous ventilation occurs. This is the point when the internal physiological control system can regain control from the external 'artificial' system.



Input to the controller is information from the various sensors. Controller output affects the respiratory muscles. Changes in ventilation are counterbalanced by the action of the respiratory muscles via a negative feedback loop.

FIGURE 3.1 a) BASIC ELEMENTS OF THE RESPIRATORY CONTROL SYSTEM
(After Figure 8.1, West, J.B., 1979)

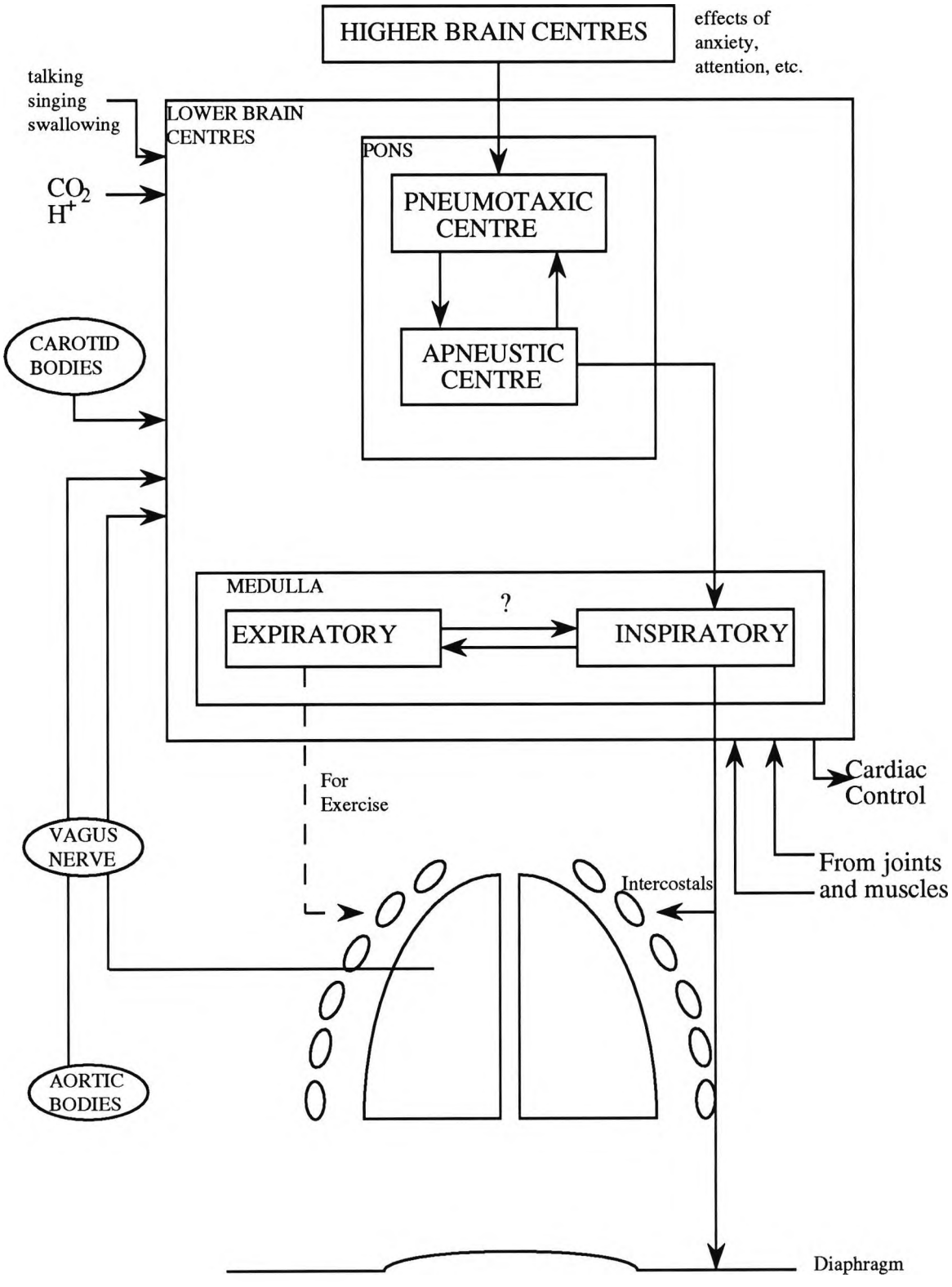


FIGURE 3.1b) BASIC STRUCTURE OF NEURAL RESPIRATORY CONTROL
(after Figure 20, Fincham and Beishon, 1973)

A difficulty encountered in servo-controlled ventilator systems is that of obtaining a reliable measure of the 'controlled' variable. For instance, clinicians would like to use PCO_2 in arterial blood (P_aCO_2) as the controlled variable, as this measure reflects respiratory status. However, to obtain this measurement an invasive technique is required, which is fraught with its own complications, although this procedure may be deemed beneficial for some patients in a Critical Care Unit. To date, end-tidal CO_2 concentration ($ETPCO_2$), measured by a capnograph, is usually used as an indicator of P_aCO_2. $ETPCO_2$ reflects alveolar concentration of carbon dioxide which is itself a function of P_aCO_2.

Section 3.3 illustrates the artificial intelligence approach for the provision of a patient-specific management plan. As well as ventilator control, the system has a domain specific knowledge-base appended to it. This knowledge-base has the function of providing intelligent advice to the user, which enables an optimal or near optimal respiratory management trajectory to be set.

3.2 A Control Systems Approach to Ventilator Management

Katona describes two reasons for developing systems to automatically control the designated respiratory variables (Katona, 1983). First, ventilator settings may have to be continually adjusted to allow optimal gas exchange to take place for the metabolic requirements of the patient. Second, it is sometimes desirable to keep some of the monitored respiratory variables constant so that interpretation of respiratory manoeuvres can be more easily made. Both of these principles are used in the systems described below.

3.2.1 $ETPCO_2$ as an Indication of P_aCO_2, (Ohlson et al., 1982)

It has been indicated previously that $ETPCO_2$ is used as an approximation to P_aCO_2. A microcomputer-based feedback control system was designed to test this hypothesis under different physiological conditions. The significance of this research is that in some disease states large differences can occur between $ETPCO_2$ and P_aCO_2 (West, 1979). As a consequence of the evaluation study performed on this system a number of automatically controlled ventilation systems which use $ETPCO_2$ as a basis for their control action become compromised.

The system designed by Ohlson and colleagues was based on a SIEMENS-ELEMA 900B servo-ventilator. The ventilator was modified to accept control signals from a computer, which exhibited control of

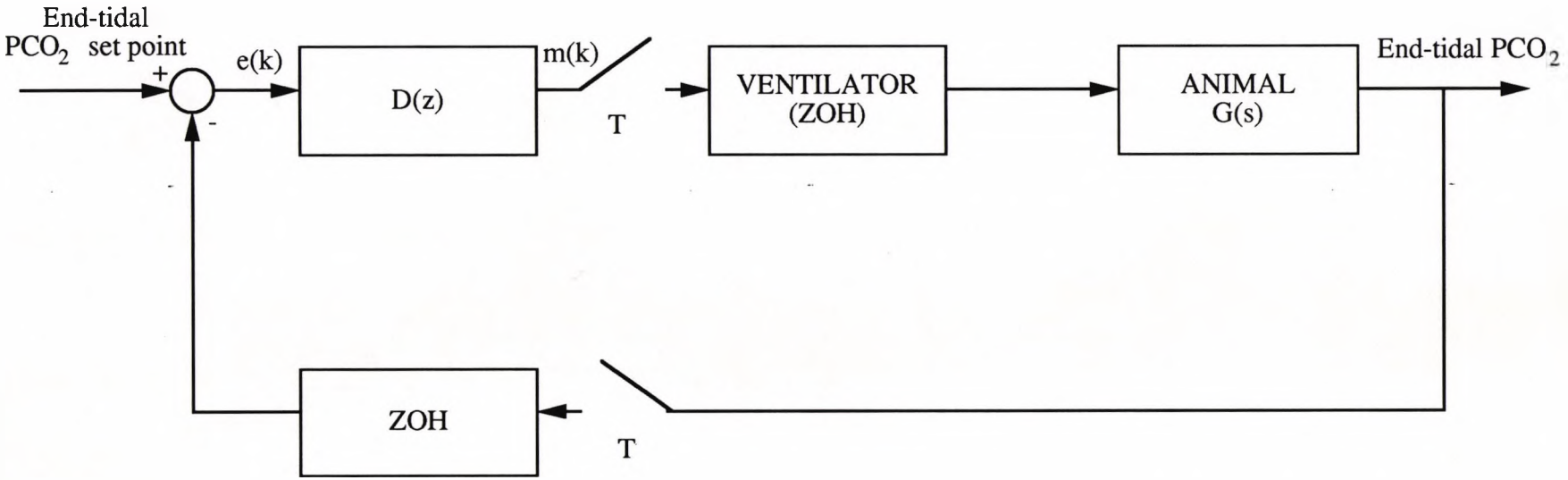
both minute volume and ventilation rate. A CO_2 analyser was used to measure $ETPCO_2$, and both monitoring and lung mechanic calculator modules were used. This configuration allowed the display of flows, pressures and mechanical factors, (for example, resistance and compliance).

This ventilator-computer system was tested in closed-loop mode and evaluated using six dogs. Although only $ETPCO_2$ was captured by the computer, P_aCO_2 was also monitored continuously. A model of the closed-loop feedback control system is shown in Figure 3.2. D(z) is the Proportional plus Integral plus Derivative (PID) controller, whose output, 'm', is the sum of the current error, the accumulated error and the change in error. These parameters are related by a set of difference equations. The sampling interval, that is, how often the physiological variables were measured, was determined empirically and set at 5 seconds.

System performance was evaluated under different physiological conditions induced by the following pertubations :-

1) $NaHCO_3$ was infused intravenously.
2) A main branch of the pulmonary artery was occluded by a Swan-Ganz balloon catheter.
3) One lumen of a double lumen endobronchial tube was occluded.
4) An air embolism was administered.

Each of these pertubations mimic a pathophysiological state, their significance can be described as follows. Infusion of $NaHCO_3$ for 10 minutes at a rate of 0.21 mEq/kg/min simulates a change in CO_2 minute production, thereby changing the metabolic rate of the animal. This situation occurs regularly in patients who are admitted to a Critical Care Unit. Occlusion of one of the main branches of the pulmonary artery effectively increases the physiological dead-space. This is of pathological significance because it simulates disease states which are caused by poor blood flow in the lungs, where physiological dead space is effectively increased due to little or no gas exchange in those areas. The left lumen of a dual lumen endobronchial tube was occluded which meant that all ventilation was diverted to the right lung. This mechanism simulated a right-to-left shunt. Air (1 ml/kg) was rapidly infused into the right atrium of the heart via one lumen of the Swan-Ganz catheter. This modelled a transient disturbance, and is also effectively another method of increasing physiological dead-space.



D(z) is a PID controller (computer software);
Ventilator acts as a zero order hold (ZOH);
The animal is considered as a continuous process G(s);
The summing junction is contained within thee computer software algorithms;
T = 5 seconds.

FIGURE 3.2 CLOSED - LOOP CONTROL SYSTEM USED BY OHLSON ET AL., (1982)

When $NaHCO_3$ was infused, the PID controller adjusted minute volume based on $ETPCO_2$ measurements, and kept P_aCO_2 within desirable limits even though CO_2 production increased by as much as 44%. However, control was not as effective when the other three pertubations were performed. One reason for this ineffectiveness could be that all three pertubations caused large changes in the ventilation - perfusion ratios, thus causing large differences between $ETPCO_2$ and P_aCO_2. If this large $ETPCO_2$ - P_aCO_2 mismatch were constant then the PID controller set-point could be altered so as to accommodate this change. The resulting change in minute ventilation could then bring the P_aCO_2 measurement back to within desirable limits. One conclusion of this work was to begin research into methods that would monitor ventilation - perfusion ratio, as this could determine the $ETPCO_2$ - P_aCO_2 relationship.

3.2.2 Automatic Control of a Ventilator Using $ETPCO_2$ as the Controlled Variable (Smith et al., 1978)

A similar control system to that described by Ohlson and colleagues (above) was proposed four years earlier. Here $ETPCO_2$ was measured via an infra-red CO_2 analyser. The value of $ETPCO_2$ was to be used to control the motor rate of a fixed-volume respirator. The CO_2 analyser - controller - respirator system operated in a closed-loop mode which allowed for the automatic control of $ETPCO_2$. The instrumentation system contained a variable gain and lag compensation network, which permitted critical damping and thus prevented oscillation.

The purpose of this system was to investigate the respiratory neural control of paralysed animals, where a constant $ETPCO_2$ was required. Tidal volume and respiratory rate (which define minute volume) were set at values necessary to provide the desired $ETPCO_2$. This measure then became the set-point value. The system operated in a way such that if there was a change in $ETPCO_2$ the controller would alter the rate of the respirator in a way to minimise the effect of that change.

3.2.3 Investigation of the Response to Hypoxia and Hypercapnia Using $ETPO_2$ and $ETPCO_2$ as the Controlled Variables (Kawakami et al., 1981)

This system was developed to control P_aO_2 and P_aCO_2 simultaneously and independently of each other. The control action alters the inspiratory F_IO_2 and F_ICO_2. The purpose of the system was

to investigate the ventilatory response of human subjects to hypoxia and hypercapnia. Using the same system it was also possible to assess the effectiveness of gas exchange while changing ventilatory pattern, and simulate the values of the arterial blood gases for specific conditions such as exercise.

The instrumentation system used in this study is shown in Figure 3.3. Expiratory gases were analysed by mass spectrometry, this is a technique capable of measuring both $ETPO_2$ and $ETPCO_2$ simultaneously. The O_2 and CO_2 controllers regulated the directions of the movement of two pulse motors, which were connected to two mixers which allowed the separate determination of F_IO_2 and F_ICO_2. Nitrogen, as well as oxygen, was supplied to the O_2 mixer, the output of which was connected to the CO_2 mixer. The gas mixture was available in any concentration between 0% to 100% for both O_2 and CO_2. The output of the CO_2 mixer was connected to the ventilator, (BENNETT PR-2), where the gases were decompressed and humidified. This gas mixture was administered to the subject via an overflow bag, J-valve and mouthpiece. If the values of $ETPO_2$ or $ETPCO_2$ were at a level to bring about a control action, under normal operation of the system the increase in F_IO_2 and F_ICO_2 occurred using a ramp function set at 2 %/min for O_2, and 3 %/min for CO_2. A provision for a more rapid change was also included where the ramp function was set at 23 %/min for O_2 and 32 %/min for CO_2. This rapid change function was included to off-set any life-threatening situations. For measurement purposes it was assumed that the end-tidal concentrations mirrored the concentrations of the gases in arterial blood.

Three different physiological conditions were induced in the subjects in order to evaluate the system allowing an insight into the relationship between end-tidal and arterial blood gas concentrations. They were:-

1) normoxia $P_aO_2 = 12$ kPa ; $P_aCO_2 = 5$ kPa
2) normocapnic hypoxia $P_aO_2 = 5$ kPa ; $P_aCO_2 = 5$ kPa
3) normoxic hypercapnia $P_aO_2 = 12$ kPa ; $P_aCO_2 = 7$ kPa

Arterial blood for the blood gas determinations was drawn from an indwelling catheter placed in the brachial artery, and analysed directly after sampling. In this way the measurements of P_aO_2 and P_aCO_2 could be compared with the end-tidal concentrations used by the control system. All measurements were made three minutes after reaching the steady-state condition.

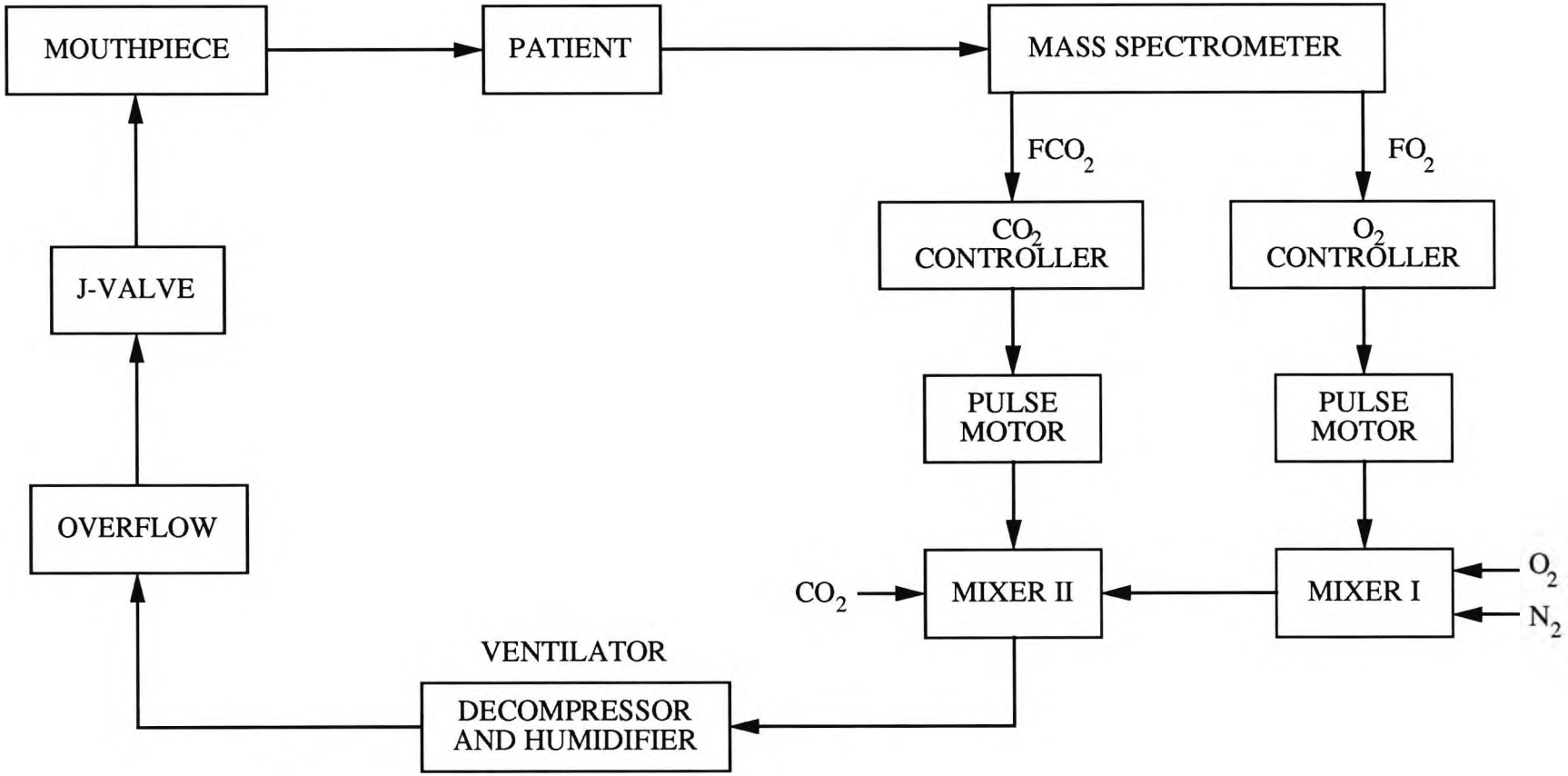


FIGURE 3.3 CONTROL SYSTEM FOR ARTERIAL BLOOD GASES

(after Kawakami et al, 1981)

The conclusions drawn from this study were that the instrumentation system performed well under normal physiological conditions, that is, when the end-tidal concentrations matched the arterial blood gas determinations. However this relationship was not as effective under the experimental conditions which introduced possible pathological factors.

3.2.4 Use of $ETPCO_2$ as the Controlled Input to Investigate Ways of Optimising Drug Therapy (Swanson et al., 1971)

This system was used for studying normal and drug-altered respiratory physiology, and combined the use of a computer-controlled breathing chamber together with a dynamic mathematical model of the physiological processes. The work is of importance because it allowed interpretation of the response of the respiratory system to the action of new drugs. The methodology used by the system hinged upon the fact that re-breathing of CO_2 was a proven useful experimental method for assessing the effects of drugs. However previous analysis had not taken into account drug action in terms of its dynamic properties, (time constants, gain, circulatory time, etc.), which was included in this study.

System input was $ETPCO_2$, which could be altered using an open-loop control system which regulated inspired CO_2. The system output was a control action which varied tidal volume or the ratio between time for inspiration to time for expiration.

A dynamic mathematical model was used to describe the CO_2 regulatory system, where the values of the measured data were used for model parameter estimation. Pertubation in $ETPCO_2$ was said to be a function of inspired CO_2, alveolar ventilation and mixed venous concentrations of PCO_2. The effective action of the drugs could be instantiated to the change which took place in the values of the parameters of the model. This system was designed so that the uncertainty factor associated with each parameter was minimised, thus helping to identify optimal drug therapy.

3.2.5 A Breath-by-Breath Method to Automatically Control a Ventilator Using $ETPCO_2$ as the Controlled Variable (Bhansali and Rowley, 1984)

A microcomputer controlled servo-ventilator system which continuously monitored $ETPCO_2$ and adjusted minute volume on a breath-by-breath basis has been proposed. This system is similar to the one described by Smith and colleagues (above). Alterations to minute

volume was the control action, taken on the basis of the measured value of $ETPCO_2$. The difference between the two systems is that Smith and colleagues alter respiratory rate to change the minute volume, whereas Bhansali and Rowley alter the tidal volume, thereby creating a different physiological dead-space.

A modified SIEMENS-ELEMA 900B servo-ventilator was used in this study, the modification being a variable orifice in the compressed air-line connected to the ventilator. This had the effect of altering air flow, which was the device used to change tidal volume. $ETPCO_2$ was monitored using a CO_2 analyser. The algorithm used for adjusting the minute volume is shown in Figure 3.4, and was capable of detecting long-term and short-term changes; that is, a ramp change between measured and set-point value of $ETPCO_2$, and a breath-by-breath change respectively. This algorithm was implemented in BASIC, and operated in the following way. A set-point of $ETPCO_2$ was entered, (S), and compared with its current value, (Z). If Z was not equal to S then an appropriate control action was taken; if Z equalled S, the measure was compared to the previous value of $ETPCO_2$, (X). A difference between Z and X brought about a control action in order to increase or decrease the minute volume accordingly. The previous sample, X, was then replaced by Z, (that is, Z was instantiated to X), and a new sample of the $ETPCO_2$ measure was taken.

This system was evaluated in an animal study by altering acid-base status. An infusion of 0.5N HCl induced an increase in minute ventilation which required control action to maintain $ETPCO_2$ at a constant value. The system responded well and in real-time, although the system response was underdamped. The authors partly attributed this to the fact that the body acts as a physiological buffer. It was suggested that further evaluation studies were required before using this system on humans.

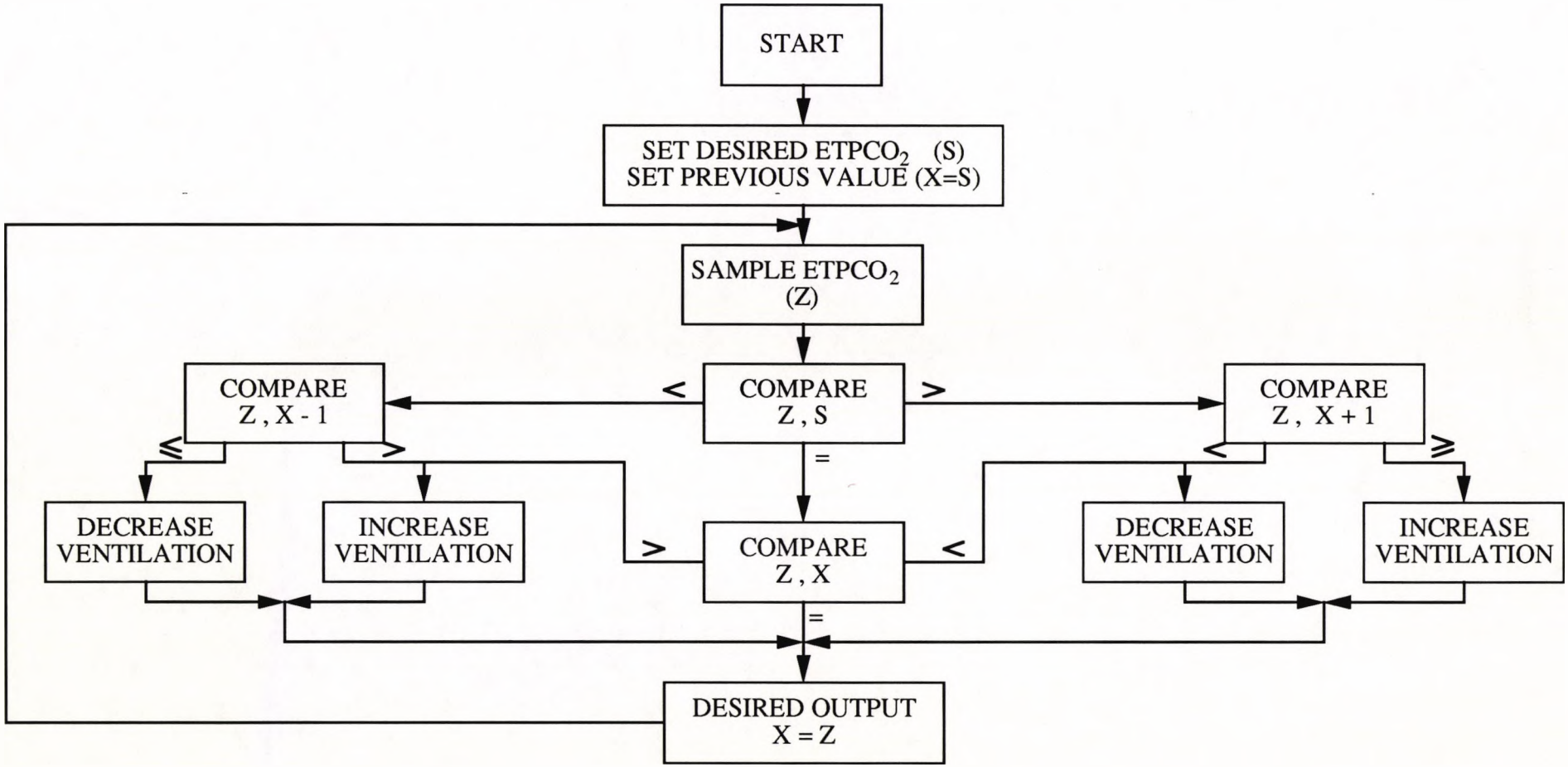


FIGURE 3.4 ALGORITHM FOR ADJUSTING MINUTE VENTILATION
(From Bhansali and Rowley, 1984)

3.3 An Artificial Intelligence Approach to Ventilator Management

This section comprises systems which use a knowledge-based approach to ventilator management. Research using artificial intelligence in this domain began in 1973, and the various methods by which intelligent systems have been used are reviewed below.

3.3.1 A MUMPS-based Ventilator Consultation System (Menn et al.,1973)

One of the first computer-based medical consultation systems was designed and developed at the Massachusetts General Hospital, U.S.A. (MGH). It comprised a suite of programs, each one dedicated to a particular patient care - data management application area. One such area was designated as the care of patients who enter the state of respiratory failure, (Menn et al, 1973). The purpose of the system was to provide a set of recommendations from which ventilator therapy could be planned in an optimal manner.

The system used a specialised programming environment, taking its name from the acronym of MGH Utility Multi-Programming System, (MUMPS). This was described as a versatile, user-friendly, text-orientated language which allowed fast access to a dynamic database. Program input consisted of answering a series of menu-driven questions. The information requested included the vital statistics of the patient, his state of consciousness, the settings of the ventilator, arterial blood gas determinations, pulmonary diagnosis and the type of ventilator and airway used. The program could cope with five types of ventilator and nine different airways. The primary data set used in the MUMPS-based consultation system is shown in Table 3.1.

Program output consisted of five areas of assessment:-

i) Oxygen Assessment
ii) Acid-Base Assessment
iii) Ventilation Assessment
iv) Weaning Assessment
v) Airway Care

Oxygen assessment yielded an 'ideal' inspired oxygen concentration made on the basis of the results of the arterial blood gases. The suggested ventilator settings were then shown to attain that particular value of F_IO_2.

Acid-Base assessment used a modified Bleich method to determine acid-base status from a linear plot of P_aCO_2 and Hydrogen ion concentration. The program was able to distinguish between pure and mixed acid-base disturbances.

PaO_2	Partial Pressure of Oxygen in Arterial Blood
$PaCO_2$	Partial Pressure of Carbon Dioxide in Arterial Blood
SaO_2	Arterial Oxygen Satuaration
$AaDO_2$	Alveolar - Arterial Oxygen Gradient
PH	pH
HCO_3	Bicarbonate Concentration
EEP	End - Expiratory Pressure
FIO_2	Fraction of Oxygen in Inspired Air
VT	Tidal Volume
VD/VT	Ratio between Dead Space and Tidal Volume
RR	Respiratory Rate

TABLE 3.1 PRIMARY DATA - SET FOR THE MUMPS - BASED CONSULTATION SYSTEM

Ventilation assessment used the fact that tidal volume, respiratory rate, and mechanical dead space all affect the P_aCO_2 of the ventilated patient. The program stipulated what tidal volume and respiratory rate were required for optimal carbon dioxide elimination.

The parameters used in weaning assessment included the state of consciousness of the patient, respiratory rate, vital capacity, P_aCO_2, pH, VD/VT, $AaDO_2$, and F_IO_2. The temporal trends of all these variables and parameters were also used in the program for weaning assessment. Computer advice was divided into three states; the first was where weaning could be contemplated with a high probability of success; the second was where weaning could be started but with increased patient observation, so that any breathing difficulties the patient might have could be dealt with quickly and efficiently; the third was where the program advised against weaning at that time.

For Airway Care, assessment recommendations were made for the maximum duration for which various airways should be used before performance of either extubation or tracheostomy.

During the initial phase of its use, all assessments made by the advisory program were evaluated by an experienced senior clinician. The system was used mainly by medical students so that they could obtain a therapeutic plan an expert would use. Thus, as well as being able to organise the vast amount of data generated by a patient in a respiratory care unit, the system could be used as a valuable teaching tool.

However, there were problems encountered when using the MUMPS-based consultation system. These could be split into two groups; operational problems, and what can be described as historical problems. Although most input to the system was quantatative and therefore communicable via a modem link, which allowed access to computer advice from terminals remote to the Massachusetts General Hospital, the original program allowed only 22 simultaneous users. This limited the utilisation of the information contained within the program. The historical problems associated with the system could be described as waiting for relevant computer hardware and software to catch up with the program specification. It must be remembered that MUMPS was designed in the early 1970s, when the advent of the age of the microcomputer was still ten years away.

The heart of MUMPS-based system consisted of a tree-structured consultation mechanism with the knowledge base defined

implicitly in the inference mechanism. This was known to be a rigid construct, now superseded by expert systems where the knowledge base and inference mechanisms exist as separate software entities. As an example of this rigidity, the validity of a MUMPS interpretation was dependent on the accuracy and completeness of the data that were entered. In more advanced systems, using expert system technology that evolved in the late 1970s, both accuracy and completeness of data are problems that can be overcome. Other lesser problems encountered during the operation of MUMPS when it first became available were the physical time to enter the data required by the system, and the time taken to obtain a printed copy of the data, (both taking up to 10 minutes).

MUMPS became a popular programming environment, the source code being exported to various other national (U.S.A.) and international centres. This popularity can be indicated by the number of MUMPS user-groups in existence, although that number is now on the decline. A possible reason for its former popularity is due to the fact that its authors were harnessing the might of the then new age of mini-computing power to a real world medical problem. With the advent of the microcomputer and an increase in sophistication of software tools available, other more advanced systems were designed to help in the care of patients who require mechanical respiratory support. However the MUMPS-based consultation system deserves its place in this review as it made a significant contribution to medical computing.

3.3.2 VM (Fagan, 1980)

VM was developed principally by L. M. Fagan in the Departments of Medicine and Computer Science, Stanford University, during the late 1970s, (Fagan, 1980; Fagan et al, 1980). The program was designed to interpret on-line quantitative data in an ITU setting, which were used to manage post-surgical patients receiving mechanical ventilatory assistance. Operation of VM depended on its ability to perform the following five tasks:-

i) to detect possible measurement errors
ii) to detect errors in the instrumentation system, and if an error was present to suggest corrective action
iii) to summarise patient-state based on the incoming data
iv) to suggest adjustments to the patient-specific management plan
v) to maintain a set of patient-specific expectations and goals for future evaluation of the program.

Central to the operation of VM was a model of therapeutic procedures carried out in an ITU, enabling interpretations of the time-varying physiological data to be made in the context of their therapeutic value at the time of measurement. Adjustments to a patient-specific management plan were made by comparing current patient data with the expectations of the measurement values of the data for that particular patient-state. Thus both patient-state trajectory and long-term therapeutic goals had a bearing on plans of patient management in this system.

To implement VM the EMYCIN shell was used. However, whereas MYCIN used a goal-directed (backward chaining) approach to form a coherent line of reasoning, VM used a data-driven (forward chaining) approach. VM had five major rule groups, the format of which is shown in Figure 3.5, with an example of an actual rule in Figure 3.6. The rule groups have the following headings; Initialising rules, Status rules, Therapy rules, Transition rules, and Instrument rules.

> INITIALISING RULES set up the initial expectations of the system defining the normal (and therefore abnormal) ranges of the values of the incoming measured physiological parameters.
>
> STATUS RULES recognised the physiological status of the patient based on incoming data.
>
> THERAPY RULES identified the readiness of the patient to undergo a new set of therapeutic goals. They established future expectations with respect to the proposed new therapy and also recommended appropriate ventilator settings.
>
> TRANSITION RULES were used when the patient transferred from one ventilator mode to another.
>
> INSTRUMENT RULES were used to define and characterise artifactual data.

On examination of these rules, and using the general formula of a production rule, that is,

IF (premise)
THEN (action)

both the premises and actions of VM rules contained three types of entity. The rule premise could be either inputed data, description of patient-state, or mode of ventilation; and the rule action could be factual conclusions, suggestions for therapeutic action, or a new set of data expectations.

RULE GROUP : Rule Name

DEFINITION : Rule Definition

APPLIES TO : Context(s) for Rule Evaluation

C : Comment

IF

M : Match (left - hand side of rule)

THEN

I : Intepretation

S : Suggestions

E : Expectations

FIGURE 3.5 GENERAL FORMAT OF A VM RULE

STATUS RULE : Status Hypoventilation

DEFINITION : Identify Hypoventilation and Recommend Correction

APPLIES TO : Volume, Assist, CMV, T - Piece

C : Should correct $ETCO_2$ for PCO_2 - $ETCO_2$ gradient

IF ONE OF

M : $ETCO_2$ HIGH

M : $PaCO_2$ HIGH

THEN

I : Hypoventilation present

S : Hypoventilation

FIGURE 3.6 EXAMPLE OF A VM RULE

VM used symbolic values in the rule premise demonstrated by looking at Figure 3.6, where $ETCO_2$ and $PaCO_2$ are described as 'high'. Numeric input to the program was converted into symbolic data via tables found in the Initialising rules. These rules defined the normal range; those values which were outside this range were divided into degrees of abnormality. These numeric - symbolic conversion tables differed according to the mode of the ventilator, so that what may have been an acceptable value in one mode of operation may be described as either 'high' or 'low' in another mode of operation.

A problem with the knowledge-base architecture of VM was that the domain-dependent knowledge was defined implicitly. This is not necessarily a disadvantage, but it is to be avoided in domains such as medicine where knowledge is constantly being updated. Past experience with production rule systems has shown that if new knowledge presents itself, it is difficult to incorporate the concomitant new rules into the system without affecting the pre-existing rules. The outcome of this is that the whole system has to be 'tuned' again, with appropriate re-evaluation and re-validation. Ironically, as Hunter has pointed out, (Hunter,1986), it is an increase in instrument technology rather than medical knowledge which makes VM redundant. For example, in modern ventilators the mode of operation is available automatically as an item of data, whereas VM has specialised Transition rules to infer the current mode of ventilation.

VM remained a developmental system, never used on-line in a clinical setting. Instead patient data were recorded on magnetic tape at intervals between 2 and 10 minutes; the tape was then removed from the ITU and the data analysed via the VM program at a remote site. So although the data used by VM were not on-line, in a sense it was still in real-time. Data which represented the complete record for one patient over 24 hours took up 15 minutes of CPU time on the computer. The evaluation of VM was performed on the equivalent of five days worth of patient physiological data, although the outcome of the study has never been recorded. One of the off-shoots of this research programme was the PUFF system for interpreting respiratory laboratory data, (Aikins,1983). To date, this is one of the few computer systems actually in clinical use on a permanent basis.

3.3.3 VQ-ATTENDING (Miller,P.L., 1984)

The ATTENDING system, implemented using LISP, was developed at the Department of Anaesthesiology, Yale University School of Medicine, (Miller, P.L., 1984). Like the MUMPS-based consultation system the ATTENDING system comprises a suite of programs. Others in the series investigate anaesthetic management, pharmacological management of patients with hypertension, and the management of patients with a suspected phaeochromocytoma. Of interest to this study is VQ-ATTENDING which investigates ventilator management. The purpose of this system is to investigate the feedback loop between arterial blood gas determinations and ventilator settings.

A feature of the ATTENDING system is that it can be described as a goal-directed critiquing system. In systems such as MYCIN, 'inference goals' are implicitly defined in the **IF.....THEN** production rules. VQ-ATTENDING uses 'treatment goals' which are defined explicitly. This has important consequences in the software design of the system, as the knowledge-base is split into two parts. Defined explicitly is the 'strategic' knowledge about treatment goals which is separate from the 'tactical' knowledge about the management choices applicable for achieving those goals, which are defined implicitly in the production rules of VQ-ATTENDING.

Program input involves describing the patient in terms of age, sex and weight; and any underlying diseases the patient has which may influence ventilator management are also requested. Data required by the system include pH, PO_2, PCO_2, minute ventilation and respiratory rate. Finally the current ventilator settings followed by the proposed new ventilator settings are required. The particular ventilator settings the system critiques are shown in Table 3.2. The treatment goals thought to be relevant for a particular patient are chosen, and are activated by the production rule inference mechanism. The critique of the management plan entered by the clinician then occurs from the perspective of those treatment goals.

In its critical analysis the program looks at treatment goals with respect to the patient's oxygenation status, then ventilation status is investigated. These treatment goals are shown in Table 3.3. More than one goal can be activated at any one time, and conflicting goals can be handled.

An attractive feature of VQ-ATTENDING is that the critique is in the form of prose, as the system has a natural language

F_IO_2	Fraction of Oxygen in Inspired Air
PEEP	Positive End - Expiratory Pressure
RR	Respiratory Rate
TV	Tidal Volume
MODE	Mode of the ventilator (A/C - Assist/Control; IMV - Intermittant Mandatory Ventilation)
DEAD SPACE	Amount of Extra Tubing in Patient Circuit

TABLE 3.2 VENTILATOR SETTINGS WHICH ARE CRITIQUED IN VQ - ATTENDING

A) OXYGENATION GOALS

1. To achieve an oxygenation (PO_2)
2. To maintain an adequate PO_2
3. To avoid the risk of potential oxygen toxicity
4. To reduce the risk of potential oxygen toxicity
5. To avoid the risks associated with high PEEP
6. To reduce the risks associated with high PEEP
7. To reduce the level of oxygenation support
8. To maintain FIO_2 at maintenance levels
9. To maintain PEEP at maintenance levels

B) VENTILATION GOALS

1. To maintain a normal PCO_2 and normal work of breathing
2. To achieve a normal PCO_2 and normal work of breathing
3. To maintain a moderate hypocapnia
4. To achieve a moderate hypocapnia
5. To maintain a moderate hypercapnia
6. To achieve a moderate hypercapnia
7. To counteract, if possible, the patient's primary hyperventilation
8. To reduce the level of ventilatory support

TABLE 3.3 TREATMENT GOALS OF VQ-ATTENDING

interpreter. This allows for a comprehensive understanding of the program output, a feature to be encouraged in all medical consultation systems.

3.3.4 KUSIVAR (Rudowski et al., 1988)

KUSIVAR is an on-going co-operative project between Linkoping University, the South Hospital in Stockholm, and a ventilator manufacturer, (SIEMENS-ELEMA). The aim of this project is to develop and clinically evaluate a knowledge-based system for ventilator management, (Rudowski et al., 1988). This includes both the monitoring of patient data and its use to semi-automatically alter patient therapy. The algorithms contained in such a control system use an extant microcomputer-based system so that relevant patient data are provided on-line. Advanced measurement techniques are used to obtain data pertaining to carbon dioxide, oxygen, and their flow characteristics within the lungs. From this the state of the respiratory system can be inferred.

Developmental work on the project is being carried out using the SIEMENS-ELEMA servoventilator 900-I connected to a SPERRY-EXPLORER workstation. It is envisaged that for clinical use the KUSIVAR system will be downloaded to an advanced PC. The user interface will allow interaction to take place in various modes, including advisory (c.f. VM), critiquing (c.f. VQ-ATTENDING), and a semi-automatic mode. For the latter mode, ethical, moral and legal perspectives will have to be taken into consideration. A new knowledge-base is being developed and the expert system within which it resides uses KEE, (Knowledge Engineering Environment). It is hoped that the use of the fully developed KUSIVAR system will allow the optimal control of therapeutic planning for patients who require mechanical ventilatory support.

3.3.5 ESTER (Hernandez et al., 1989)

ESTER is a system for ventilatory therapy advice undergoing development in the Department of Applied Physics, University of Santiago de Compostela, Spain (Hernandez et al., 1989). It is designed for use in the post-surgical recovery room where it gives clinical advice about weaning the patient from a ventilator. For the process of weaning to be successful ESTER divides the problem into four sub-tasks. These are: a semi-quanatitative estimate of the risk associated with a change in ventilatory protocol is obtained using information from the past history and diagnostic state of the patient; physiological variables are monitored in order to infer current

patient state; a therapeutic regime is recommended; and finally this recommendation is checked for clinical prudence.

The risk assessment is applied by using the APACHE-II clinical scoring criteria (Knauss et al., 1985), the outcome of which is a numerical morbidity factor. This factor can be converted to a qualitative value by using a simple look-up table. For example, a morbidity factor of greater than 80% could translate to a risk assessment of 'very high', whereas a value of less than 10% could translate to a risk assessment of 'very low', with a full qualitative spectrum between these values.

Current patient state is inferred by consideration of the current values of variables which measure haemodynamic and respiratory status, together with respiratory gas analysis and a measure of cardio-respiratory stability. This potentially rich data-set must be captured automatically in the fully implemented system, as the time scale with which ESTER operates is on a minute-to-minute basis.

On the basis of current patient state one of eight possible therapy regimes is chosen. These range from full mechanical ventilation to full spontaneous (endogenous) respiration, (see Table 3.4 for the complete range). Once the therapy regime has been suggested by the system a 'censorship' procedure is used, which ensures that the therapy protocol and the risk assessment are compatible with each other. This procedure also makes certain that 'aggresive' changes to therapy protocols are eliminated.

The knowledge base of ESTER is organised according to criteria of normality/abnormality, so that the time taken to infer patient state is optimised. Its inference engine is controlled by a set of meta-rules which select which one of the eight rule bases to apply, (see Table 3.5 for a list of these rule bases). Both backward and forward chaining of rules are employed in the execution of the program: backward chaining to obtain the data and for user interaction; forward chaining to allow the reasoning process to be elucidated. In this way a simple 'how' explanation query can be dealt with, that is, tracing the antecedants and precedants of the rules used to get to a certain point constitute the reasoning strategy.

ESTER is currently implemented on an advanced microcomputer, using the GENIE knowledge engineering tool (Sandell, 1984), which itself is a LISP-based environment.

1. CMV	100%	mechanical ventilation			
2. SIMV1	90%	mechanical ventilation	+	10%	endogenous respiration
3. SIMV2	70%	mechanical ventilation	+	30%	endogenous respiration
4. SIMV3	60%	mechanical ventilation	+	40%	endogenous respiration
5. SIMV4	40%	mechanical ventilation	+	60%	endogenous respiration
6. SIMV5	30%	mechanical ventilation	+	70%	endogenous respiration
7. SIMV6	10%	mechanical ventilation	+	90%	endogenous respiration
8. SPONT				100%	endogenous respiration

TABLE 3.4 VENTILATOR THERAPY PROTOCOLS USED IN ESTER

1. R - HEMOD	:	haemodynamic analysis
2. R - RESP	:	respirator parameter analysis
3. R - ESTAB	:	evaluation of cardio - respiratory stability
4. R - GASON	:	respiratory gas analysis
5. R - ESTAD	:	patients diagnostic state
6. R - TERAP	:	proposed therapy regimes
7. R - CONT	:	censorship procedure
8. R - CLAS	:	risk assessment

TABLE 3.5 RULE BASES USED IN ESTER

3.3.6 COMPAS (Sittig et al., 1988)

The COMputerised Patient Advice System (COMPAS) is under development at the University of Utah Scool of Medicine, U.S.A. (Sittig et al., 1988). It invokes the use of the well-known Health Evaluation through Logical Processing (HELP) system, which is an on-going database development programme that was initiated in 1973 (Pryor et al., 1983). The HELP system allows for automatic data capture in all clinical environments. It is the ultimate goal of the development programme to have a hospital-wide distributed database system. In its present implementation the data required for each patient in the Intensive Care Unit are 'on-line', with a computer terminal for access to the database at every bedside.

COMPAS was designed specifically to test the viability of using the HELP system to assist in the management of patients undergoing a controlled clinical trial. The subject of the trial is Extra-Corporeal Carbon Dioxide Removal ($ECCO_2R$) therapy together with the use of continuous positive pressure ventilation for the treatment of patients with Adult Respiratory Distress Syndrome (ARDS). This is an ill-defined syndrome which can affect severely ill patients. ARDS also has a high mortality, hence the use of extreme measures such as $ECCO_2R$ therapy, a procedure which constitutes an artificial lung. The purpose of COMPAS is to judge the effectiveness of using open-loop ventilator control and to 'lead' the physician step-by-step through the clinical protocols used. The intelligent advice is obtained from a data-driven expert system which is needed because the clinical knowledge is new and known only to a few human experts, and also in order to maintain a strict compliance with the detailed clinical care protocols established for the pertinent conduct of the clinical trial.

Preliminary evaluation of COMPAS has identified several types of problem inherent in the system. For example, one of the stimuli for this research was to implement an expert systems approach to problem solving within the HELP system with a minimum of disruption to the normal clinical routine. However, an operational problem concerning the length of time taken to access the patient database when the file was already open (due to a prior query) was identified and subsequently rectified. The summative evaluation of COMPAS using retrospective data revealed that 84% of its therapy suggestions were valid, (320/379 therapy suggestions over a time period of 624 hours).

It is evident that the success of COMPAS owes much to the clinical environment in which it is being developed and implemented.

3.4 Summary

Two approaches to the external control of the respiratory system have been considered in this chapter. First, using classical control methods and second, using an artificial intelligence approach. The critical variable for both of these methods is arterial PCO_2, which alone is effective in determining the respiratory state of the patient. The value of arterial PCO_2 is inferred from alveolar PCO_2, which itself is measured usually by the PCO_2 of the end-tidal fraction. The relationship between arterial PCO_2 and end-tidal PCO_2 is good in normal physiology but in pathological conditions the relationship may not be as adequate. Thus, classical control systems may not be founded on a true representation of pathophysiology, as shown by Ohlson and colleagues in Section 3.2.1. An artificial intelligence approach is based on a deeper knowledge-based representation, so altered data relationships caused by pathological changes can be accounted for in the rules which infer the knowledge.

Physiological measurement technology is gradually catching up with the demands made of it, for example, witness the advances made in non-invasive transcutaneous monitoring of the arterial blood gases, (Mendelson and Peura, 1984); and the use of a mass-spectrometer in a system for long-term, continuous, on-line monitoring of respiratory gas exchange, (Bertrand et al., 1986). The use of these and other research techniques will further advance the knowledge of respiratory pathophysiology in a clinical setting, benefiting the patient undergoing mechanical ventilatory support in the process.

Of the systems reviewed which employ a control systems approach, the emphasis of much of the work is to 'close' the control loop between arterial PCO_2 and the controlling ventilator settings. However recent evidence suggests that such opaque systems would meet much clinical resistance to their use. Knowledge-based ventilator systems offer an alternative, giving intelligent advice which allows the clinician to 'close' the control loop. The form of this intelligent advice depends on the patient data-set on which it is based and the sophistication of the mechanical ventilator. These topics are included for consideration in the next chapter, where intelligent system architecture is included as one aspect of a wider systems methodology for design, implementation and evaluation.

4 : METHODOLOGICAL ANALYSIS FOR MANAGEMENT OF CLINICAL INFORMATION

4.1 Introduction

The purpose of this chapter is to convey the practicalities involved in the implementation of change due to the introduction of new technology, and in particular for the introduction of intelligent measurement systems into complex data gathering environments, such as that exhibited by a Critical Care Unit. In this context, change can be defined as a transition of common practice from entering data manually to an automated means of data capture and interpretation. This can be exemplified by the Artificial Intelligent Respirator System (AIRS), a system designed to aid the clinical user in the management of patients undergoing ventilatory therapy. There is a clear need for a well-structured way of introducing this change of practice, which will enable the transformation to continue with the minimum of resistance. Such a structure can be imposed by the use of an appropriate methodology able to identify strategic points in the process of technological change. For a successful outcome the methodology must include phases for system design, implementation and evaluation. Before embarking on a design strategy there is a need to employ a systems methodology which will ensure that a proper systemic approach is adopted. There appears to be no totally appropriate methodology in existence for dealing with technological change, therefore this chapter reviews a number of relevant methodological strands and synthesises from existing approaches a methodology appropriate for the adoption of the design of a decision-support system. A major feature of the chosen medical application area is the introduction of a new generation of intelligent devices for clinical practice. The methodology to be introduced also has a general applicability within other domains where intelligent measurement has a significant role in the development of the underlying technology.

Methodologies and techniques drawn from the domain of systems engineering can be used to identify the components required for a methodology to implement technological change. Systems engineering can be defined as the domain concerned with a problem-solving paradigm for which the design of complex systems are required. From this four activities can be recognised : requirement analyses of the current and proposed systems; the development of a solution, or a number of candidate solutions; the implementation of the solution(s); and an evaluation procedure which yields value judgements at each of the

recognised decision points. For example, where a number of candidate solutions are offered the evaluation process may aid the end-user in the decision regarding which solution should be implemented.

To increase the operational success of the proposed system it may be prudent to perform a systems analysis. This can be defined as the systematic process of reasoning about a problem and its constituent sub-problems to identify what needs to be done to achieve a particular goal or sub-goal. An essential component of this analysis is communication between the system engineer and the full complement of end-users. From this the multi-perspective nature of requirements for the new system can be recognised. The analysis of the need for the new system is a key component, for if a perceived need can not be clearly demonstrated, the new system is doomed for the scrap-heap before it is even implemented.

The role of systems engineering, and especially requirements engineering, in the formation of a methodology for introducing technological change is discussed in the following section. Development of these ideas lead to a systems methodology for design, implementation and evaluation for the purpose of introducing technological change. A general format for this methodology is introduced in section 4.3.

4.2 Systems Engineering and Technological Change

Systems engineering is an attractive way of dealing with the problem of increasing rates of change due to a general advance in technology. It is a discipline where the available human, capital, and technological resources are used to ascertain that the overall objective of a system is attained in the most efficient way. Economic criteria can be used to optimise these resources, so for example, human resources can be optimally assessed in terms of economic variables. The system to which the change of technology is to apply, termed the 'system of interest' can be considered as dynamic in some instances, for example, where individual sub-systems are constantly evolving due to an increase in component technology. As systems engineering is deeply rooted in engineering design, it is not unnatural to review general system design methodologies as a starting point for the provision of a methodology for coping with technological change. As technology advances so does user expectation of the capabilities of that technology. User expectation can be said to drive user requirements, which have therefore been identified as a key

component in the design of systems able to integrate new technology. User requirements fall into two categories : functional and non-functional. Functional requirements seek the needs inherent in the process for successful completion of the task or sub-task of the system, whereas non-functional requirements impose constraints upon the system.

A conceptual model, which is a representation of all possible system states and system behaviour between states, helps in the visualisation of the functional requirements. An advantage of using such a model is that system boundaries can be defined, thus allowing the characterisation of the reaction between different environments. It also provides the opportunity for discourse between the system engineer and the eventual set of users. This enables the establishment of a definition for functional evaluation criteria of component design (Roman, 1985).

The types of constraint which can be imposed upon the system include interface, performance, operational, economic, and political factors. Interface constraints define the way in which the system interacts with the wider-system within which it resides. The wider-system is an environment which includes the full complement of system users, any other system to which the system of interest is linked, and the hardware and software with which it is implemented. Performance constraints include topics such as the computer memory requirement of the system and the type of computer necessary for successful implementation. Operational constraints include factors that may affect the smooth running of the system, such as a mis-match in the information-handling rate of each of the sub-systems. Economic constraints deal with the cost of the system, where 'cost' is liberally defined in financial and non-financial terms. Political constraints include the designation of the recognised users of the system, what level of interaction they are allowed, as well as the more divergent legal and ethical issues.

Two papers offer an outstanding introduction to the concept of systems engineering and the design process. Jenkins (1969) describes succinctly the stages involved in the development of a systems engineering project, whereas Finkelstein and Finkelstein (1983) review the methodology of design from which a rich glossary of domain terms can be taken. From this introduction a more directed path is required which leads to an analysis of system requirements. Four methodologies are investigated which fulfil the role of seeking system

requirements. First, the Structured Analysis and Design Technique (SADT) developed by SofTech, USA (Ross, 1985), which is a computer-based technique for handling the design requirements of large and complex systems. Second, a Software Requirements Engineering Methodology (SREM) is reviewed (Alford, 1985), which is being developed at the TRW Huntsville Laboratories, USA. Third, the Technology for Automated Generation of Systems (TAGS) is a methodology which uses a computer-based tool in the design process and is being developed at Teledyne Brown Engineering Inc, USA (Sievert and Mizell, 1985). Fourth, an expert system architecture is used for Computer-aided Control Engineering (CACE), being developed at the General Electric Corporate Research and Development Laboratories, USA (Taylor and Frederick, 1984).

The piecemeal approach to engineering problem-solving can be vastly improved if a systems engineering approach is taken (Jenkins, 1969). This imposes a disciplined and well-structured methodology for engineering problem-solving, although restricted to applications of a general character. This approach is not new, as the genesis of systems engineering dates back to its use in the Bell Telephone Laboratories, USA, in the early 1940s. Subsequent ideas were developed by the RCA and RAND Corporations in the United States from the early 1950s onwards. This development popularised the techniques used in systems engineering and in so doing enhanced the view of systems as a science.

The stages in development of a systems engineering project as defined by Jenkins are shown in Figure 4.1. They comprise the activities of systems analysis, design, implementation, and operation. As befits a systems approach each stage can be decomposed into several functional units. Systems design is the modelling stage within the development process. This includes models of the system of interest and models of the wider-system in which it resides. Jenkins stipulates quantitative simulation models, allowing the system of interest to be optimised and choice of 'best' system made. The future environment of the system (the wider-system) must also be forecast, allowing the proposed interaction between the two levels in the hierarchy to be modelled.

Systems analysis includes stating the objective(s) of the project, an operational analysis of the system, a cost-benefit analysis, and identification of key data which will be used to describe the performance of the system. Objectives can be defined at any level in the systems hierarchy, but usually lead to some perceived

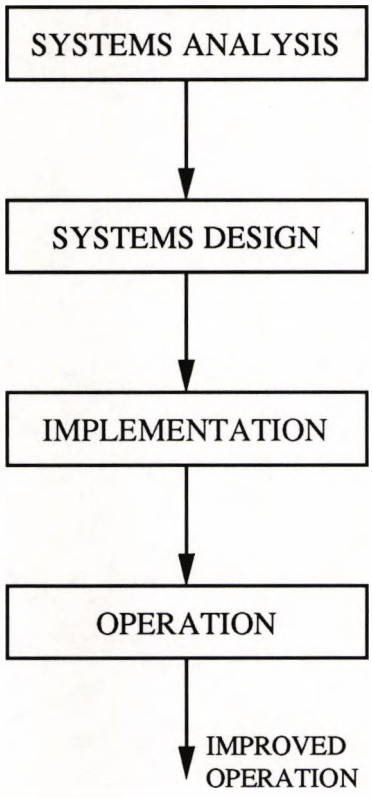


FIGURE 4.1 STAGES IN THE DEVELOPMENT OF A SYSTEMS ENGINEERING PROJECT
(after figure 4, Jenkins, 1969)

improvement in the operation of the system of interest (as depicted in Figure 4.1). To increase the transparency of the system an explanation should be included at each phase of the operational analysis. Cost benefits can take many forms; time, space and financial benefits being prime examples. Key variables should be recognised. These allow an evaluative procedure to be undertaken at a later stage in the development of the system.

The implementation stage in the development of a systems engineering project brings together the findings from the systems analysis and systems design stages. Approval is then sought from some sanctioning authority to convert the 'paper' system generated thus far into an operational system. Decisions at this stage include the choice of hardware and software to be used for implementation of the system. After implementation an alpha-evaluation should proceed, which concentrates on quality of performance rather than operational measures.

In the operation stage the fully-implemented system is handed over to the end-users. A beta-evaluation can then proceed, which includes user-orientated performance and operational measures. This is the least well understood part of the system development process as it includes subjective issues which are traditionally difficult to quantify. It is at this point in the methodology where the process can be considered as an art rather than a science.

The Jenkins methodology provides an inter-disciplinary approach to engineering problem solving and design, for as well as the engineering aspect itself other viewpoints are taken into consideration. For example, economic criteria are used (such as the appropriate use of statistical analyses and accountancy practice) to find a solution to a particular problem in systems engineering. Jenkins suggests that the most important part of systems anlysis is the attainment of a correct objective. This emphasises the role of requirements analysis to the success of the project, a topic which is discussed below.

Jenkins also suggests that design is the central activity in engineering but is rarely acknowledged as such. Finkelstein and Finkelstein (1983) try to redress this balance with their paper concerning design methodology. They describe design as :-

> "...the creative process which starts from a requirement and defines a contrivance or system and the methods of its realisation or implementation, so as to satisfy the

requirement."

(Finkelstein and Finkelstein, 1983)

The process by which design is realised or implemented follows a methodology, which is summarised in Figure 4.2. This depicts a design 'unit' which can be used at any level in the hierarchical description of a system of interest. However there are two boundary conditions: at the most fundamental level in the hierarchy the requirements which drive the design unit are primitive need statements, that is, those which cannot be further decomposed; and at the highest level of abstraction the output of the decision stage defines the information required for the creation of the system of interest.

After a requirement has been realised, whether it be from a primitive need statement or as a result of a requirement specification from a previous design stage, the first step in the design process is the gathering of information relevant to the design problem, and its organisation into a coherent form. This stage is followed by the formulation of the criteria which arise from the requirement. This defines the system specification, which may or may not have constraints imposed upon it. From this specification candidate designs may be evaluated which generate a set of proposed candidate design solutions. Analysis of these candidate design solutions yield positive and negative attributes relevant to the requirement specification. Evaluation of each candidate design solution can be measured on the basis of the value criteria previously adopted. The design methodology terminates with a decision step: the choice of optimum candidate design solution. Each stage in the methodology is iterative. For example, if none of the solutions are acceptable (the design criteria cannot be satisfied), alteration of the requirement specification is necessary. This may effect the requirement analysis of the previous design stage.

Design is described by some authors as a problem-solving exercise, an activity which has been studied extensively by Checkland (1981). He differentiates problem-solving into two paradigms : 'hard' problems and 'soft' problems. A 'hard' problem is distinguished by clear and concise system objectives taken from a well-structured problem to which a state model can be applied. The state model not only defines the current state and the desired state, but also the trajectory from one to the other. Alternatively, a 'soft' problem is characterised by objectives which are difficult to define, as the problems to which they are applied are often ill-structured. Much

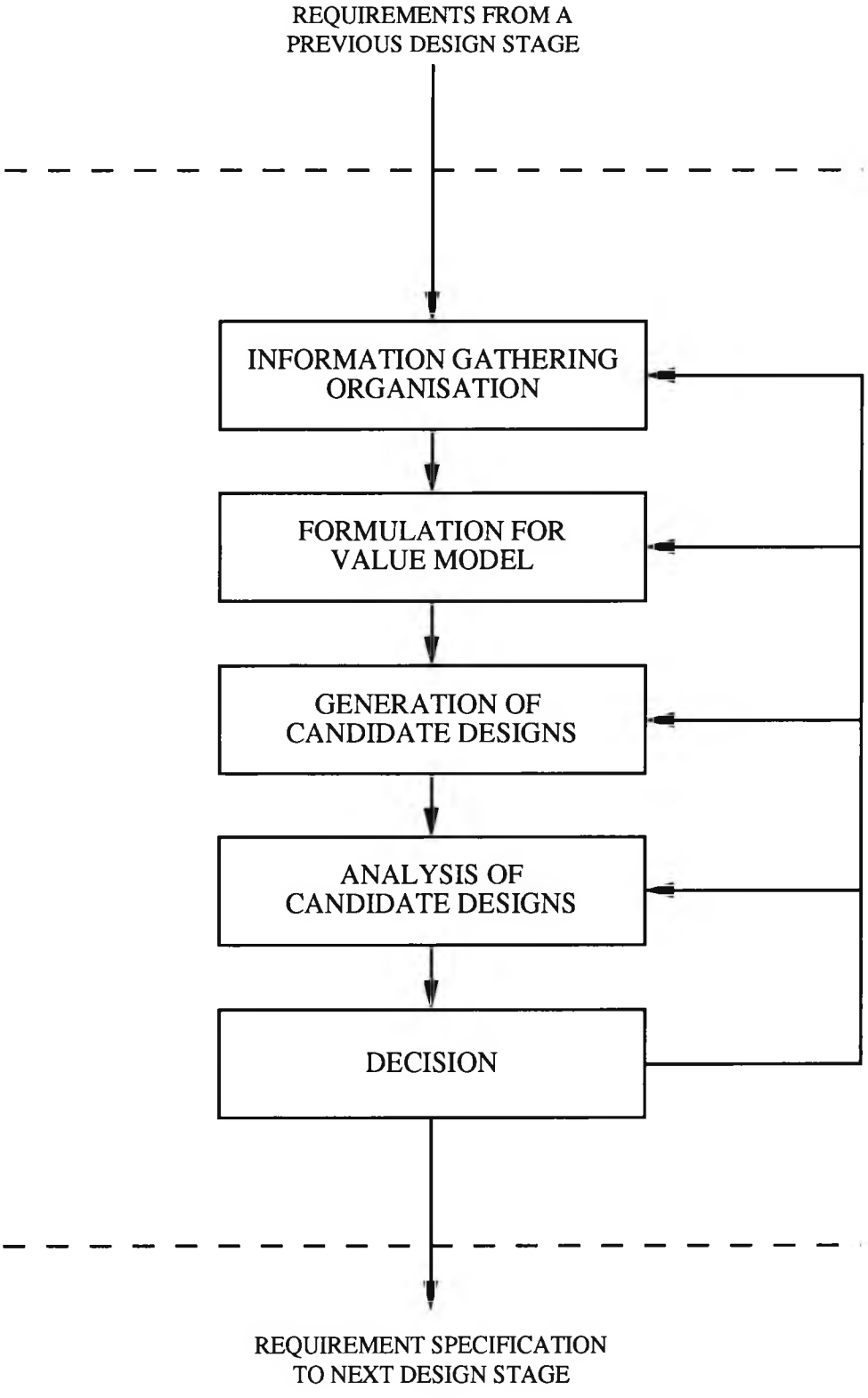


FIGURE 4.2 STAGES INVOLVED IN THE DESIGN PROCESS

(after figure 1, Finkelstein and Finkelstein, 1983)

debate has been stimulated by Checkland's assertion that 'hard' problems can be considered as special cases in the application of a soft systems methodology. It is not within the scope of this thesis to enter this debate, but the reader should be aware of its presence. Both Jenkins and Finkelstein and Finkelstein consider design engineering as a hard problem situation, with many similarities between their respective design methodologies. However the approach adopted by Finkelstein and Finkelstein centres upon the generation of design concepts for candidate solutions. Five methods of achieving this are identified: the use of existing design concepts (most design concepts are not created de novo but are developments of existing design criteria); the use of analogies, taking a design concept from one domain and applying it to another; transformation of the design concept (the principle of parsimony is often invoked); convergent generation of concepts (this uses a logical sequential process starting from the basic phyisical law underlying the design principle used); and divergent concept generation, where creative 'jumps of the imagination' and the relaxing of design constraints are employed. By using these techniques design has a central place in engineering science.

There is a similarity in the requirements analysis of both the Jenkins methodology and the design methodology proposed by Finkelstein and Finkelstein. The definition of the requirement of the system is an important phase because it defines the starting state of the system of interest. In the United States an emerging discipline has been recognised which specialises in this issue, termed 'Requirements Engineering'. An engineering (system) requirement is considered as a precise statement of need intended to increase understanding about a desired result through the use of relevant explanation. The analysis of the system necessary to achieve these statements of need requires good communication skills with the many types of user, as they are all potential sources of new requirements. Requirements engineering can be typified by the following four examplars of the emerging technology.

4.2.1 Structured Analysis Design Technique

The Structured Analysis and Design Technique (SADT - registered trademark of SofTech Inc.) was originally developed as a methodology to document the architecture of large and complex systems (Ross, 1977; Ross and Schoman, 1977). Subsequently, the methodology has been modified to allow for group decision-making strategies,

without losing the discipline that an individual disposition imposes (Ross, 1985). SADT can be split conveniently into two activities: structured analysis for which a diagrammatic language is employed; and the design technique which implements the results from the previous stage in the most effective way.

The premise which underpins the SADT methodology is that complex situations can be comprehended if they are decomposed into a hierarchy of component parts. Further, the methodology stipulates that each level in the hierarchy should not exceed six 'entries'. Each entry can be depicted by using a graphical language notation in the form of a box, (see Figure 4.3), the four sides of which represent input, output, control, and mechanism. The box represents a complete and consistent set of activities. For a given input, under certain control conditions and using a certain mechanism, a resultant output can be determined. To illustrate this point consider the well-known physiological relationship between Cardiac Output, Body Surface Area, and Cardiac Index:-

$$C.O. / B.S.A. = C.I.$$

Cardiac Index (C.I.) as the output can be determined from the Cardiac Output (C.O.) which acts as an input, and Body Surface Area (B.S.A.) being a parameter of the patient constitutes the control element. It is the arithmetic notation which acts as the mechanism and the equation as a whole represents the 'box'. It is envisaged that the input (C.O.) will itslf be an output from a previous structured analysis box.

A structured analysis 'model' is constructed when a collection of these diagrams are connected together. Models are of two diametrically opposite types: activity models and data models. Although an algebraic physiological relationship is used in the example above, most structured analysis models are verbal descriptions. In an activity model the box names are verb phrases and arrow labels are noun phrases, whereas in data models the box names are noun phrases and the arrow labels are verb phrases. An attractive feature of the SADT approach is that activity and data models can be constructed to describe the same process. The resultant cross-reference between items in the models serves to strengthen the understanding of the modelled process. Example of subject areas to which the SADT methodology has been applied are shown in Table 4.1. The design component of SADT is not yet complete, as system boundaries to which the methodology applies have not been fully characterised.

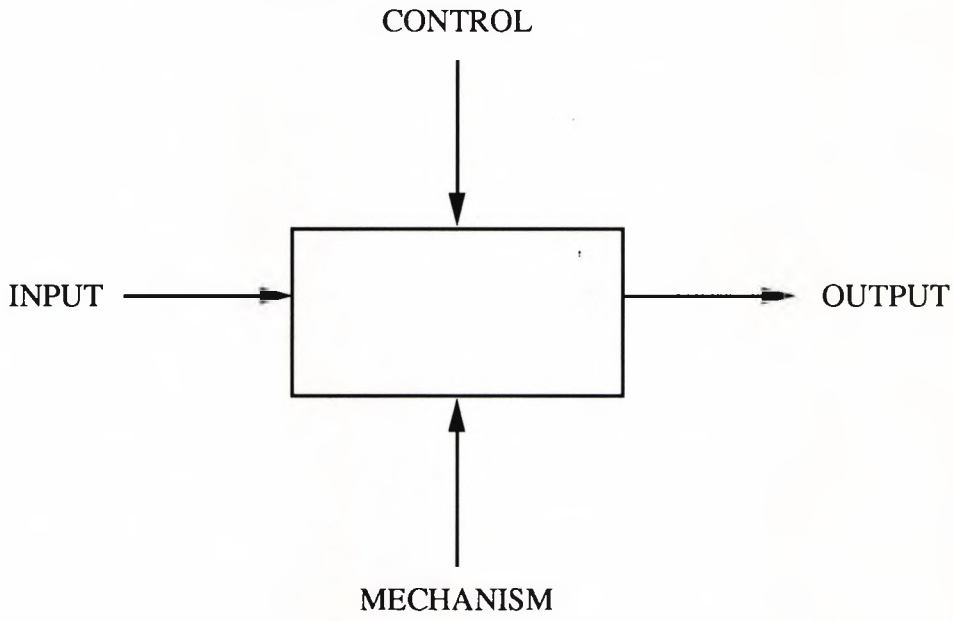


FIGURE 4.3 THE STRUCTURE ANALYSIS BOX OF THE SADT METHODOLOGY
(after Figure 1, Ross, 1985)

CATEGORY	SADT MODELS	MODEL PURPOSES
Requirements	Present Operations	Understand current procedures
	Future Operations	Visualise future procedures
	Operational System hardware/software/people	Specification and Design
Software Systems	User requirements	Includes user needs
	Functional specification	Identification of system modules and interfaces
	System/sub-system design	Top-level design
Project Management	Project operation	System development Task assignment Procedure definition Communications
Simulation	Man-machine interaction	Analysis of performance

TABLE 4.1 TYPES OF SADT APPLICATIONS
(after Table 1, Ross, 1985)

4.2.2 Software Requirements Engineering Methodology

The Software Requirements Engineering Methodology (SREM) was originally developed to manage the technical and management aspects of large, real-time, unmanned weapon systems (Alford, 1985). It has since been generalised for application to any Command, Control, Communication and Intelligence (C^3I) situation, (for example, its use in an engineering problem concerned with the aircraft industry, Scheffer et al., 1985). The technical aspects identify the activities to be performed and the subsidiary goals to be achieved before the designated top-goal can be reached. They also define any computer-based tool used to improve the efficacy of the requirements analysis. The management aspects provide techniques for project planning and evaluation of the intermediate products of the methodology, and have the overall objective of reducing costs and development time.

From these underlying principles three goals can be identified for the successful operation of SREM. The first is the development of a structured language medium in which requirements of a system can be tested for their unambiguity, testability, modularity, communicability, and design freedom. Secondly, the development of an integrated set of computer tools to ensure consistency, completeness, automatability, and correctness. Thirdly, by the interaction of the first two goals a direct method for the validation of the system requirements can be obtained.

To be testable the requirements must be specified in terms of data input and data output. The processing steps between input and output, which include the the intermediate data values and functions between the two, are termed a 'path'. Various performance criteria can be instantiated at 'validation points' on these paths. As this can invoke an increase in complexity due to the number of possible paths involved, a Requirements Network (R-Net) is established. The R-Net is implemented in a formal language, RSL, as this reduces ambiguity and provides a direct input to the support software. Automated software tools, integrated into a Requirements Engineering and Validation System (REVS), checks the requirements for completeness and consistency, maintains the traceability between the originating requirements and simulations, and generates simulations to validate the correctness of the requirements. This integrated methodology produces intermediate products which can be evaluated for completeness before advancement to the next phase is granted. Figure 4.4 illustrates an overview of the steps involved in the methodology,

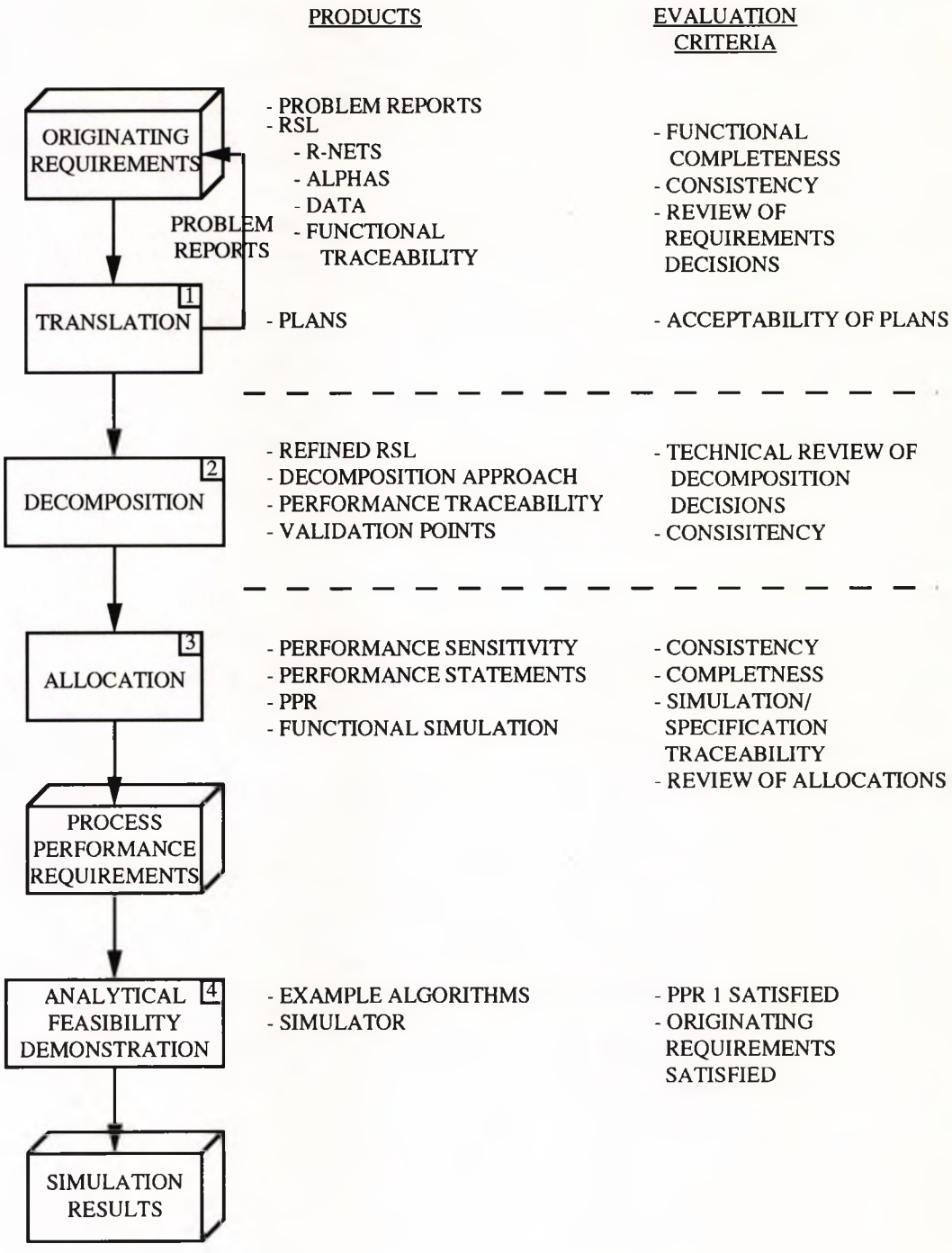


FIGURE 4.4 OVERVIEW OF THE STEPS OF THE SREM METHODOLOGY

indicating the products generated from each step and the criteria used for evaluation of their completeness.

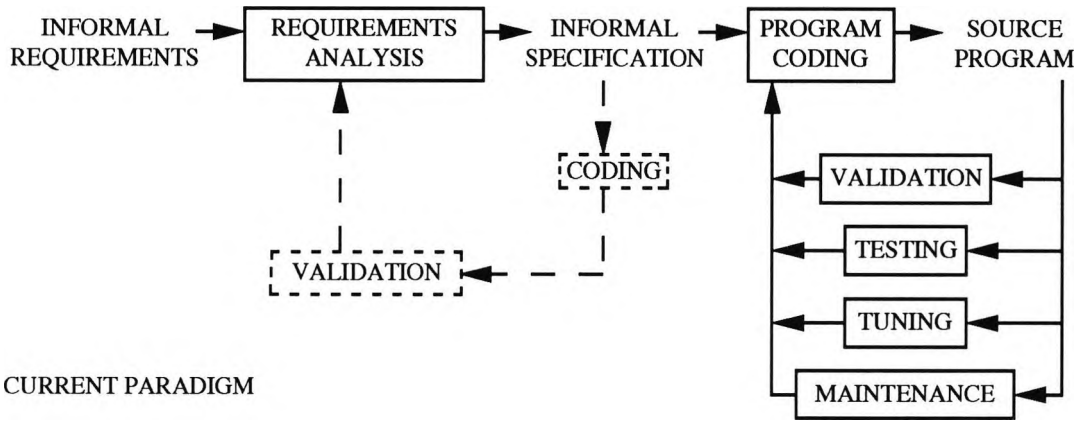
4.2.3 Technology for the Automated Generation of Systems

Technology for the Automated Generation of Systems (TAGS) was developed as a response to a perceived crisis in the existing software paradigm (shown in Figure 4.5). This crisis has occurred because of the opaqueness of software program code to the end-user. This often means that the maintenance phase of the software development cycle is ill-defined because it is at too abstract a level for the end-user to comprehend. Balzer and colleagues (1983) identified a further flaw which exacerbated this problem: there was no technology available (in 1983) to manage knowledge intensive requirements analysis and design activities. Further indications of the software crisis could be evidenced by projects overrunning in terms of both time and cost, and by the fact that the final software product did not always reflect user expectation. To counteract these problems an automation-based paradigm was proposed, as shown in Figure 4.6. The major difference between the two paradigms is that in the automated approach software maintenance is defined in terms of its underlying requirements anlysis and formalised specification. At this level the end-user can understand the issues involved in software maintenance, as they are described at a higher level of abstraction. Other differences in the two paradigms can be visualised by comparing Figures 4.5 and 4.6.

TAGS was developed to take advantage of the automation-based software paradigm (Sievert and Mizell, 1985). It is composed of three basic elements: the input-output requirements language (IORL); a computer-based software toolbox for system development purposes; and an underlying TAGS methodology.

Development of the IORL began in 1972, initiated in response to problems associated with general system propogation. In order for the language to be effective it had to meet various requirement criteria, as listed below:-

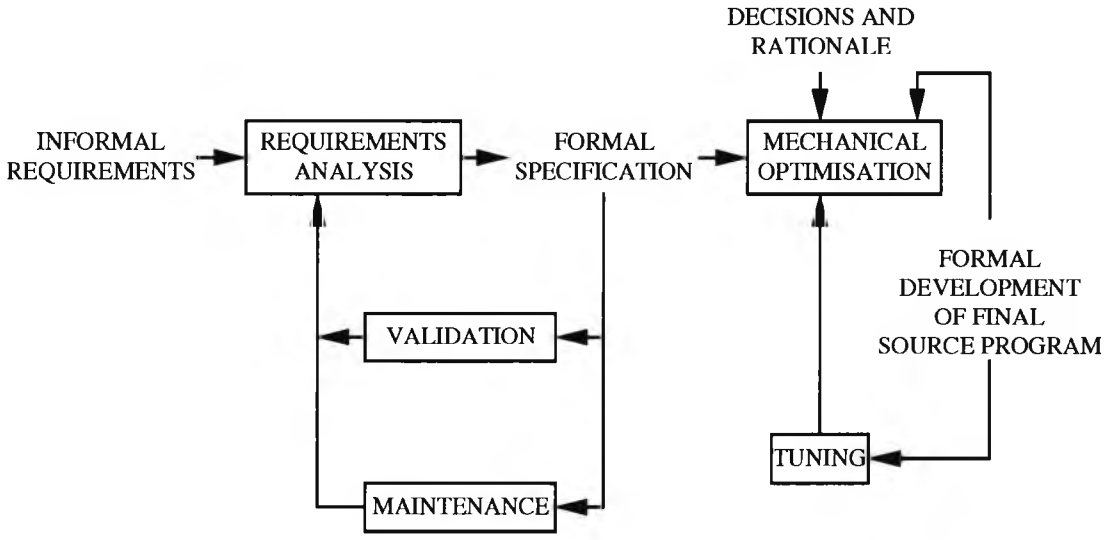
i) to enforce a rigorous methodology for system propogation
ii) to have a general applicability for all types of system
iii) it must be easy to use by a wide range of end-users; and conversely, it must be hard to mis-use
iv) it must have a user-friendly interface, allowing engineers to input system performance characteristics and algorithms in a form in which they are well-versed, (for example, the use of matrix notation)
v) from a systems perspective, it must use symbols which are derived from general systems theory.



CURRENT PARADIGM

Informal specification
Prototyping uncommon
Code validated against intent
Prototype discarded
Manual implementation
Code tested
Final source program maintenance
Design decisions lost
Maintenance by patching

FIGURE 4.5 CURRENT PRACTICE: SOFTWARE PARADIGM
(after figure 1a, Sievert and Mizell, 1985)



AUTOMATION-BASED PARADIGM

Formal specification
Prototyping standard
Specification is the prototype
Prototype validated against intent
Prototype becomes implementation
Machine-aided implementation
Testing eliminated
Formal specification maintained
Development automatically documented
Maintenance by replay

FIGURE 4.6 AUTOMATED PRACTICE: SOFTWARE PARADIGM
(after figure 1b, Sievert and Mizell, 1985)

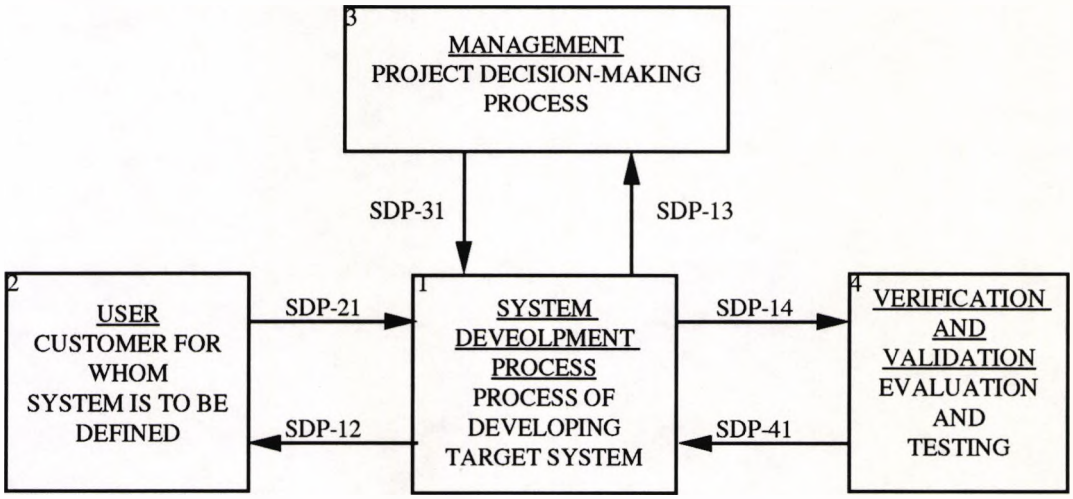
The IORL also differentiates between data flow and control flow, where data flow analysis also includes the time response characteristics of each data component. These criteria have been met, the result of which is a functional input-output requirements language. Using the IORL any individual system component can be identified, whether it be hardware, software or management aspect of the system of interest. At the functional level the IORL requires three items: that the component is a part of the system of interest and interacts through data links; that these data links are effected via an overseeing control mechanism; and that the component and its data links constitute a system hierarchy.

The TAGS toolbox comprises four tools which each exist for a specific task. The first is a data storage and retrieval package which manages the data 'seen' at any instant by the computer-based system. Second, a diagnostic analyser package checks the IORL for static errors; these include syntax errors, range errors, and errors in input or output. To illustrate the scope of TAGS there are more than 200 types of static error. Third, a simulator compiler checks the IORL for dynamic errors, generates a definition of the run-time parameters, simulates the system and processes the output data. Using this tool allows the comparison between different algorithms in the IORL. Fourth, a configuration management package uses the results obtained in the comparisons between algorithms to establish a system which is optimised at a whole-system level.

TAGS is based solidly on systems engineering principles that can be characterised by four fundamental activities:-

i) **Conceptualisation** : a conceptual model is defined which combines user concepts and requirements to form the base level for further system development
ii) **Definition** : the conceptual model is refined in terms of its function and performance requirements
iii) **Analysis** : the conceptual model is analysed to determine (at the top-most level) if it matches with the desired system; if not, the model is further refined
iv) **Allocation** : the functional and physical requirements are allocated to specific subsytems of the whole

These four activities have been incorporated into a TAGS methodological model. Figure 4.7 illustrates the development process as a part of a wider system. This includes the users, the project managers, and the processes of verification and validation. The arrows in the diagram represent flows of information which aid in the system development process. Each element of the methodological model has its



KEY:

SDP-XY: system development process - from X to Y

FIGURE 4.7 TAGS METHODOLGY
(after Figure 8, Sivert and Mizell, 1985)

own perspective on how development should proceed: the user provides the initial requirement and has a role in the reviews of system development; management have the responsibility for allocating resources to the project, including time, financial and manpower planning; the processes of verification and validation provide a formal input from which decisions on future project management can be made; and the system development process has its own intrinsic perspective on the process of developing a target system.

A TAGS development life-cycle consists of a sequence of specification-prototype validation duplexes, commencing with conceptualisation of user requirements and proceeding until the prototype matches the target system. This process is shown in Figure 4.8. On receipt of the user requirements tests are performed which evaluate their consistency, completeness and potential for implementation. The evaluation proceeds until the user requirements permit the first specification phase to be realised. This specification defines the IORL prototype which can be validated against user intentions. When the IORL of the prototype system finally matches the target system the translation phase takes place, which consists of translating the IORL into the target implementation language. This is a manual process performed at a rate of approximately 200 commands of target coding language per day per translator. Once the system has been fully implemented in the target language it can be tested using conventional software evaluation criteria.

The TAGS approach using the automation-based software paradigm has been applied to projects from the aircraft and space industries. Therefore it has potential use in any complex environment such as that exhibited by critical care medicine.

4.2.4 Computer-aided Control Environment

The development of a Computer-Aided Control Engineering environment (CACE) began in the 1960s, with a second generation appearing in the 1970s. CACE consists of four software packages, their functionalities being system identification, simulation, frequency domain design, and state space design. These packages are integrated by a unified database and command driven environment. However, a problem persists with the user interface of the second generation software: knowledge about control engineering and operation of the CACE package are assumed but are not always present in the user.

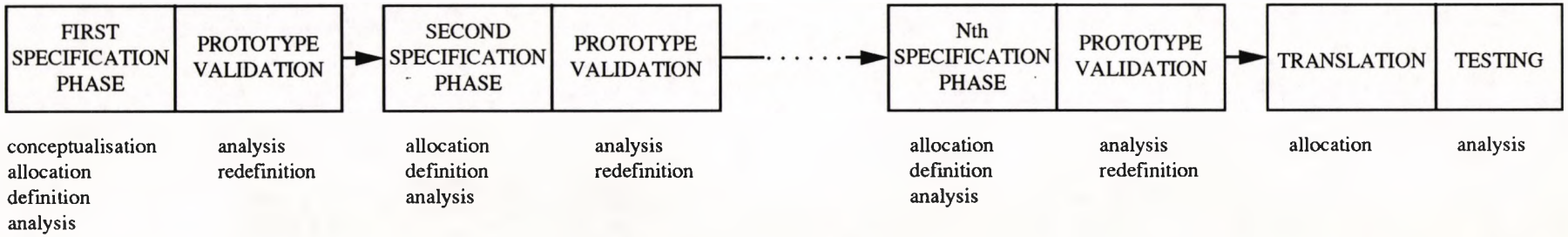


FIGURE 4.8 TAGS DEVELOPMENT LIFE CYCLE WITH SYSTEM ENGINEERING ACTIVITIES OF EACH PHASE IN DESCENDING ORDER OF IMPORTANCE

(after Figure 10, Sievert and Mizell, 1985)

Expert system technology is being introduced to alleviate this problem, allowing the less-than-expert user of CACE to work at a higher level (Taylor and Frederick, 1984).

The architecture of the resultant rule-based system is shown in Figure 4.9. This shows how the concept of having two frames (problem and solution) linked to entities external to the expert system via six separate rulebases can represent the functional structure of CACE. The problem frame can be split into three categories: the model, which describes the characteristics of the system of interest; the constraints, which describe its operational and/or functional limitations; and the specifications, which describe behaviour and performance criteria used to evaluate the desired system. The solution frame also comprises three categories: the needs, which monitor the individual requirements of the subsystem components, keeping a list of what has been achieved and what is yet to be done; the status, which provides information about how the system is developing by matching the list of what has been done to the specification of the problem frame; and a catch-all 'other' category, which contains a diverse range of constituents, for example, methodological implications derived from the order in which subsystem design is attempted.

The rulebases which link the problem and solution frames to external agencies have different functions, as detailed below.

Rulebase 1 (RB1) governs the interaction between the design engineer plus plant models external to the expert system and the model and constraint components of the problem frame. This rulebase also interfaces with a library of analytical procedures, useful for characterising factors which enhance system development. As an example of such a factor, consider the controllability and observability of a process which may be improved by a modification in the modelling phase.

Rulebase 2 (RB2) governs interaction between the design engineer and the specification component of the problem frame. This rulebase guides the less-than-expert user to describe a full specification, and checks for consistency, completeness and pragmatic utility via another set of analytical procedures. The goal of rulebases 1 and 2 is to obtain a well-formulated problem, which ensures a greater probability of success in the design stage.

Rulebase 3 (RB3) contains rules concerned with interfacing

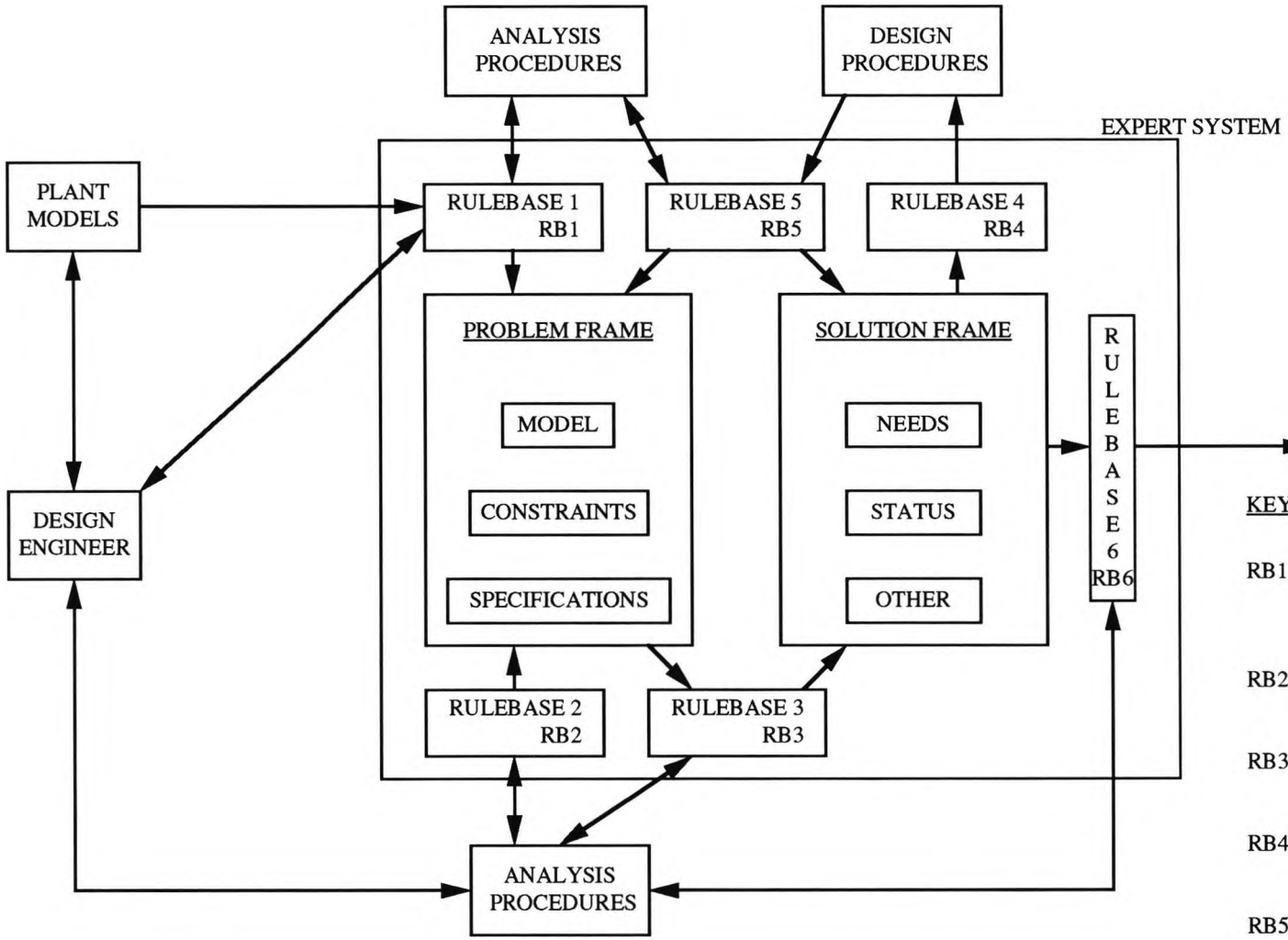


FIGURE 4.9 FUNCTIONAL STRUCTURE OF CACE
(after figure 4, Taylor & Frederick, 1984)

KEY TO RULEBASE FUNCTION:

RB1	governs interaction between engineer, plant models and model and constraint components of the problem frame.
RB2	governs interactions between design and specification component of problem frame.
RB3	governs interaction between problem and solution frame.
RB4	governs interaction between solution frame and possible design.
RB5	governs interaction between design procedures and the problem and solution frames.
RB6	governs final systems validation.

the problem to the solution frame. It deals with matters such as identifying the amount of information in the problem frame required to provide an acceptable solution for the fulfillment of a particular (sub)goal. The identification process requires appropriate analytical procedures to perform its analysis.

Rulebase 4 (RB4) governs interaction between the solution frame and the available design procedures. That is, it takes the information required to provide a solution identified using rulebase 3, and invokes a design procedure for the attainment of that requirement.

Rulebase 5 (RB5) governs interaction between the design procedures and the problem and solution frames of the expert system. As it is part of the solution, each time a design procedure is used the solution frame must be updated to reflect the techniques employed. Difficulties may arise when there is a conflict of information contained within the solution. These can be resolved by relaxing the original specifications, hence the link between RB5 and the problem frame.

Rulebases 4 and 5 represent an iterative loop. System development is cycled through this loop until all the specifications of the problem frame have been added to the list of the solution frame.

Rulebase 6 (RB6) governs system validation, this contributes towards converting the idealised system design to its practical implementation. The ultimate output of CACE is software code of the target implementation language.

The multi-rulebase structure of the CACE expert system is beneficial for application orientated end-users because it clarifies many of the conceptual aspects of the design process. This structure also enables easy access for updating the various rulebases. At a higher level, the use of expert system technology shows great potential for improving the human-computer interface, and allows a greater divergence of personnel to be involved in the design of products from the domain of control engineering.

4.3 A Systems Methodology for Design, Implementation and Evaluation

The level of complexity within measurement science is increasing due to novel techniques of measuring entities thought previously unmeasurable and a general increase in sensitivity leading

to an increased frequency of measurement. Technology has kept apace with these developments by incorporating various degrees of intelligence within the measurement device. In the first instance 'smart' measurement systems were developed which were capable of low-level processing, for instance, the conversion of data into information via simple data classification techniques. Development of these systems using advanced information technology resulted in a further transformation of information into knowledge, via appropriate explanation and/or justification, which constitutes high-level data processing. This type of system has been termed an 'intelligent' measurement system from which three measurement processes can be identified: inferential measurement; pattern cognition; and measurement as part of an integrated information system, (Finkelstein and Carson, 1986). Most intelligent systems have not appeared de novo, rather constant development of existant systems is the norm. However, any methodology used to incorporate intelligence into a measurement system should be able to handle both of these situations. The use of an appropriate methodology will provide a formal framework that will cover the entire system development process. This will ensure that system development is well-structured and that key points within the methodology can be identified for system evaluation. To introduce an intelligent system capability into a measurement system requires the activities of design, implementation and evaluation. A methodology for this purpose is described. Although key components can be found in existing (but separate) methodologies, there appears to have been no previous unifying gestalt.

A methodology to incorporate intelligence into a measurement system has its evolution in the disciplines of engineering and computing. One of the key components of such a methodology is knowledge-based system design and implementation. In the past it has been traditional to perceive knowledge-based system development as a special case of general software development. However, for the proposed methodology to assist in the technology transfer process of the whole measurement system, it is necessary to widen the system of interest. As well as software generation for the knowledge base of the application, inclusion of sensor component technology and issues concerned with the human-computer interface must be taken into consideration. These points are reflected in the methodology for design, implementation and evaluation of an intelligent measurement system, as shown in Figure 4.10. It distinguishes between processes undertaken and states achieved. A key feature is the means of

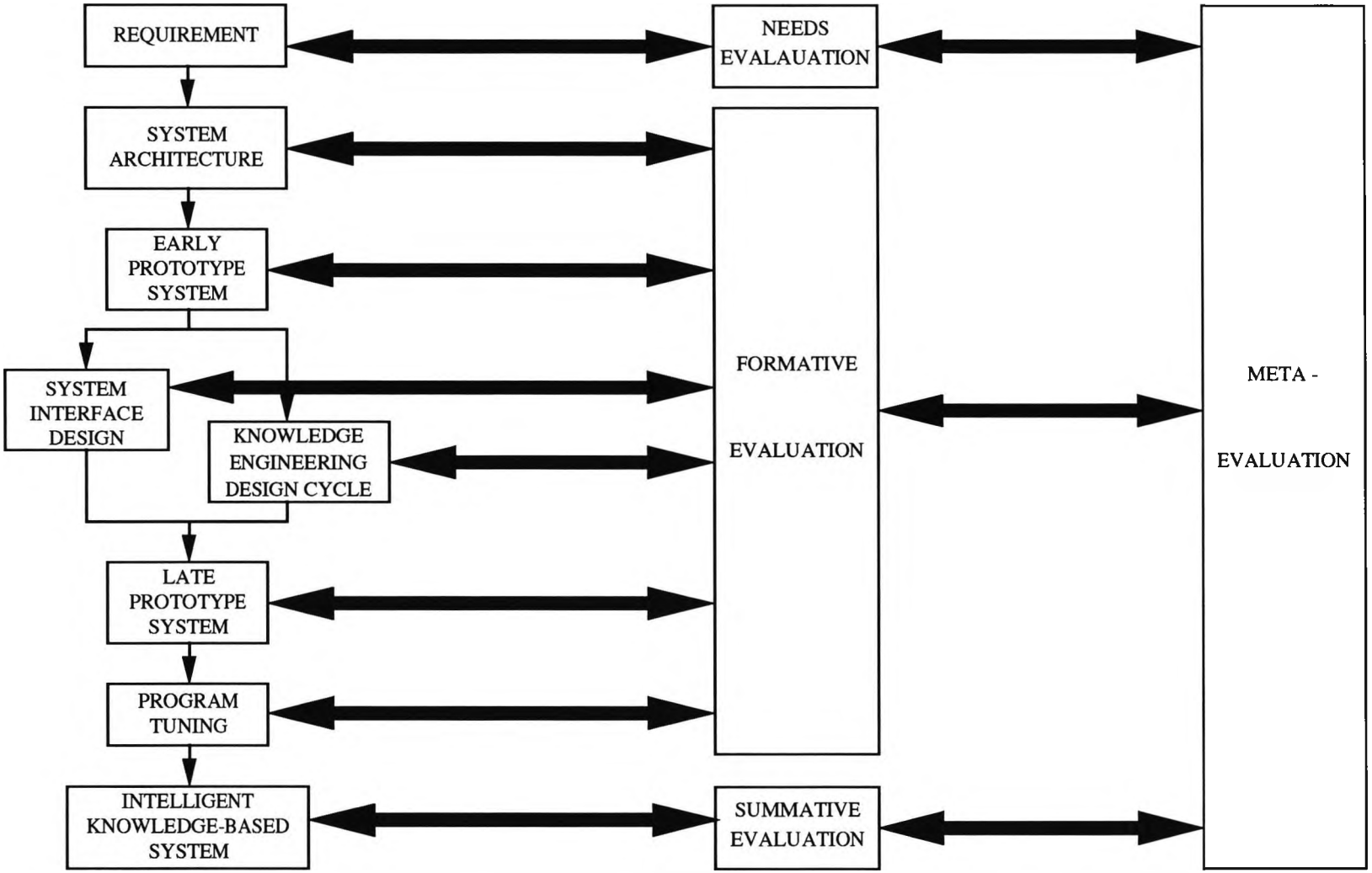


FIGURE 4.10 DIAGRAM SHOWING METHODOLOGY FOR INTELLIGENT SYSTEM DESIGN, IMPLEMENTATION AND EVALUATION

evaluation of the proposed intelligent system. To increase the chance of a successful implementation (and therefore use) it is argued that the evaluation procedure should be made as explicit as possible. It should also start at the system requirements stage of the methodology, that is, as early as possible in the development of system design.

Figure 4.10 should be viewed as a three-tier process which has been sheared sideways : the bottom tier describes the process for system design and implementation, the uni-directional arrows at this level show the path of progress; the second tier 'covers' the first, and comprises three separate evaluation procedures (needs, formal and summative); and the top-tier is the evaluation process (the 'evaluation of the evaluation') which unifies the evaluation of the system to its design and implementation. The tiers are linked with bi-directional arrows which represent both associational and iterative links. When thought of as associational links the evaluation is being used passively. For instance, it can be said that the 'late prototype system' requires a 'formal evaluation'. However, when thought of as an iterative link the evaluation process is being used actively, that is, evaluative queries are being made of the system which determines the level at which the evolving system re-enters the methodology. In this sense the evaluation of the developing system is used to fire the associations between different levels in the methodology, which includes passing through the 'meta-evaluation' stage if necessary.

Below is a brief description of each recognised stage in the methodology. These descriptions will be expanded in Part II where the methodology is applied in the context of management of patients who require ventilatory therapy.

4.3.1 Requirement

User requirements vary according to the expertise of the end-users. The requirement analysis of the proposed intelligent measurement system should reflect the different viewpoints expressed by the full range of intended users.

4.3.2 System Architecture

The proposed system should be broken down into its most fundamental components. Each component should be examined for optimal design, although this should not detract from a more holistic view of the system. Computer software and/or environments should also be included in this analysis. This would allow, for example, the reasons for use of a particular software package to be stated expicitly.

4.3.3 Early Prototype System

The early prototype system will be generally a family of 'hollow' systems. The computer will be incorporated into the measurement system, but programmed to show only a possible set of user interfaces, (windows, menus, etc.). What constitutes the intelligence of the system is yet to be implemented. This alludes to the purpose of this stage of the methodology, which is to demonstrate the user interface(s) to each of the putative end-users. This allows the system to be visualised, which may stimulate interest and allow advice for on-screen display format. Although this process may appear to have a low utility, it is in fact an important stage in the development of the system and should not be undervalued. It is at this stage of development when a rapport between the system engineer and the set of end-users can be instigated, which is essential for success of the project.

4.3.4 Knowledge Engineering Design Cycle

The knowledge engineering design cycle is the part of the methodology where the intelligent decision-making of a human expert is captured and transformed into a machine readable form. This includes the processes of knowledge acquisition and knowledge representation. The former describes the method by which knowledge is elicited from a human expert, and the latter describes how the resulting 'paper' system can be transformed into a programmable form. This constitutes the intelligent element of a knowlege-based system. The knowledge is of two types: domain (factual) knowledge; and task (operational) knowledge. Domain knowledge is, for example, the simple relationship between a measureand and its value at some instant. The value can be of a quantitative or qualitative nature. Task knowledge is concerned with two dynamic processes: firstly, how inferences between knowledge entities are made; and secondly, the software control of the programming environment, which ensures that the correct part of the knowledge base is activated at the correct time.

4.3.5 Interface Design

The interface design cycle is concerned with the fact that no part of the instrumentation is isolated from the rest. Therefore some sort of interface must exist between each element of the system. Interfaces in measurement systems are generally of three kinds: sensor to an element of a measuring machine; inter- and intra-machine

interfaces; and machine element to human user (that is, the human-computer interface). In certain circumstances three other interfaces can also be identified: the system of interest to the sensor; user-user interfaces; and the system of interest to the human end-user. The latter type of interface should not be forgotten when the system of interest is (patho)-physiological.

4.3.6 Late Prototype System

The late prototype system is the culmination of the states achieved and processes involved thus far in the methodology. When the expert responsible for the intelligence embedded within the system views the late prototype system, valuable insights of how the system can be improved can be gained from his feedback. The suggestions may vary according to the success of the implementation, but whatever the outcome the system is still evolving at this stage, which illustrates the highly iterative nature of the methodology.

4.3.7 Program Tuning

Program tuning is different from the iteration between the knowledge engineering and late prototype system stages of the methodology. Whereas in the latter the contents of the knowledge base can change, program tuning deals with a static knowledge base, but certain parameters within it may change for correct system operation. For example, the values at which patient alarms are set may be at an inappropriate level and thus have to be changed.

4.3.8 Intelligent Knowledge-based System (IKBS)

To achieve an operational IKBS is the goal-state of the methodology. The functional attributes of the final system must be able to integrate its sub-system components in context.

4.3.9 Needs Evaluation

The evaluation of the need of the system is associated with the initial requirements stage of the methodology, and has as its most fundamental evaluative query, "What are the requirements of the system and how can they be met?". One of the precursors necessary to answer this query is a functional analysis of the system about to be replaced (if one exists). This enables constraints imposed on the system to be recognised and dealt with. It also allows for a conceptual model of information flow within the system, which not only identifies the type of information used but also identifies the information sources and receptors.

4.3.10 Formative Evaluation

Formative evaluation describes the process which asks how the system can be improved whilst under development. Thus, all stages in the methodology except the initial requirements stage and the final goal-state are associated with formal evaluation. To cope with the different levels of development which exist, the formative evaluation must be wide-ranging. For instance, at one level a query may be, "Is the multi-perspective nature of the system maintained?", while at another the query may be, "What metrics are used to describe the efficacy of the software?". Some evaluative queries are effective at all levels, for example, "To what extent has a specific sub-goal been attained?". These queries will vary from system to system, dependent on the specific domain.

4.3.11 Summative Evaluation

Summative evaluation is the type of evaluation which is most often quoted in the literature. The goal-state is associated with the summative evaluation, which is concerned primarily with questions of standardisation, effectiveness and comparison (see Figure 4.11). The outcome of a summative evaluation determines if further development of the system is necessary. In some cases an external evaluation (beta-testing) should be performed, which involves allowing peers to have access to the system for the purpose of its independent assessment.

4.3.12 Meta-Evaluation

The meta-evaluation describes the 'evaluation of the evaluation'. It is concerned with the query, "How has the evaluation process proceeded?". This query determines the outcome of each stage in the evaluation process. The meta-evaluation can be viewed as the phase in the methodology which describes the degree of success of a particular system in reaching its pre-determined goal. For this reason its outcome can be used to ask the ultimate question in project management: "Should development of the system proceed?".

4.4 Summary

Commonalities in the analysis and design techniques reviewed in Section 4.2 are that they have all been developed over a number of years by well-funded industrial and/or university concerns. They are also only applicable to large-scale projects. To take advantage of the apparent technology gap at the micro end of the analysis and design spectrum, a methodology which incorporates system analysis, design and

STANDARDISATION	:	Does the system meet a set minimum standard for accreditation?
		Are Standard inter-machine interfaces used?
EFFECTIVENESS	:	How effective is the system in attaining its goal?
		What are the effects of the system on the subject of interest?
		What are the effects of the of the system on the end-user?
		What is the cost-effectiveness of the system?
		How does the system compare to the system it is replacing? (if appropriate)
COMPARISON	:	How does the system compare to other similar systems?

FIGURE 4.11 THEMES INVOLVED DURING SUMMATIVE EVALAUATION

implementation has been proposed. The central tenet on which the methodology is based is that it will assist in the technology transfer process from the laboratory to the real-world.

As the review of requirements engineering continued, emphasis changed to a computer-aided development model. These computer-based development processes are perceived in various ways, ranging from acting as an organisational aid to being used as a source of knowledge. This wide range of techniques is indicative of the comparative youth of the subject area. The manual methodology introduced in Section 4.3 goes further than system analysis and design, as implementation of the candidate design solution and an explicit evaluation procedure are featured. This illustrates the increased scope of the proposed methodology when compared to those used in conventional software development.

This chapter completes Part I of this thesis. The historical context of both the medical domain of interest and intelligent computer-aided decision support systems have been discussed. This allows knowledge of the level of the type of solution being sought. An artificial intelligence and control systems approach to the management of patients on ventilators has been included to focus upon subjects and techniques to be used in Part II. Finally, a novel methodology has been described, which is to be imposed upon the development processes necessary to create an intelligent measurement system. Thus, the research tools are in place, ready for exploitation by a suitable application: this is described in Part II.

PART II

5 : SPECIFICATION FOR AN ARTIFICIAL INTELLIGENT RESPIRATOR SYSTEM

5.1 Introduction

The methodology developed in Section 4.3 is used to impose a framework for the introduction of an intelligent computer-based system to aid in the management of patients who require ventilatory therapy. Requirements for the introduction of an intelligent system capability and the proposed system architecture can be combined to produce a 'working' specification. This constitutes the design phase of the methodology. Essential components of the design phase of any computer-based system are the choices of hardware and software to be used for its implementation. The instrumentation system to be described is being developed and implemented in a microcomputer environment using PROLOG as the language base. PROLOG is becoming increasingly popular as a means of incorporating intelligence into a system. To illustrate this finding, a brief section in this chapter is devoted to its fundamental functionality. To complete the background study necessary to understand the complexities of the application domain a brief section which explains the pertinent respiratory physiology is also included.

The Artificial Intelligent Respirator System (AIRS) is a patient management system designed specifically for use in an adult Critical Care Unit. AIRS is one element of a wider integrated information system able to assist clinical and nursing personnel with all aspects of management of the critically ill patient (Carson et al., 1988). The system combines a microprocessor controlled ventilator and intelligent microcomputer-based advisory system from which a wealth of patient data are produced. Three degrees of intelligence are exposed in the data analysis module of AIRS: the use of context sensitive information processes; the use of pattern matching algorithms; and the use of expert system technology.

The specification of AIRS defines the design phase of the methodology introduced in the previous chapter. To complete this assignment Section 5.5 introduces the needs evaluation associated with the requirement of the system.

5.2 PROLOG : An overview

PROLOG is an interactive language designed primarily for symbolic, non-numeric data processing. It is based on the predicate calculus, a powerful subset of classical logic for the definition of relationships between data terms (Roussel, 1975). A major attraction

of programming in PROLOG is that the code generated is both concise and readable, and therefore more understandable than equivalent code written in conventional programming languages, (for example, FORTRAN, C and PASCAL). PROLOG programming has minimal syntax, indeed it is possible for programmers to define their own syntax for specific applications.

The basic structure in logic programming is the 'term'. A term can be a constant, a variable, or a compound version of itself. A constant is either an integer or an atom (beginning with a lower case letter), whereas a variable begins with an underscore character or upper case letter. PROLOG programming consists of three main tasks :-

i) declaring facts about objects and their relationships
ii) defining rules about objects and their relations
iii) querying objects and their relations

Consider the following example shown in Table 5.1, in which facts are declared that define the cardiac_output and body_surface_area of different patients, where a rule is defined for obtaining cardiac_index from the given facts, and a query used to find patients with a similar cardiac_index, where 'similar' is defined as values within +/- 5% of each other. This type of query may be useful when classifying patient cohorts for a drug trial.

```
facts
cardiac_output(patient1,8.25).      body_surface_area(patient1,3.380).
cardiac_output(patient2,5.86).      body_surface_area(patient2,1.139).
cardiac_output(patient3,2.52).      body_surface_area(patient3,8.708).
cardiac_output(patient4,2.12).      body_surface_area(patient4,3.609).

rule
cardiac_index(Patient,Cardiac_index):-
cardiac_output(Patient,Cardiac_output),
body_surface_area(Patient,Body_surface_area),
Cardiac_index is Cardiac_output / Body_surface_area.

query
answer(Pt1,Cardiac_index1,Pt2,Cardiac_index2) :-
cardiac_index(Pt1,Cardiac_index1),
cardiac_index(Pt2,Cardiac_index2),
Cardiac_index1 > Cardiac_index2,
20 * Cardiac_index1 < 21 * Cardiac_index2.
```

TABLE 5.1 PROLOG representation of Cardiac Index

The syntax used in this example can be used to describe its mode of operation. The 'cardiac_output' and 'body_surface_area' predicates have four clauses. Each clause can be interpreted in the following way:-

cardiac_output(patient1,8.25). ------> the cardiac_output of patient1 is 8.25 litres

In the clause, 'patient1' is written with its first letter in lower case to denote that it is a constant. If an upper case initial letter were used denoting a variable, the interpretation of the clause would be meaningless.

In the 'cardiac_index' rule the ':-' symbol is interpreted as 'IF', and the commas are interpreted as 'AND'. This rule also illustrates the principle of unification which is often used in PROLOG. Unification embodies pattern matching that can be applied to data structures of any complexity. To fire the 'cardiac_index' rule the 'Patient' argument of the 'cardiac_index' clause must be instantiated. If 'Patient' is instantiated to 'patient1', then the cardiac_output and body_surface_area clauses can be fully unified, from which 'Cardiac_index' can be instantiated, allowing the 'cardiac_index' rule to succeed in the process.

To fire the query the patients whose cardiac_index are to be compared should be entered by the user, for example :-

?-answer(patient1,Cardiac_index1,patient4,Cardiac_index2).

The 'cardiac_index' rule is called twice, the first time to determine the cardiac_index of patient1, and the second time to determine the cardiac_index of patient4. A simple numerical algorithm in the query clause then checks to see if the results are within 5% of each other.

PROLOG has many features which can be considered as unique advantages when compared with conventional programming languages. For example, the following features are all used in the implementation of AIRS :-

* clauses can represent data or rules and can be treated as arguments to higher level clauses allowing data to be manipulated by appropriate meta-clauses which effect software control

* software procedures can be multi-purpose and have more than one input and output

* required iteration (via backtracking of goals) can be handled and controlled by a special purpose 'built-in' predicate, (the 'cut' operator)

* incomplete data structures are allowed, using pattern matching to unify variables

* for database applications the terms themselves define the record structure; any number of fields are allowed, and there is no restriction imposed on field-type

* a declarative clausal form is allowed as well as the more normal procedural form

This last feature is important as it uses the relational nature of PROLOG programming to determine the declarative meaning of WHAT the output of the program will be. This can be compared to the procedural meaning, which is concerned with HOW the output of the program is obtained.

PROLOG has been applied to many programming tasks from a wide spectrum of domains. For example, understanding natural language, mathematical logic, symbolic equation solving, and many areas within the field of artificial intelligence. PROLOG was also chosen as the implementation language for the Japanese Fifth Generation Computer System initiative. This necessitated software able to cope with highly parallel computer architectures; the logic and relational nature of PROLOG met this requirement.

There are many versions of PROLOG available, from main-frame to microcomputer implementations. AIRS has been developed and implemented on a standard microcomputer using PROLOG-2 (Chemical Design Ltd.). This version of PROLOG was chosen because it offered over 300 'built-in' predicates (including predicates for creating a windows environment), code could be interpreted and compiled, and it conforms to the (pseudo)-standard of Edinburgh syntax.

5.3 Respiratory (patho)physiology

The respiratory system is critical for immediate survival. Air contains a number of gases, the two which contribute to the maintenance of living organisms are oxygen and carbon dioxide. The respiratory process can be viewed from two perspectives : external respiration, which describes the exchange of gases between the lungs and the bloodstream; and internal respiration, which is the exchange of gases in solution, between the bloodstream and cells. Thus, the

overall purpose of respiration is to acquire oxygen which is eventually distributed to the cells for metabolism, and to eliminate carbon dioxide which is accumulated from the effects of cellular metabolic activity. The metabolic utilisation of oxygen can be described by the following mass-balance equation:-

FOOD + OXYGEN ----> CARBON DIOXIDE + WATER + by-products + ENERGY

The rate at which the products of the right-hand side of the above equation are formed from the initial components on the left-hand side, termed the metabolic rate, differs from person to person. For example, the metabolic rate of a person lying motionless in a Critical Care Unit is much less than that of an athlete, and therefore their respective respiratory demands are also very different.

Artificial mechanical ventilation is used when patients are unable to breathe spontaneously. To understand the reason for mechanical replacement therapy consider the physiological system it is to supersede. The respiratory system can be arbitrarily divided into four functional units, as follows:-

i) Airflow mechanics; this incorporates movement of the chest wall under the influence of the pressure difference between internal and external environments, causing air to be drawn into the alveolar spaces, as well as the physical properties of the lungs and air themselves.

ii) Blood flow mechanics; this describes the distribution of blood in the lungs.

iii) Diffusion and Gas exchange; this describes the transfer of respiratory gases through the alveolar membrane, and the interacting effects of blood and airflow on oxygen and carbon dioxide concentrations.

iv) Regulation of respiration; this describes the control mechanisms involved in the respiratory process. The regulation of breathing can be described by two parameters, respiratory rate and tidal volume. Respiratory rate is the number of breaths per unit time, and tidal volume describes the amount of air inspired during normal breathing. Normally, these two parameters are controlled by the nervous system and a homeostatic mechanism which monitors the partial pressure of carbon dioxide in the bloodstream.

The first and the last of these functions are replaced when there is no endogenous respiratory function, that is, when the mechanical ventilator is in Continuous Mandatory Ventilation mode (CMV). The machine can be preset to deliver the required respiratory rate, tidal volume and air/oxygen mixture in the correct pressure-time cycle. When limited physiological control returns, the ventilator can be switched to Synchronised Intermittant Mandatory Ventilation mode (SIMV), which allows a preset ratio of spontaneous to ventilator-induced breaths. To wean the patient from the ventilator this ratio is gradually changed until the patient is breathing spontaneously and independent of the mechanical assistance. For successful ventilatory therapy the physiological mechanisms which contribute to the other functions of the respiratory system must remain intact. As indicated, the parameters set by the ventilator can influence the amount of venous blood oxygenation, and can therefore indirectly contribute to the control of metabolic rate.

5.4 Design Specification for AIRS

5.4.1 Requirement

A question of fundamental importance to be asked before any system development work commences is:-

"Is there a need for the proposed system, or would it be, perhaps, an example of unnecessary technology which would be infrequently used?"

At the highest level of definition the purpose of AIRS is to assist in the management of patients who require artificial ventilation. This group of patients are often sub-optimally managed (Sittig, 1988), therefore there is a need for a system such as AIRS.

Pragmatically, an evolutionary system must be better than the system it replaces, otherwise it will remain redundant technology. In the context of AIRS the term "better" has two meanings: first, in the operation of the system; and second, the system must lead to enhanced patient care. The first of these meanings depends upon the 'friendliness' of the Human-Computer interface. To be 'better' this interface must be attractive to use and quick to learn and exploit. The ability to enhance patient care depends upon how the user perceives the change of working practice instigated by the new technology. When a computer-based data-logging device (the data capture functionality of AIRS) replaces the manual system employed currently, much of the clerk-type role of the nurse will disappear,

freeing time for more valuable nursing activities. The need for automatic data-logging is emphasised by a clear trend in the number of parameters included in an intensive care primary data set,which has increased approximately threefold in sixteen years (Price and Mason, 1986). Nurses record 45% of these data, the rate of which can be up to 2000 items per patient per day.

The advisory mode of AIRS can be used as a means of dealing with complexity. In respiratory critical care there is often an overload of sometimes conflicting data. A computer-based advisory system can organise the patient management options into a suitable format for selection. Such a system would also serve a useful role for educational and training purposes.

5.4.2 System Architecture

AIRS consists of four functional units (patient, operator, ventilator system and computer system) which can be further sub-divided into constituent subsystem elements, (Figure 5.1).

The **patient** can be considered as a complex physiological super-system, whose constituent systems are interlinked in such a way as to optimise the process for sustaining life. Of interest in this study is the respiratory system.

Each category of **operator** will have a different level of system interaction depending on their status. The users of the system will include doctors, nurses, paramedical staff, system engineers, and medical students and student nurses. A brief indication of system interaction at each level is given below.

* Doctors
    + to enter the initial settings of the ventilator
    + to enter the initial alarm settings
    + to enter any changes in ventilator settings as the patient progresses
    + to enter any change in alarm settings
    + to delegate any of the above to another system user
* Nurses
    + to enter data via the computer keyboard or alternative means of data entry
    + to be aware of the meaning of the audible and visual alarm

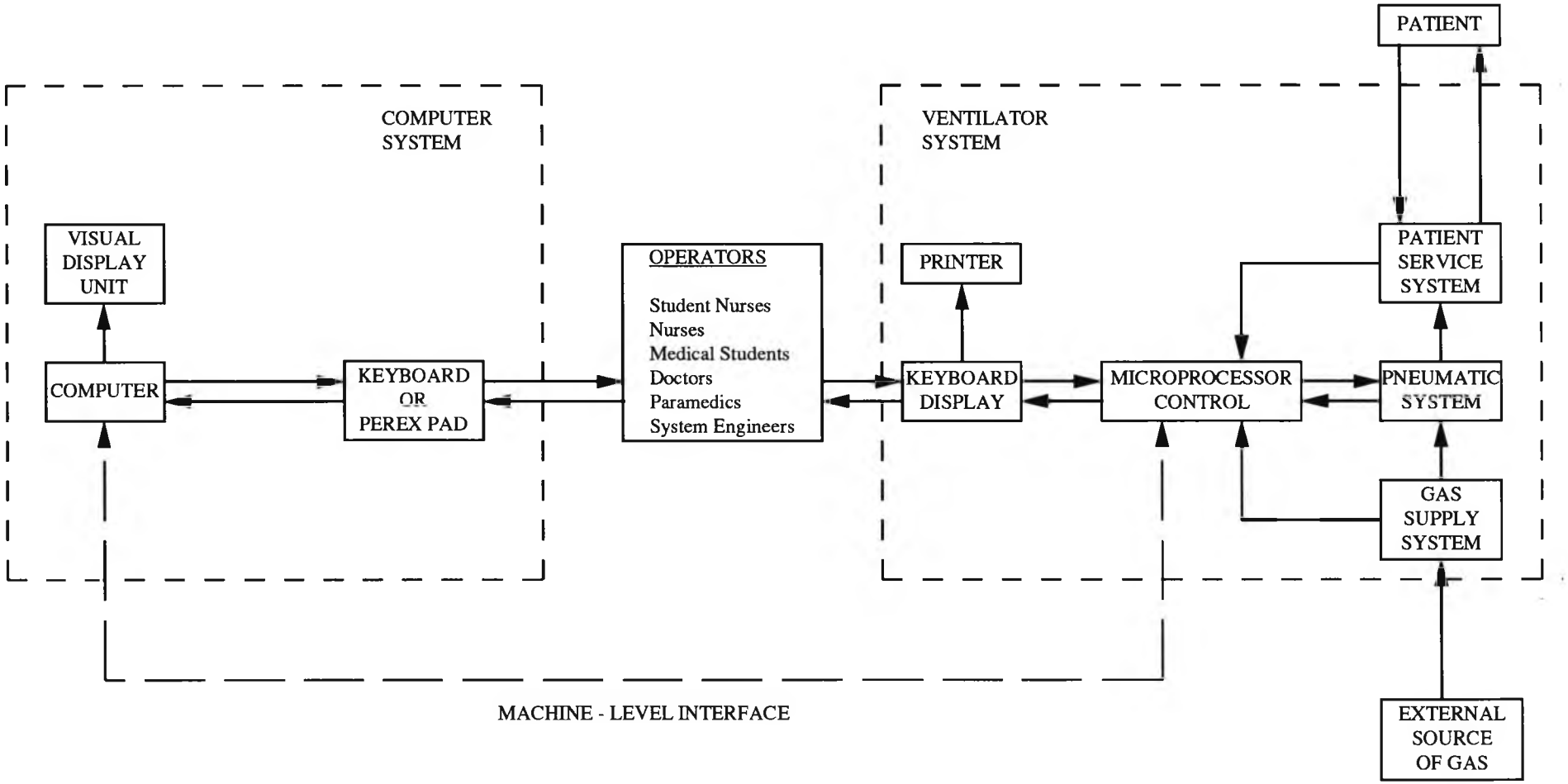


FIGURE 5.1 DIAGRAMMATIC OVERVIEW OF SYSTEM ARCHITECTURE

signals and to take appropriate action

* Paramedics

  + to enter data via the computer keyboard or alternative means of data entry

  + to be aware of the meaning of the audible and visual alarm signals and to take appropriate action

  + to have superficial knowledge about the operation of AIRS

  + to have superficial knowledge about the operation of AIRS

  + to liaise with the system engineer

* System Engineer

  + to have knowledge about AIRS at every level

  + to manufacture appropriate computer software

  + to obtain the primary data-set used for subsequent information processing.

  + to investigate alternative methods of data entry

  +to decide on computer output format (in conjunction with other users)

  +to decide on the format for printer output (in conjunction with other users)

  + to liaise with all other system users

* Medical Students and Student Nurses

  +AIRS should have an educational and training role

The **ventilator system** comprises six functional units: the microprocessor control unit, the gas supply system, the pneumatic system, the patient service system, the keyboard display, and the printer.

The microprocessor control unit is responsible for the control action of the ventilator. It receives inputs from the keyboard display unit and from pressure, flow and temperature sensors found in the gas supply, pneumatic, and patient service systems. A controlled output port leads to the pneumatic system which generates and controls gas flow. An output to the keyboard display unit indicates patient and ventilator performance. The microprocessor control unit has a fail-

safe mechanism which contains instructions for use in emergency situations, for instance, when another part of the ventilator is functioning incorrectly. A battery-powered back-up unit is on stand-by if the mains power supply is interrupted.

The gas supply system allows two inputs (air and oxygen) from the external source (the wall supply). Filters on the input side prevent particles larger than 0.3 microns and moisture from entering the ventilator. As the wall supply pressure is between 35 and 100 pounds per square inch - gas (psig), an internal pneumatic regulator is required to reduce the pressure to a nominal 10 psig for ventilator operation. If wall supply pressure falls below 35 psig, or if working pressure falls below 7.5 psig, an automatic switching circuit becomes active which transfers the ventilator into emergency mode. In these situations the ventilator settings are pre-set by the manufacturer, the values being stored in the microprocessor control unit.

The pneumatic system consists of two parallel circuits, one for air and the other for oxygen. A pair of solenoid valves which monitor gas flow are important elements in this system. The solenoids are adjusted continually which allow the desired volume and composition of gas to be delivered to the patient. For patient safety there is a valve on the ventilator output port which prevents the patient from receiving gas at either too high a pressure or flow rate. There are also pneumatic sub-systems for: temperature correction; providing gas flow to the patient via a nebuliser; providing pressure to a balloon valve, (this seals the exhalation port during inspiration); and uni-directional check valves that prevent gas backflow.

The patient service system describes the patient-specific elements of the ventilator system. It includes the pipe circuitry which delivers the gas to the patient and returns the exhaled gas to the ventilator. A humidifier and/or nebuliser can be placed in the patient input side of the circuit. A flow sensor is situated on the patient output side, when integrated this signal yields the volume of exhaled gas. The patient service system also contains bacterial filters on both input and output sides, which confines any bacterial contamination to within the humidifier or patient delivery system. Moisture from exhaled air is prevented from entering the ventilator by a water trap. Unidirectional valves in the patient mouthpiece prevent any gas backflow.

The keyboard display system can be divided into three functional units. Patient data can be viewed and stored; patient and ventilator status can be inferred from the alarm unit; and ventilator setting adjustments can be made.

The printer is capable of making a hard-copy of four types of report; a patient data log, a chart summary, ventilator status, and ventilator self-test.

The **computer system** comprises the system unit, a keyboard and/or alternative methods of data entry, and a visual display unit to observe the output of the system. An example of an alternative device for data entry is a pressure sensitive graphics pad where the function and size of the input area can be pre-specified. Up to 256 A4-size paper overlays can be designed, allowing capture of a wide spectrum of data. Advantages of using this type of pad for data entry include: personnel do not have to learn typing skills (however rudimentary they may be); it can be quicker than keyboard entry; and non-numeric or qualitative data can be contemplated.

5.5 Needs Evaluation

In current practice much of the patient-related information flow is the concern of the nurse, so much so that a clerk-type role is assumed, (taking care of the data rather than taking care of the patient). This is unsatisfactory for both the nurse and the patient. Introducing an automatic means of data capture offers the potential for a reduction of this clerk-type nurse workload. Nurses often mistrust computer-based (automatic) monitoring systems, possibly due to the 'black box' approach commonly adopted. To combat this fear of new technology an effort was made to make AIRS as transparent as possible to the end-users.

For AIRS to be used on a regular basis it must fulfil various requirement criteria, the most important of which is that it should be better than the existing manual system. In this sense "better" means more cost-effective, measured in terms of amount of time and resources saved. The system must also lead to more enhanced patient care.

To enhance the care of the patient is the driving force for this study. In a similar study in the United States, an attempt was made to determine if correct ventilator adjustments were made in instances where patient PaO_2 fell below 50 mmHg (Sittig, 1988). Current expert opinion suggests that in such cases either F_IO_2 and/or PEEP be increased as soon as possible, or that mode of ventilation be

changed to SIMV or CMV. The results of this study showed that a correct action was taken in 85 % of the cases within 26.5 minutes (S.D. = 19.4, n = 41). However in 12.5 % of the cases (n = 6) a confirmatory blood-gas analysis was obtained, a procedure which took on average 46 minutes (S.D. = 24.1). This delay before any corrective action was taken constitutes a life-threatening event. It is suggested that such a delay would be less likely if a computer-based decision support system were *in situ*.

Resource constraints imposed on the development of AIRS meant that the ventilators used for the development of the system were in almost constant use within the High Dependency Unit. Therefore the system was developed off-line, where ironically most implementation time was devoted to a manual means of entering patient data. Nevertheless a means of converting the data output of the ventilator to a computer readable form was successfully identified.

5.6 Summary

The first two elements of a methodological framework imposed to introduce an intelligent component into an existing data-rich environment have been used to define the specification for its implementation. This constitutes the design phase of the methodology introduced in Section 4.3. An evaluation of the needs for a system such as AIRS reveals both user requirements in terms of replacing the clerk-type activity of the nurse and system requirements in terms of dealing with the complexity of incoming data and its transformation into a more meaningful form. Patient care is enhanced as a result of this change in technology.

The background information to achieve the implementation has been addressed, and includes details of the programming environment (PROLOG) together with a basic introduction to respiratory physiology. In the next chapter these strands are drawn together for the implementation of the system.

6 : IMPLEMENTATION AND EVALUATION

6.1 Introduction

This chapter draws upon both the methodology for intelligent system design, implementation and evaluation described in Chapter 4 and the set of design features generated in the last chapter.

A PROLOG programming environment is used for the implementation of AIRS. There are several reasons why this is an attractive alternative to the use of commercial expert system development packages: the style of programming mimics the 'test-and-hypothesise' cycle used by clinicians; modular programming is supported which allows ease of maintenance of the program; software routines and program features can be written for any type of application (for example, management and/or diagnosis); and the program source code is more readable than conventional programming languages, as discussed in the last chapter. There are two drawbacks in using PROLOG for the implementation of the system. First, the memory allocated to 'workspace' within the environment is not large enough to accommodate the processing required in complicated cases. Second, the time taken to execute some clauses within the program can be extremely slow, causing problems with user acceptance of the human-computer interface. However, as AIRS is a prototype system, the constraints of workspace size and speed of execution can be relaxed. It can be argued that the purchase of an (expensive) software development environment would have decreased the time taken to complete the implementation. This can be offset by the number of features that can be included in a customised implementation taken together with the number of redundant features in a commercial development package.

In Section 6.2 the Early Prototype System is described. This illustrates the user interface, denuded of a knowledge base and other features of a fully integrated system. The purpose of this phase of the development cycle is to show the Human-Computer Interface to a putative set of end-users. Sections 6.3 and 6.4 describe the Knowledge Engineering Design Cycle and Interface Design Cycle respectively. These processes illustrate the way in which intelligence is incorporated into a computer system. Both processes are highly iterative in nature. Within the Knowledge Engineering Design Cycle the sub-processes of knowledge acquisition and knowledge representation are documented. It is these phases of the implementation where the

information processing functionality is included in AIRS. This capability elevates AIRS above the data processing functionality of most existing instrumentation systems for patient care in this area. Although the Interface Design Cycle describes the off-line version of AIRS, the modification necessary for on-line operation is illustrated. When the output of both the Knowledge Engineering and Interface Design phases are brought together a 'Late Prototype System' is formed. This is described in Section 6.5. This version of the system can be fine-tuned until the clinical users are confident that the resulting knowledge-based system can be improved no further. This milestone in system development is formalised in Section 6.6. The output of the methodology, an intelligent knowledge-based system, is detailed in Section 6.7. The ability to build knowledge-based systems has been evident for the past decade, yet there are only a tiny proportion of such systems in routine operation. Perhaps one reason for this is that an appropriate evaluation strategy has yet to be formulated. The evaluation of AIRS follows the development procedure from the requirements phase. A formative evaluation (Section 6.8) occurs at every intermediate phase of development; a summative evaluation is performed when the system has reached fruition (Section 6.9); and an overarching meta-evaluation (Section 6.10) ensures that the evaluation procedure has proceeded in an orderly manner.

Where appropriate, some features of the implementation are presented. Accompanying annotated software code listing can be found in Appendix II.

6.2 Early Prototype System

The early prototype system provides an opportunity to demonstrate several types of screen interface to the clinical and nursing staff who form the end-users of AIRS. From informal but close questioning about the use of computers in clinical medicine, it was recognised at an early stage that a Windows-Icons-Menus-Pulldowns (WIMP)-type interface would be potentially the most acceptable format for screen display and control. This choice was influenced by users comparing the facilities offered by a DOS-based IBM compatible microcomputer with the WIMP interface of an Apple Macintosh microcomputer. Using PROLOG it is possible to implement a WIMP-type interface (the 'W', 'M' and 'P' components provide no problem, but the 'I' is more difficult to implement and is excluded in this prototypic version). Menus are used to control the logical flow of operation in

AIRS, the outcome being a nested hierarchy of screens. In all menu-formats an option is given to return to the preceding menu or to go to the 'top' (i.e. first) menu encountered in the system.

The clinicians and nurses advised on the screen display format and use of peripheral devices, thereby refining the early prototype system. For example, using a computer keyboard to operate the screen interface was too slow. To counter this problem a 'mouse' was included as a peripheral device to speed up screen management and data entry, although some users had difficulty initially in using the device. The order of the menu options was also chosen at this stage, though some changed subsequently as development of the system progressed. A more fundamental change in menu options involved the use of the explanation facility. Instead of having just one menu entry to the explanation facility at the top-menu level, an explanation option was included in each of the relevant sub-menus. The result of this change allowed the explanation screen to be viewed with far fewer key-clicks, also saving time in the interaction between the user and the system. This had the effect of increasing the acceptance of the system.

Figure 6.1 shows the screen output of the top-menu screen display, this complements the program code excerpt given in Appendix II (AII.1 and AII.2) which describes the implementation of the WMP-system interface.

6.3 Knowledge Engineering Design Cycle

To the user, the levels of intelligence embedded within AIRS are exhibited in the **analysis** option of the topmost menu of the system. In fact three levels of system intelligence can be defined which match the three phases of ventilation. Within the AIRS **start-up** phase context-sensitive information is portrayed. This yields advice and explanation about the initial machine settings as the patient is connected to the ventilator. The **maintain** phase uses a data-driven algorithm to define the respiratory state of the patient in terms of alarm status (normal, high, low). A higher level of intelligence is employed in this phase than the previous one as each item of data is checked using a pattern-matching algorithm and classified accordingly. The highest level of intelligence is exhibited in **weaning** the patient from the ventilator. In this phase knowledge-based technology is used to represent the weaning process.

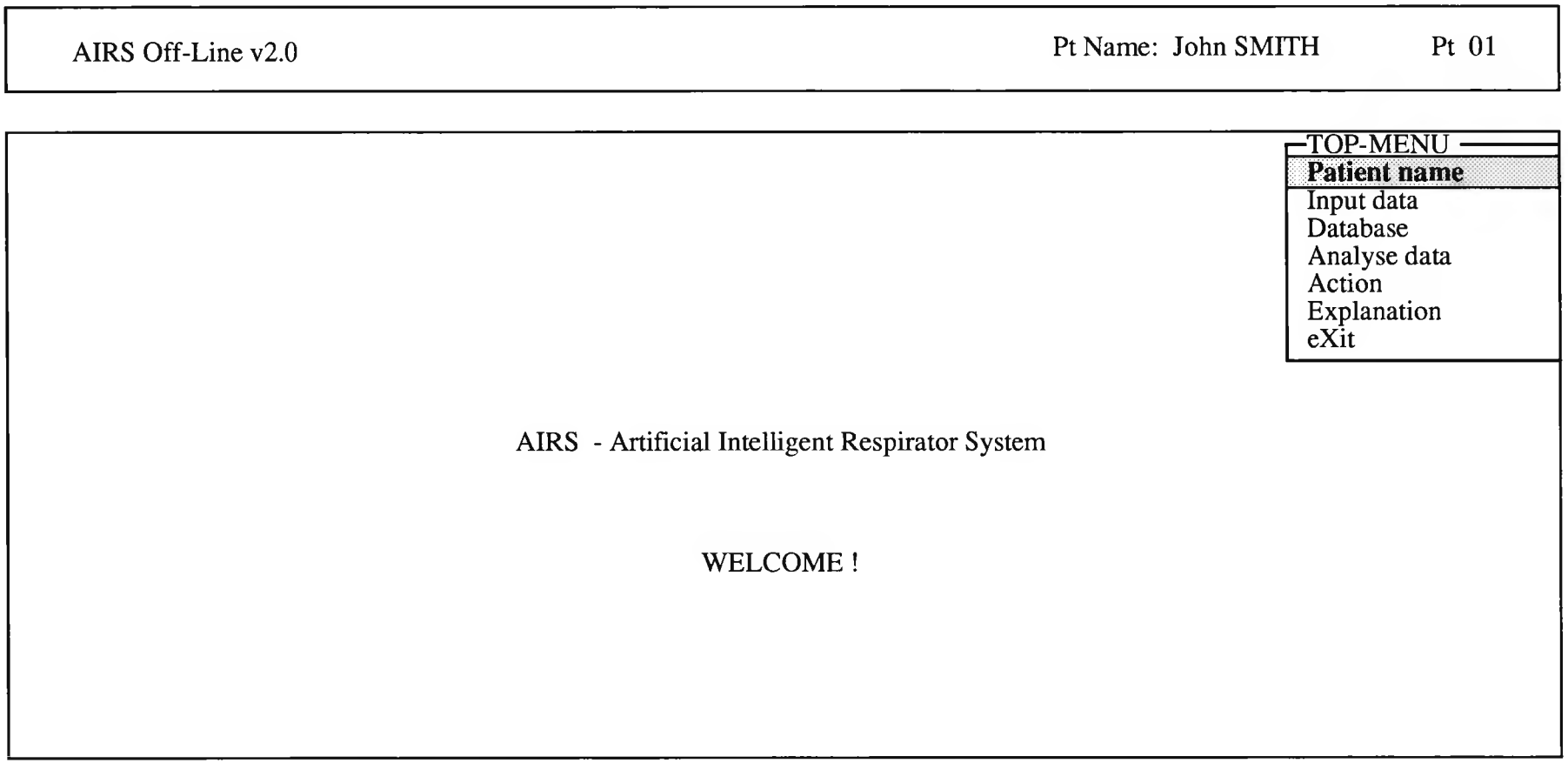


FIGURE 6.1 THE BANNER SCREEN - AIRS

6.3.1 First level of intelligence: AIRS start-up

To initialise the patient on the ventilator the working diagnosis is required so that patient state can be established. Patient state can be described in various ways, some of which are variations on a single theme. For instance, it seems fashionable to try and assign a quantitative value to a number of diagnostic findings, which in turn contribute to a final patient 'score'. It is the deviation of this score from the norm which determines patient state. Studies by research groups in the United States seem particularly to favour this type of technique (Siegal, 1981; Shoemaker et al.,1982). In AIRS the diagnostic category in which the patient resides is sufficient evidence to establish patient state.

A retrospective analysis of the Intensive Therapy Unit log book at the Royal Free Hospital, London, between May 1986 and December 1988 revealed that the 630 patients who spent more than 24 hours in the Unit could be classified into 14 independent diagnostic states. Only 5 patients (0.008%) in this database were unclassifiable, (see Table 6.1). It follows that to match a patient to the correct start up settings, 14 separate options are required, one for each of the dignostic states. To illustrate the data processing involved, software source code for the ventilator start up settings for cardiac patients is shown in Appendix II-3. Whereas the **initialisation** screen shows the various ventilator settings, two further options are also possible: the **action** screen and the **explanation** screen. The **action** screen takes into account the weight of the patient, so gives patient-specific ventilator settings. The **explanation** screen gives a context-sensitive description of why a particular ventilation strategy is favoured. Figures 6.2, 6.3 and 6.4 show respectively the **initialisation**, **action** and **explanation** screens for a particular cardiac patient.

6.3.2 Second level of intelligence: AIRS maintain

At the centre of the programming module which describes the second level of intelligence is a value-matching algorithm. The information which forms the basis of this algorithm was elicited from clinical personnel. Clinical indicators which describe some aspects of patient respiratory state were prioritised, the resulting seven indicants being partial pressures of oxygen and carbon dioxide in arterial blood, pH, tidal and minute volume, respiratory rate and peak inspiratory flow rate. Figure 6.5 shows the form in which information for one indicator, tidal volume, was gained.

CLASSIFIER	No. Patients
HEALTHY LUNG	180
POST-OPERATIVE	229
FLAIL CHEST	5
LEUKAEMICS	33
PNEUMONIAS	55
SEPSIS	31
ARDS	11
CARDIAC	54
PULMONARY EMBOLUS	5
ASTHMATICS	8
COAD	5
MULTIPLE ORGAN FAILURE	29
RENAL	28
EPIGLOTTITIS	6
UNCLASSIFIED	5

NB: 54 patients are in two groups

TABLE 6.1 DIAGNOSTIC STATES

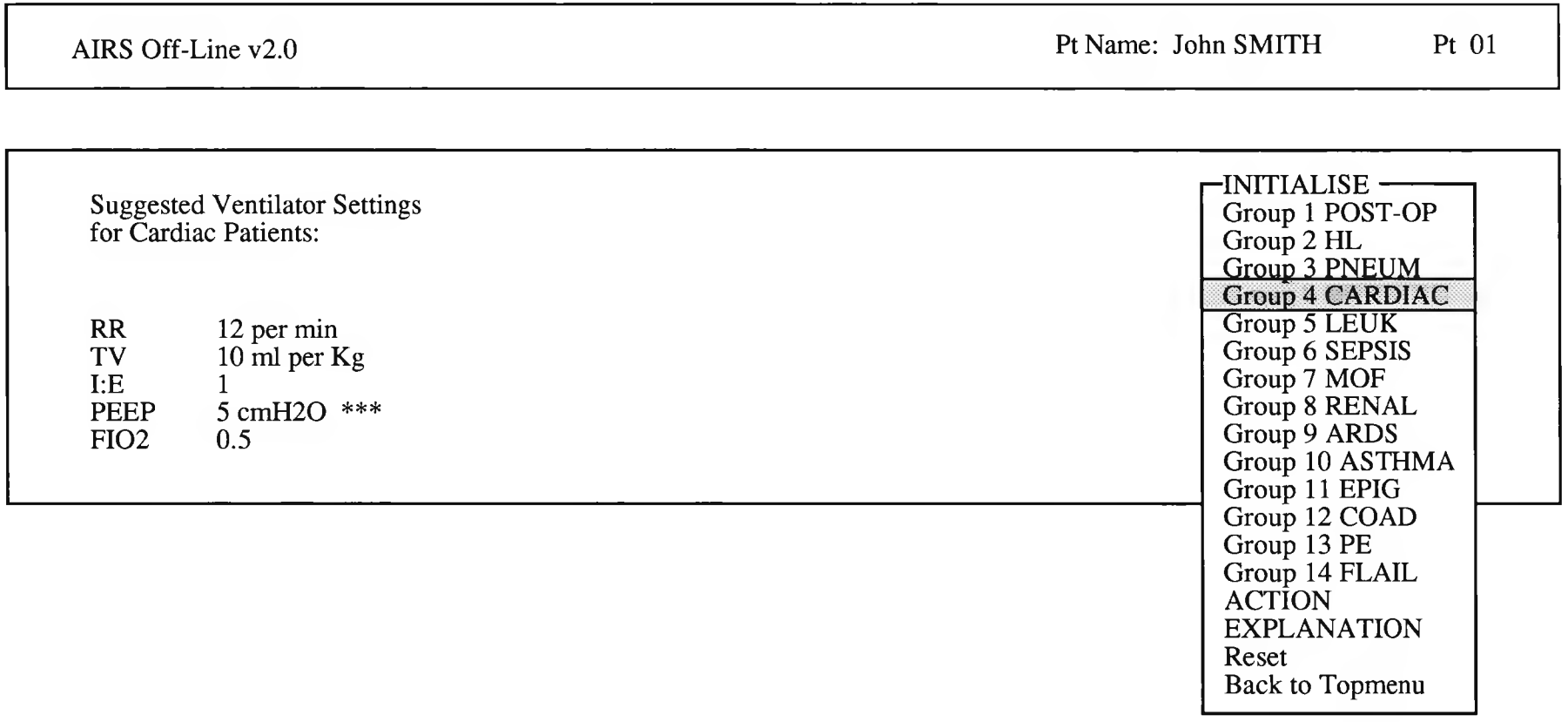


FIGURE 6.2 INITIALISATION SCREEN FOR A CARDIAC PATIENT

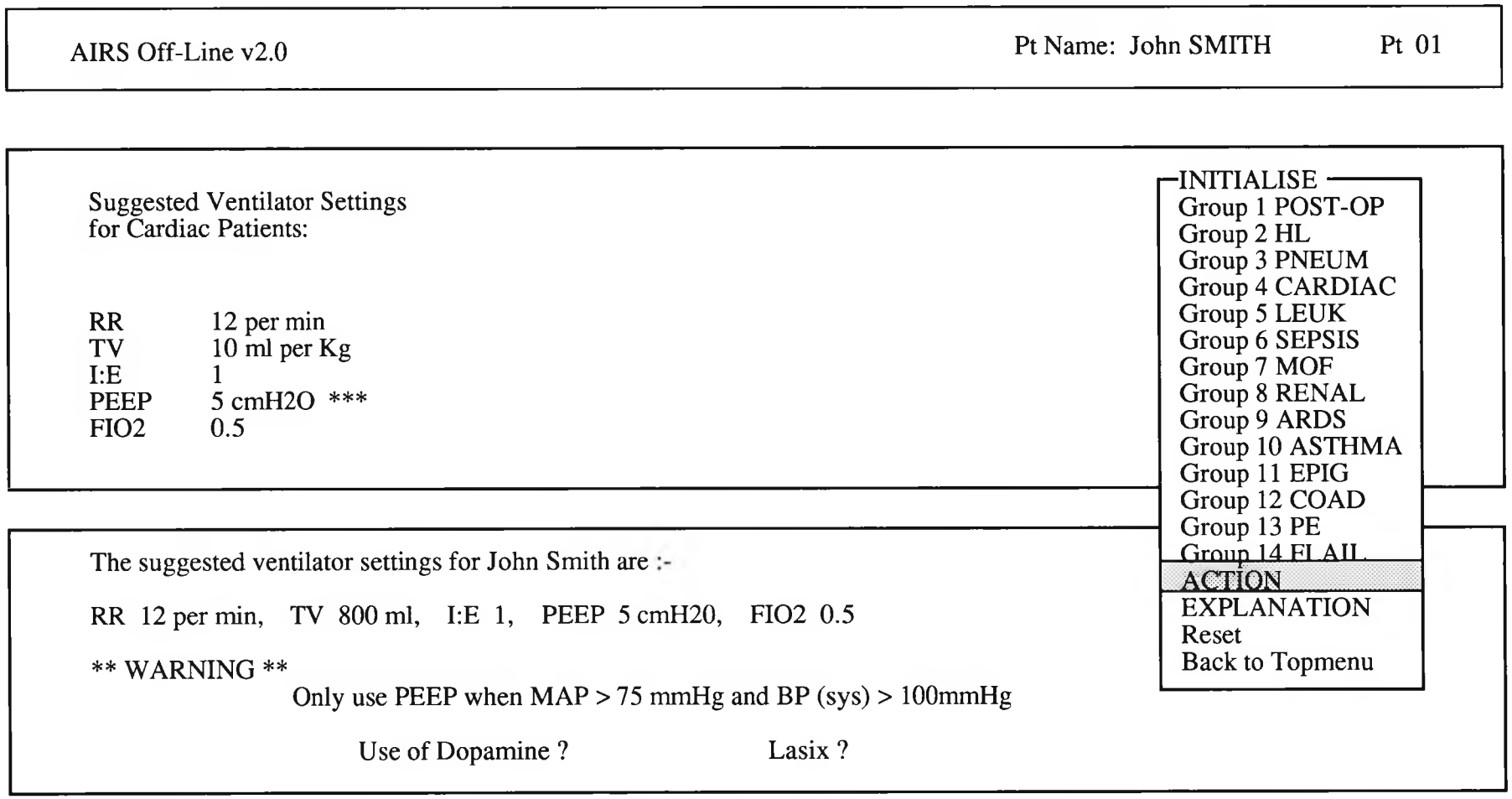


FIGURE 6.3 ACTION SCREEN FOR A CARDIAC PATIENT

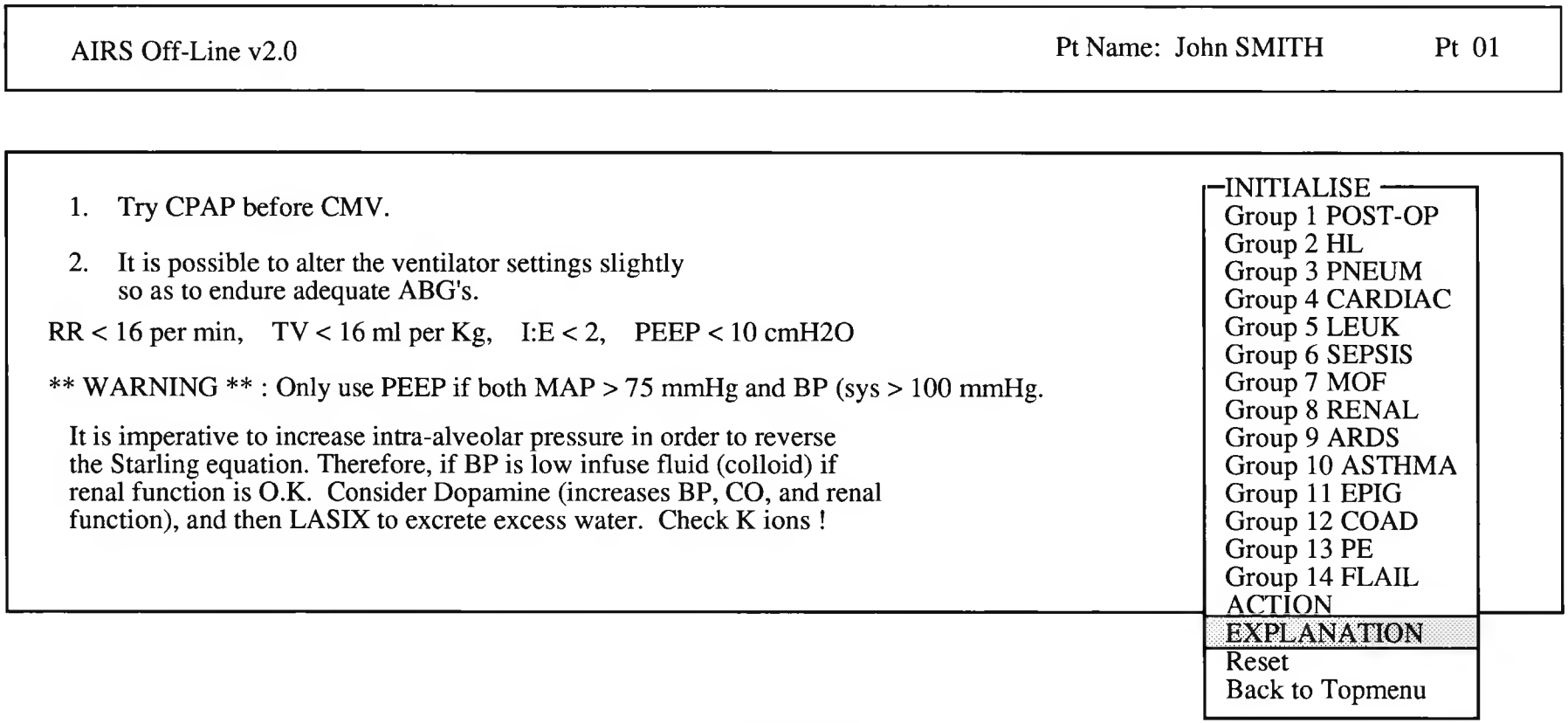


FIGURE 6.4 EXPLANATION SCREEN FOR A CARDIAC PATIENT

VARIABLE	ABBREV.	UNITS OF M'MENT	FREQ OF M'MENT	SET-POINT ALARM HIGH LOW	TREND ALARM	ACTION SET-POINT	TREND
						CMV	
TIDAL VOLUME	V_T	ml	h**	$30mlKg^{-1}$ $1mlKg^{-1}$	if ≤ 50% of previous recorded value.	patient disconnected (leak) machine malfunction	check for leak
		$mlkg^{-1}$ (when calculated)	or as necessary				
	TV (PB7200* Manual)	l (PB7200* Manual)					
						SIMV	
						patient disconnected (leak) machine malfunction	↑ rate of machine breaths by two
						↑ rate of machine breaths patient malfunction	
						SPONT	
						re-initialise patient to ventilator are ventilator settings appropriate ?	has RR ↑ ? is patient tired ? is V_T low because of respiratory depressants ?
						FOR ALL ALARMS CHECK FOR INAPPROPRIATE VENTILATOR SETTINGS	

* PB7200: Puritan-Bennett 7200a ventilator
** Present time resolution of paper record system is hourly

FIGURE 6.5 MAINTAIN: TIDAL VOLUME

Great care was used in the design of the **maintain** screen, one of the more important design concepts was to obtain a display which appeared uncluttered yet still had sufficient information content to remain useful. Each window icon (box) associated with one of the indicators is split into two sections: the top-half displays the set-point alarm state, and the bottom-half displays the trend alarm state. By using this combination of set-point and trend alarms it is the aim of the program to keep patient-threatening events to an acceptable minimum.

Appendix II-4 illustrates the software code for PCO_2 when the ventilator is in 'Controlled Mandatory Ventilation' mode. Before this level is reached, various pre-processing modules have checked the data for completeness and have set up the window presentation system which gives a qualitative interpretation of the chosen measurements. In the first instance, this design feature has been achieved by using the colour green for a value in the normal range, blue if the value is low and red if the value is high. At an earlier prototyping stage, a flashing blue window signified a value in the 'very low' range and a flashing red window corresponded to a value in the 'very high' range. If requested by the users, this facility can be re-installed.

6.3.3 Third level of intelligence: AIRS wean

The weaning phase differs significantly from AIRS: initialisation and AIRS: maintain, as knowledge is processed rather than data or information. This requires the use of knowledge engineering techniques to transfer knowledge from a human expert to an internal computer representation. A diagrammatic representation of the knowledge-based system design cycle is shown in Figure 6.6. In the development of **AIRS: wean** various 'experts' were used, covering technical as well as clinical domains. This allowed the capture of knowledge about machine malfunction together with patient dysfunction. Knowledge elicitation, the process by which a 'paper' system of expertise can be constructed, was performed by a series of structured interviews (see Chapter 7 for further discussion). This process was exacerbated by rotation of clinical staff. Paradoxically, the resultant knowledge base may be more widely acceptable because of the increased scope afforded by the different perspectives from the many staff involved in knowledge elicitation.

Knowledge is represented as premise-action PROLOG clause pairs. The control and reasoning strategy is entirely data driven.

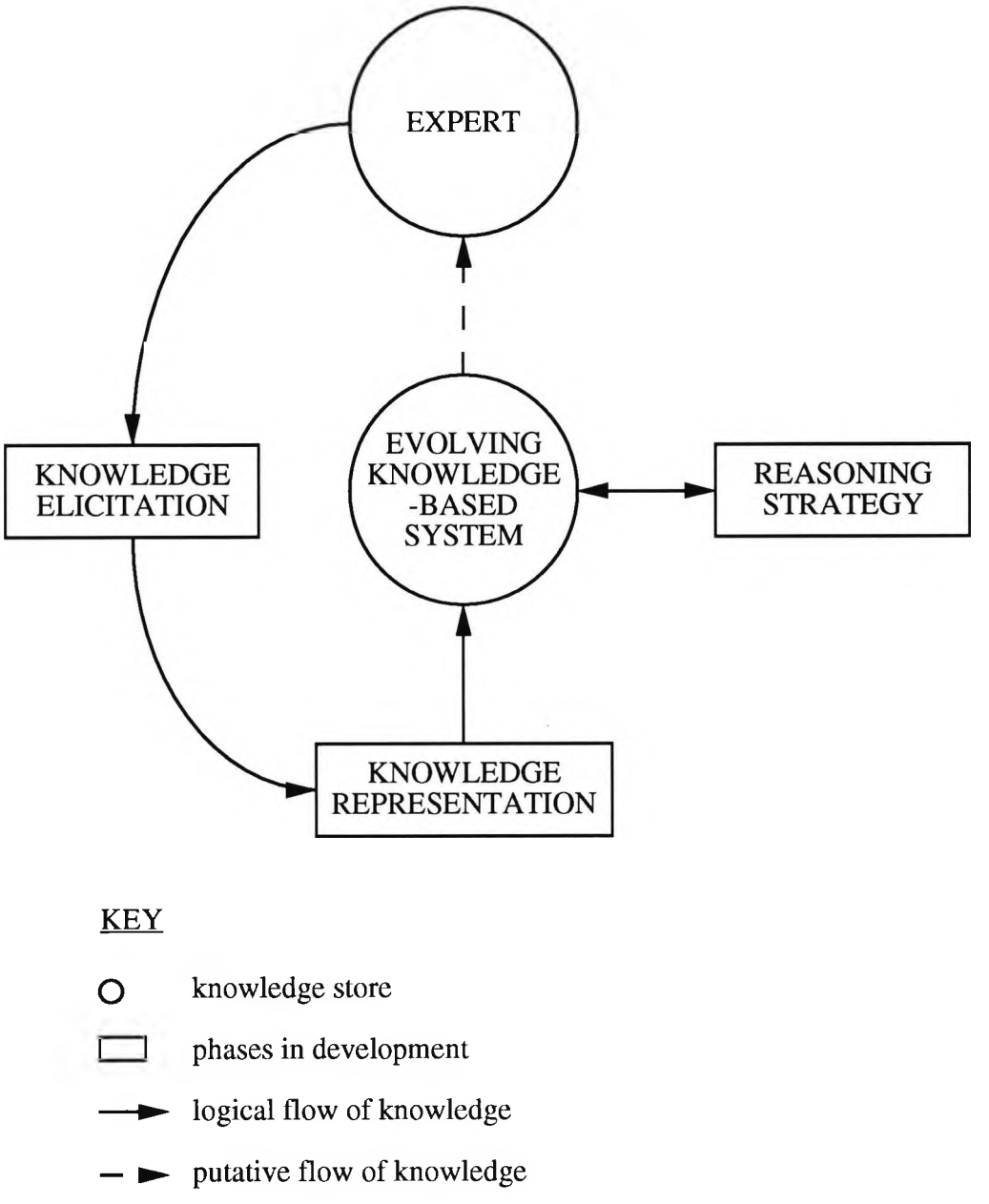


FIGURE 6.6 KNOWLEDGE-BASED SYSTEM DESIGN CYCLE

Associated with each premise of the clause pairs is a list of data findings which must be known (instantiated) before the premise can be evaluated, (see Appendix AII.5 for more details).

A conceptual model of the weaning process is shown in Figure 6.7. The start state is depicted as the patient on the ventilator in controlled mandatory ventilation (CMV) mode, that is, the ventilator is in complete control of patient ventilation. It is possible, and is common in current practice, for the patient to be connected to the ventilator in synchronised intermittent mandatory ventilation (SIMV) mode, that is, the ventilator initiates inspiration when triggered by patient inspiratory effort with a fail-safe mechanism of delivering a set number of breaths per minute. In the model the patient progresses form the start state to the goal state of spontaneous ventilation via a number of intermediate states.

Five types of rule become apparent to implement this conceptual model, as shown in Figure 6.8: weaning rule, progression rule; regression rule; termination rule; and a fail-safe meta-rule. If CMV mode is the start state, then the 'fit to wean?' criteria of the weaning rule must be met before progress to the intermediate states can commence. These criteria are quite extensive and cover causal mechanisms for respiratory muscle fatigue, fluid-electrolyte imbalance, physiological system failure, and patient anxiety. Figure 6.9(a-e) illustrates the causal mechanisms and relationships in a conceptual diagram. Similar conceptual diagrams are used to show the progression and regression rules (Figures 6.10 and 6.11 respectively).

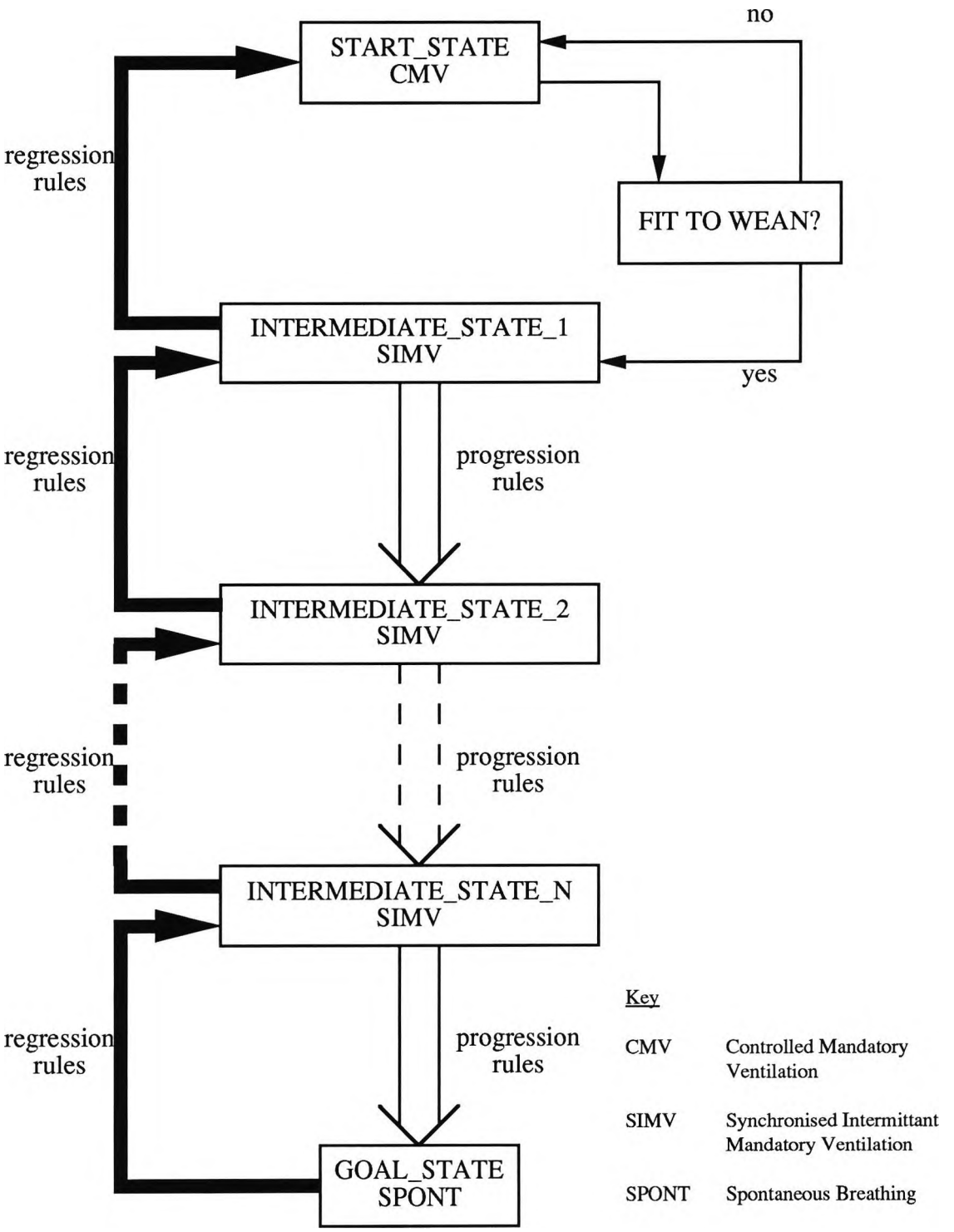


FIGURE 6.7 CONCEPTUAL MODEL OF THE WEANING PROCESS

WEANING_RULE_1 :	IF START_STATE and fit_to_wean THEN go to INTERMEDIATE_STATE_1.
PROGRESSION_RULE_2 :	IF INTERMEDIATE_STATE_N and progression_rules_succeed THEN go to INTERMEDIATE_STATE_N+1.
REGRESSION_RULE_3 :	IF INTERMEDIATE_STATE_N and regression_rules_succeed THEN go to INTERMEDIATE_STATE_N-1.
TERMINATION_RULE_4 :	IF GOAL_STATE THEN STOP.
META_RULE_1 :	If regression conditions succeed and progression conditions succeed THEN do REGRESSION_RULE_3

FIGURE 6.8 VERBAL DESCRIPTION OF CONCEPTUAL MODEL OF IMPLEMENTATION

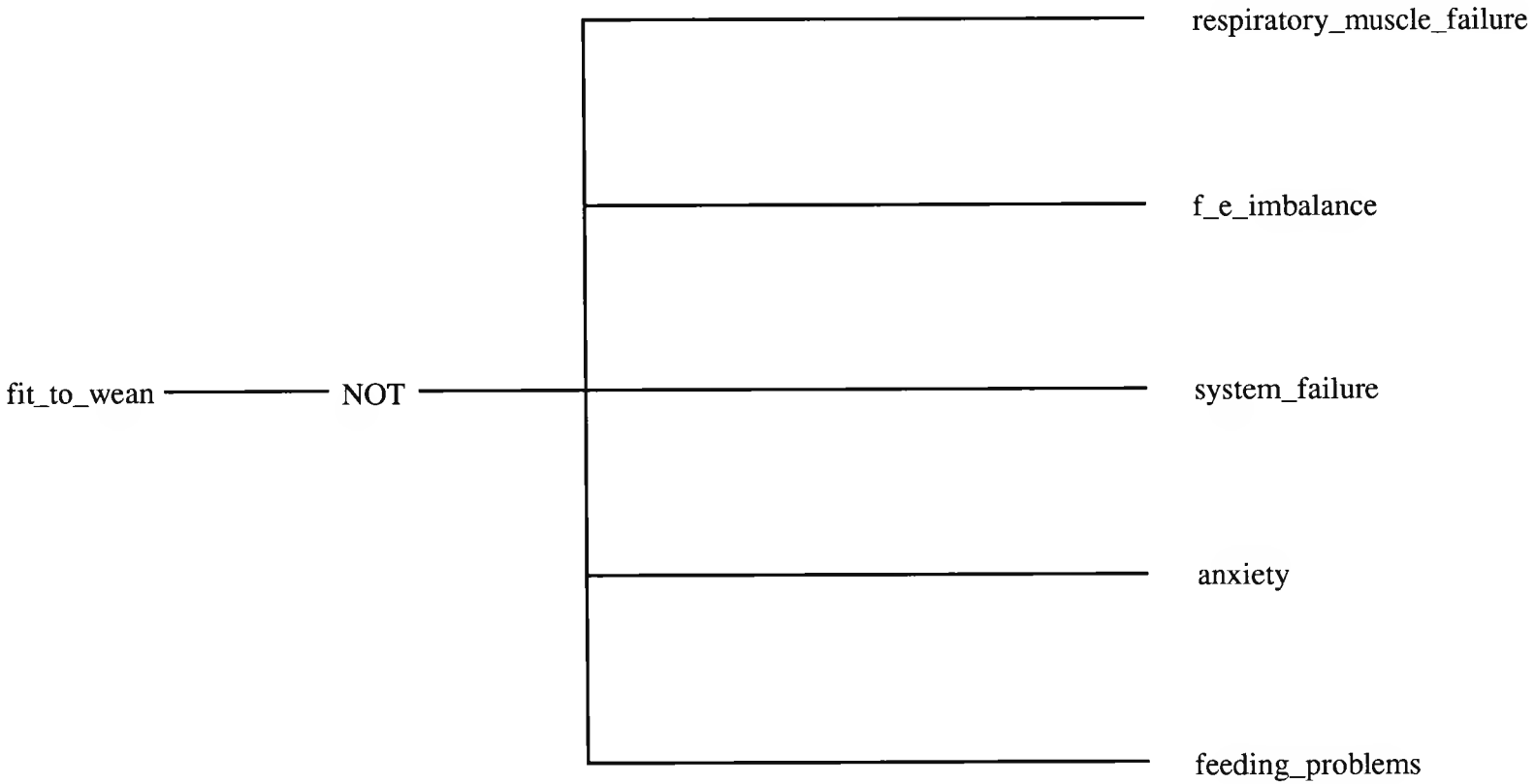


FIGURE 6.9a RULE ASSOCIATIONS FOR FIT_TO_WEAN

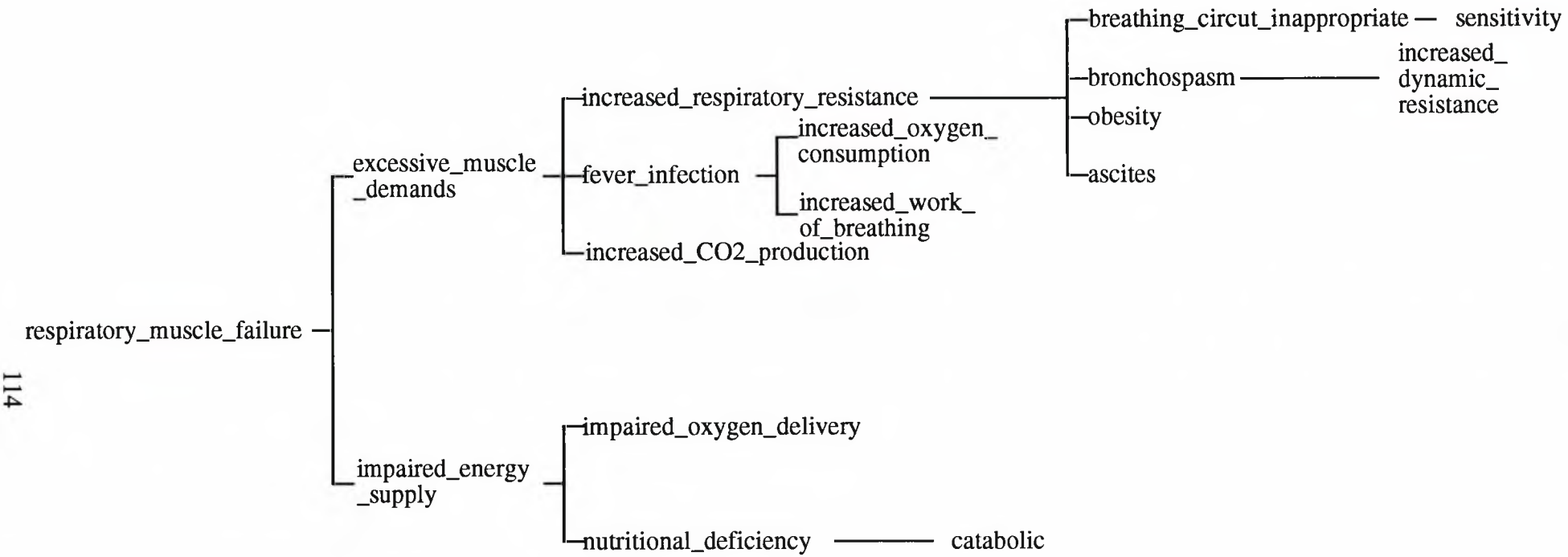


FIGURE 6.9b RULE ASSOCIATION FOR FIT_TO_WEAN (Contd.)

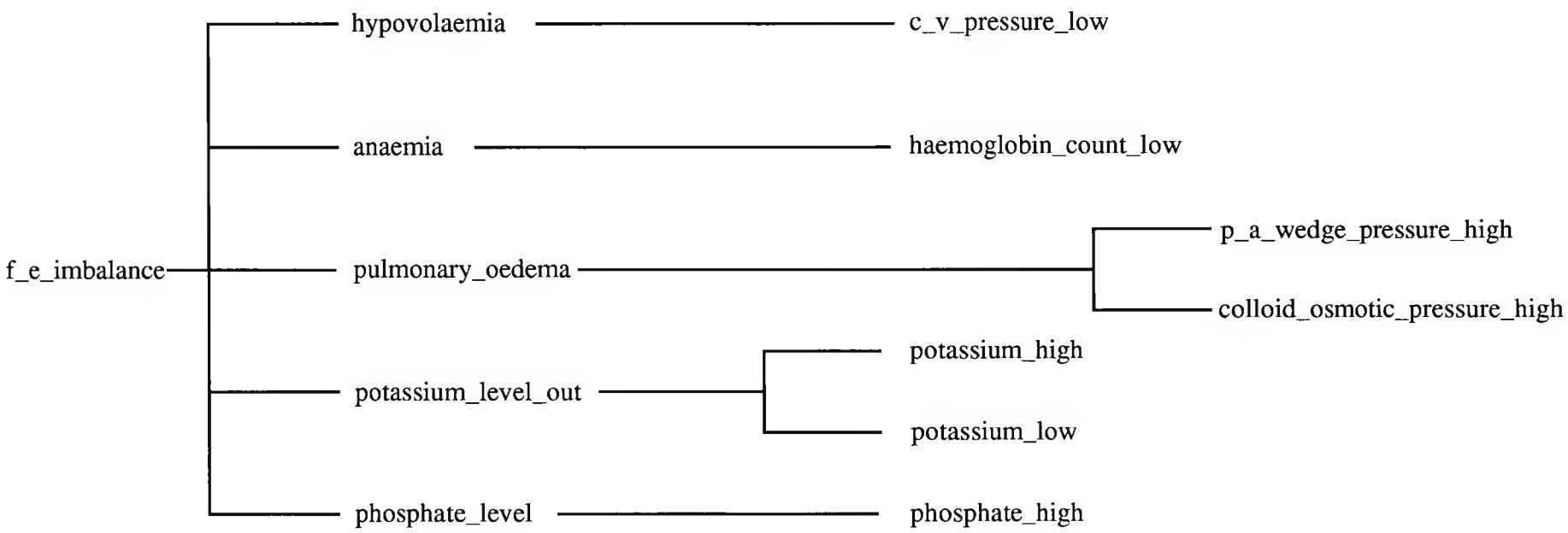


FIGURE 6.9c RULE ASSOCIATION FOR FIT_TO_WEAN (contd.)

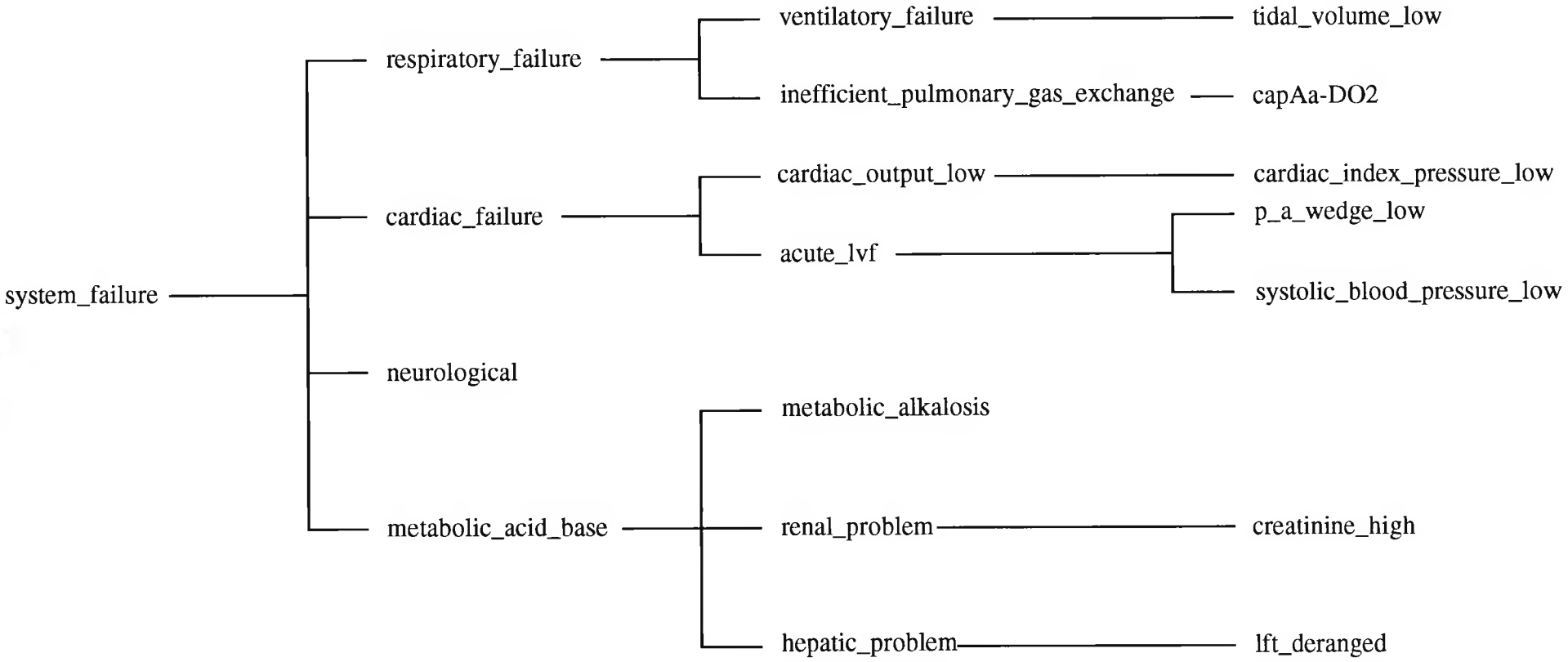


FIGURE 6.9d RULE ASSOCIATION FOR FIT_TO_WEAN (contd.)

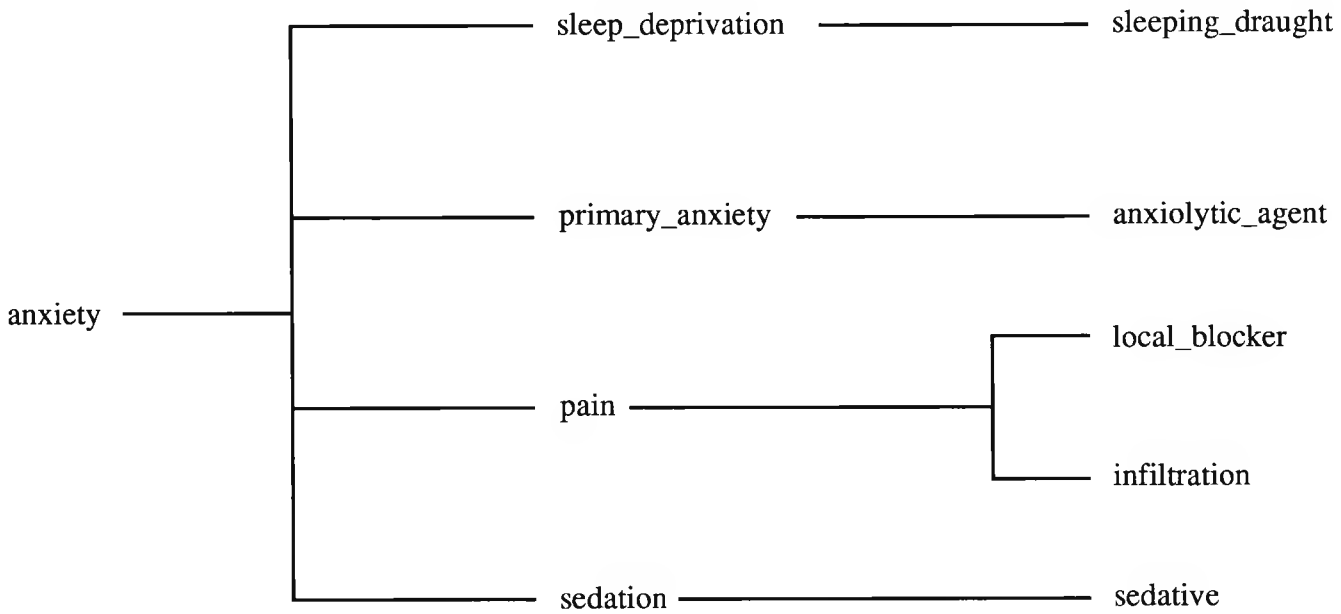


FIGURE 6.9e RULE ASSOCIATION FOR FIT_TO_WEAN (contd.)

THIS PAGE IS LEFT BLANK INTENTIONALLY

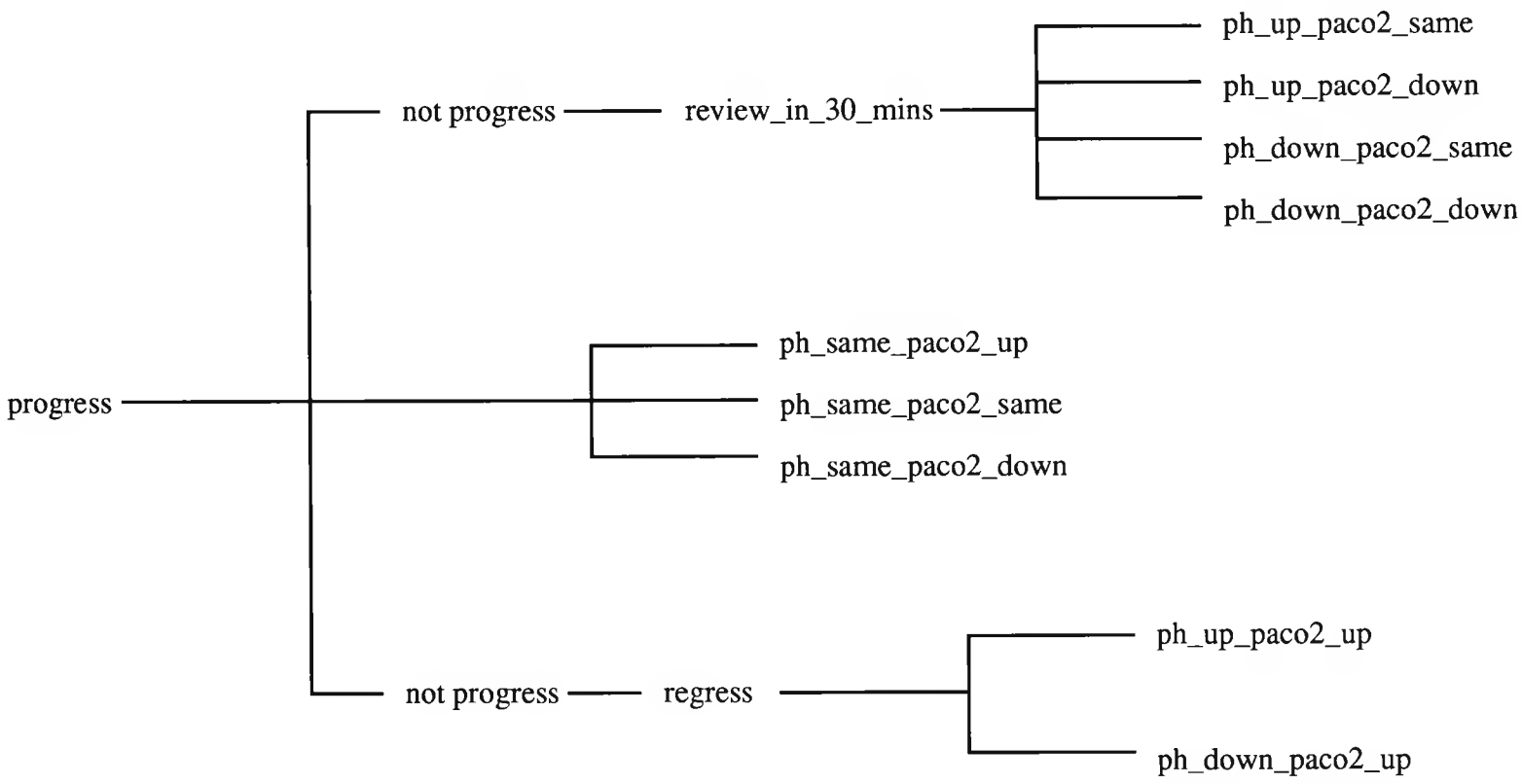


FIGURE 6.10 PROGRESSION RULES

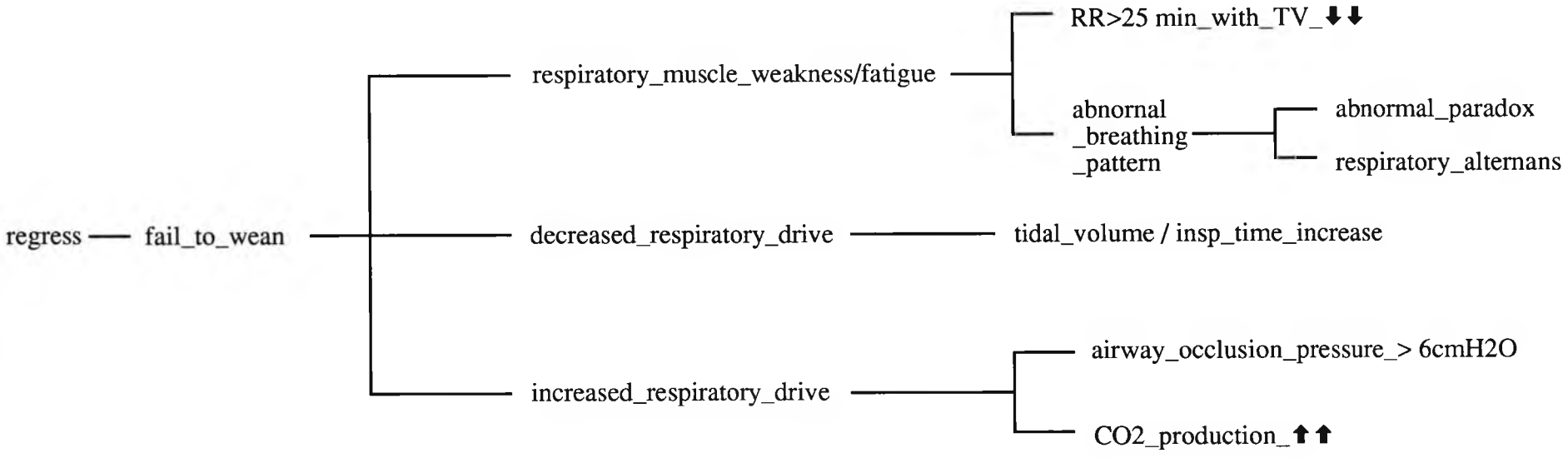


FIGURE 6.11 REGRESSION RULES

6.4 Interface Design

6.4.1 Data Transfer at the Machine Level Interface

At the machine-level software interface the PROLOG programming environment can deal directly with a list of ASCII characters that would form the output from the digital communications interface of the ventilator. Equally, characters which specify instrument commands can be converted into a list of ASCII characters, allowing commands from the computer to be interpreted by the ventilator. This two-way communication link is implemented by use of two built-in predicates which have the effect of converting a list of ASCII characters into the required atom notation; they are **list(List,String)** and **name(Atom,String)**. (**Note:** in some PROLOG implementations this conversion is made even more simple, as the **name** built-in predicate is defined as **name(Atom,List)**). In the program code illustrated below in Table 6.2 **pdtest/0** is the clause which defines communication from the ventilator to the computer, and describes a list of patient data. To decode what the data represent needs further information obtained from an ASCII look-up table (not shown).

```
pdtest:-
list([80,68,44,49,51,58,52,53,32,44,49,50,46,49,32,32,
      44,54,46,48,54,32,32,44,48,46,57,32,32,32,44,49,
      46,51,32,32,32,44,48,46,53,48,32,32,44,48,46,48,
      48,32,32,44,52,46,52,32,32,32,44,52,46,50,32,32,
      32,13],X),
name(Y,X),
write(Y).
```

TABLE 6.2 Machine Communication in PROLOG

The two arguments in the **list/2** predicate are a PROLOG list and an ASCII string. In the example above, **X** becomes instantiated to

"PD,13:45 ,12.1 ,6.06 ,0.9 ,1.3 ,0.50 ,0.00 ,4.4 ,4.2"

Apart from the string identifier **PD**, there are nine data fields each consisting of six characters. These data fields comprise the time of measurement and eight items of patient respiratory function. The last character in the list is ASCII(13), this is a non-printable character which represents "carriage return". Thus, when the **name/2** predicate is reached, whose arguments are an atom and a string containing the characters of the name of the atom, the string has become instantiated

and its atom equivalent is returned. This is then used by the **write/1** built-in predicate to send the patient data to an output device. At this point the data are in the following format :-

```
PD,13:45 ,12.1  ,6.06  ,0.9   ,1.3   ,0.50  ,0.00  ,4.4   ,4.2
```

To convert this output into a more acceptable form a data compression routine can be employed. This can be achieved in a number of standard ways. Illustrated below in Table 6.3 is a routine to eliminate each occurrence of a space character (ASCII equivalent is 32), adapted from Bratko (1986).

```
squeeze:-
get0(C),
put(C),
dorest(C).

dorest(13) :- !.          /* ASCII(13) = carriage return */

dorest(32) :- !,          /* ASCII(32) = space */
get(C),
put(C),
dorest(C).

dorest(Letter) :-
squeeze.
```

TABLE 6.3 Data Compression in PROLOG

The **get0/1**, **get/1** and **put** built-in predicates are used for character input and output. This routine will read the first character of any output atom, and then continues character processing in one of three ways. If a carriage return is met the routine terminates; if a space is met it is disregarded; if any other printable character is met it is returned to the output. Notice the powerful use of recursion in this procedure. If the patient data from the previous illustration is compressed in this way, the output will become :-

```
PD,13:45,12.1,6.06,0.9,1.3,0.50,0.00,4.4,4.2
```

Further list processing is required to obtain the standardised data format for PROLOG input to other modules, where each number must be separated by a comma (rather than a colon, as in the time data field).

This example shows that PROLOG can handle data conversion at the machine-level software interface. However, in the development of AIRS this approach was abandoned due to pragmatic constraints. Indeed

it is doubtful whether PROLOG processing is fast enough to cope with the required data-rate, and other faster commercial packages are available which provide the same functionality with a higher utility. Further investigation is required in this area before any firm conclusions can be reached.

6.4.2 Data presentation

Presentation of data occurs at two levels. The first uses a six-line 'data capture' window (**ventdb**) and the second uses the full screen to display values in the 'database' window (**ventdatdb**). The reason for this dual display is that users require some immediate visual feedback of the data being entered, hence the data capture window. For ease of use a similar menu to that used for data capture is employed to display data in the database window. Whilst in the database facility, it is possible to view the previous 22 values of any data item captured by the system.

6.5 Late Prototype System

This phase in the development of an intelligent system can be viewed as an important milestone, it brings together concurrent work in knowledge engineering and interface design. It allows the different levels of intelligence employed and the user interface to be examined by end-users. Whereas the early prototype system is denuded of knowledge, the late prototype system is a functioning system. To take advantage of bringing the users and full system together for the first time, the system engineer or developer may seek advice on how to improve the system and take note of the comments made by the full range of intended users. This milestone leads naturally to the next phase of development, which is to adapt the program for specific use.

6.6 Program Tuning

From the perspective of the system engineer, program tuning can be defined as the modification (and evaluation) of the software to improve performance so as to meet specific objectives. It is a two stage process: detection of problems in performance of the system (for example, response times); and modification of the system to correct the detected problems. Intuitively, this is an iterative process in which improvement of the system is heuristic. This indicates, perhaps, that a successful outcome is due to more artistic merit than scientific endeavour.

The techniques involved in program tuning include improving

the code and enhancing the file structure. In AIRS a significant improvement in response times was achieved by optimising the items contained in the PROLOG database. For example, the clauses used most often were placed at the top of the list in the database. The file structure was also changed so that a patient index number linked demographic data files to the time-stamped clinical data collected and archived by the system. This allowed the write-once only demographic data files to be accessed when necessary rather than take up valuable working memory.

Three 'laws' became evident when in this phase of development. A law of diminishing returns: it is estimated that 80% of results are obtained with 20% of effort, but it takes the remaining 80% of effort to achieve the remaining 20% of results (although rarely achieving the 'perfect' system). A law of pragmatism: there may be many interesting facets of development within a system, but if inappropriate behaviour is the outcome, effort has been wasted. Finally, Ockham's Razor: the simplest (or most obvious) way of doing something is often the way to proceed. All three laws were involved during the tuning of the AIRS software.

6.7 Intelligent Knowledge-based System

The development of the intelligent knowledge-based system into a final product is another important milestone. However development of the system does not stop once this phase has been reached. It is crucial to realise that AIRS will remain viable only as long as the knowledge it embodies remains current clinical practice. Knowledge in the domain of respiratory therapy is dynamic, new treatment regimes are being reported at regular and frequent intervals. The onus is on the clinicians and knowledge engineers alike to incorporate new knowledge into the system.

6.8 Formative Evaluation

The Formative Evaluation is an amalgam of evaluative queries active at different times in the development process of the system. This is reflected in the way that this evaluation process is presented below.

The components of the AIRS architecture comprise a ventilator system, a computer system, a range of possible end-users and the patient (the system of interest). Only functional decomposition of the computer system was allowed, the other systems

being standard. To enhance the acceptability of computer hardware a high resolution screen is used for data presentation, and an alternative means of manual data entry other than the QWERTY keyboard is being sought. The use of large, high resolution screens allows their content to be intelligible from a distance, freeing the nurse from the immediate environs of the patient yet still allowing a constant check on patient data.

The Early Prototype System shows the type of interface that is possible to the end-users. AIRS has a window-based menu system which is acceptable to all categories of end-user (student nurse to senior clinician). Speed of data entry was increased by using a mouse, although initial difficulties in handling the device meant that data entry was in fact slower before any benefit could be obtained. Menu-options were chosen at this stage, which were subsequently changed as development of the system progressed. A more fundamental change involved the use of the explanation facility: instead of having just one option at the main-menu level, an explanation option was included in each of the relevant sub-menus. Therefore far fewer key-clicks were necessary, thus saving time before an explanation was generated, which increased user acceptance of the system.

Within the off-line version of AIRS there are four types of interface: the hardware interface; the software interface; the human-computer interface; and the inter-personnel interface. In the on-line version of AIRS a physical connection must exist between the ventilator and the computer system so that data and instructions can be transferred between the two. It is forseen that this link will be a standard RS-232 or similar. The ventilator system issues data to its output port in terms of an ASCII equivalent to the character set. As shown in Section 6.4.1, the translation of ASCII characters back into alphanumeric format is a trivial problem using PROLOG. However in the off-line version of AIRS it was decided to omit this stage of data processing, as lists of numbers (ASCII code) do not enhance the transparency of the system. Instead much work was involved in the design of the human-computer interface, which uses a hierarchy of menus for manual data capture. It was important to make this method of data entry attractive, even though the majority of the interface would become redundant in the development of the on-line system. The evaluation of the inter-personnel "interface" can be viewed in terms of the educational needs of the persons involved. This functions at two levels: how to operate the system itself, and how to use the

information contained within the system. The first question should be addressed initially by the system engineer, and the second by senior nursing or clinical personnel. In AIRS the "interface" between clinical personnel and the patient remains an important aspect of the system. Evaluation of all these interfaces can be couched in terms of levels of communication between the sub-systems. If any one breaks down the integrity of the whole system is compromised.

The Formal Evaluation of Knowledge Engineering Design is linked in many ways to the formal evaluation of later phases of the methodology, and to the summative evaluation of the fully developed product. For example, the quality of the rule-set will not be known until various outcome measures are known. These include knowledge base "coverage" (that is, completeness of the knowledge base) and relevance of action suggested or explanation generated. This illustrates the iterative nature of the knowledge engineering design process.

The evaluation of the Late Prototype System is a first opportunity to query AIRS task knowledge. That is, does the system do what you want/expect it to do? This is the phase in the methodology when various teething problems can be identified and dealt with. Minor problems did occur at this stage in the development of AIRS, but these were all found to be faults with the program rather than errors in the process of operation.

Evaluation of the Program Tuning stage of AIRS is involved with observing the levels at which the set-point and trend alarms are set. It is also important to ensure that the correct action and appropriate explanation are generated for each alarm state. The system is able to handle situations where more than one alarm is active at any one time.

6.9 Summative Evaluation

The summative evaluation deals with that part of AIRS which is involved with data processing and interpretation. Therefore the analysis option of the main-menu is the focus of attention. For the purpose of this study the ventilatory process is deemed to consist of three separate functional stages: initialisation of the patient on the ventilator; maintaining the respiratory needs of the patient; and weaning the patient off the ventilator.

The initialisation stage involves knowledge of patient diagnosis. For each diagnostic state there are specific ventilator settings that should be used to initialise the patient. The maintain

option of the analysis sub-menu is concerned with keeping certain measured variables from the patient within desirable physiological limits. This is an alarm-driven technique which uses pattern cognition to elicit a visual alarm when the monitored data go outside the pre-set limits. Weaning a patient off the ventilator is the stage where expert system technology is used. The problem addressed is the determination of the point when the patient is fit enough to start the weaning process, and then move progressively through the SIMV mode of ventilation until spontaneous respiration is restored once again.

To provide a summative evaluation of AIRS a retrospective case study was undertaken. This utilises a computerised database which contains patients who have passed through the Intensive Therapy Unit, Royal Free Hospital, London, from May 1986 to December 1988. As it is routine practice to admit some post-surgical patients before they return to their surgical ward, all patients who entered the unit for less than twenty-four hours have been eliminated from the database. This leaves a total of 630 patients entered between May 1986 and December 1988; of these 427 (67 %) were ventilated at some time during their stay. These patients have been classified into fourteen diagnostic states (as previously seen in Table 6.1, p.103) which enables a grouped data analysis. Patients from each diagnostic group present their own challenge to the management of ventilatory performance. To meet this challenge AIRS has fifteen complementary management strategies. Preliminary results for three diagnostic groups which compare the initial ventilator settings as suggested by AIRS to those obtained from the retrospective study are shown in Table 6.4. This table shows the differences in respiratory rate (RR), tidal volume (TV), and the fraction of oxygen in the inspired air (F_IO_2). A more detailed statistical analysis awaits further data.

6.10 Meta-evaluation

From the evaluation of requirements, the need for an instrumentation system such as AIRS is strongly indicated. Progress in this field would be enhanced even if the system were used to identify possible problem areas for a subsequent generation of intelligent instrumentation. An attractive user-interface has been developed which has a fully integrated data presentation and storage/retreival system as a by-product of the off-line implementation.

	PNEUMONIAS			HEALTHY LUNG			CARDIAC		
	RR	TV	FIO2	RR	TV	FIO2	RR	TV	FIO2
RETROSPECTIVE STUDY	14.4 ± 3.1	0.74 ± 0.12	0.67 ± 0.18	12.7 ± 1.7	0.76 ± 0.11	0.43 ± 0.07	11.3 ± 0.1	0.76 ± 0.01	0.55 ± 0.20
AIRS	12	0.75	0.5	10	0.75	0.4	12	0.75	0.5

TABLE 6.4 COMPARISON OF INITIAL VENTILATOR SETTINGS IN THREE DIAGNOSTIC GROUPS

6.11 Summary

The methodology introduced in Section 4.3 and applied throughout Part II of this study covers total system development, incorporating software development and wider issues of technology transfer. As well as providing coverage for the entire development process, the application of a methodology provides a framework for support for system evolution, and is necessarily an open-ended process.

The methodology employed allows the development process to be partitioned into discrete phases, these are necessary to define milestones for evaluation purposes. This enabled the creation of decision points in the development of the system which facilitated the direction of effort.

Part III of this study considers the decision-support process from a wider perspective, although it builds on work presented in Parts I and II. It illustrates some of the processes undertaken to achieve the implementation of AIRS and gives information on **how** and **why** particular avenues of thought were pursued.

PART III

7 : DISCUSSION

7.1 Introduction

Part I of this thesis described the evolution of the components which comprise an intelligent instrumentation system to be applied to the domain of critical care medicine; Part II illustrated how a methodology for system design, implementation and evaluation could be used to generate a specific application; Part III now brings together these two strands and includes the wider issues on which the research is based.

For ease of discussion this chapter is divided into five sections which reflect the different methodological activities introduced in Part II. Section 7.2 discusses clinical information systems in terms of their underlying measurement and control functionality and indicates how intelligence can be embedded into such systems by using a knowledge-based formalism. This description illustrates the design concept of an intelligent measurement system. For its implementation, knowledge derived from human experts has to be transformed into a set of symbols which represent machine-readable software code. Two distinct phases can be identified within this transformation process: knowledge elicitation and knowledge representation. Discussion of each of these phases can be found in Sections 7.3 and 7.4 respectively. No matter how well knowledge is acquired and represented the intelligent system will not be used unless the interface to the user is attractive and meets other defined user requirements. These matters are discussed in Section 7.5, the Human-Computer Interface. A measure of the worth of the intelligent system can be determined at various points during its evolution. This, and a more general discussion of the evaluation of intelligent systems can be found in Section 7.6. Both specific and general conclusions of this study follow in Chapter 8.

7.2 Clinical Information Systems

Clinical information systems are underpinned by clinical measurement, which itself can be defined as those data collected in context of providing information about the physiological well-being, or state, of a patient. If patient state is regarded as a dynamic entity, then clinical measurement provides the basis from which patient state trajectory can be determined. Although this is a description which many clinicians would fear, the concept of patients

residing in a 'state space' has been used with some success by several clinical groups (Siegal, 1981; Shoemaker et al., 1982). Such an approach is favoured because it is more rigorous than the more traditional diagnostic-based approach to patient management issues.

Measurements, data, information and knowledge are related to each other by a series of transformations (see Figure 7.1). When recorded measurements of a particular variable are ordered, for example as a time series, they can be described as data. To transform these data into information requires classification of the data elements. Finally, knowledge can be obtained by interpretation of information together with appropriate explanation and/or justification.

For a complete understanding of clinical information systems employed in a critical care unit it is crucial to comprehend the general principles of measurement which form their base. A definition of measurement can be given as:-

> "Measurement is the process of empirical objective assignment of numbers to the properties of objects and events of the real world in such a way as to describe them."
>
> (Finkelstein, 1982)

If this definition is accepted then it is imperative to realise that measurement is concerned with two complementary systems: the natural system of the 'real' world and the formal system of the 'model' world.

Natural systems comprise some perceived aspect of the real world which we wish to study. There is a set of qualities in which exist definite relations. A perceived quality can be termed an 'observable', which is the most fundamental unit of the natural system. Relations between two or more observables can be termed a 'linkage'. Thus, the study of natural systems can be described in terms of system observables and linkages. For example:-

> "If an interaction between any two natural systems s_1, s_2, causes some change in s_2, say, then the vehicle responsible for this change is an observable of s_1, and conversely."
>
> (Rosen,1985)

Quantitation can be introduced into the analysis of natural systems by noting that:-

> "...if a quality of such a system corresponds to an observable, a quantity corresponds to a specific value of an observable."
>
> (Rosen,1985)

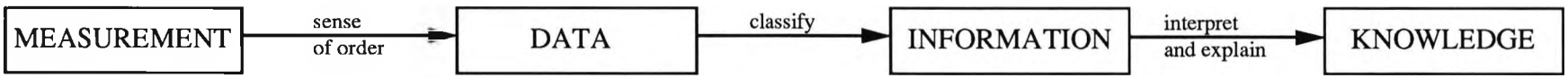


FIGURE 7.1 TRANSFORMATIONS WITHIN AN INFORMATION SYSTEM

This introduces the formal system which can be described in terms of mathematical entities and the mappings between them together with the relationships that they satisfy.

To establish the essential relationship between natural and formal systems, the entities and linkages contained within each system are deemed synonymous. Figure 7.2 depicts the degree of synonymity. Within the natural system there are two entities, 'A' and 'B', with a causal link between them, that is, a relationship exists between a cause and effect of an action in the natural world. The encoding relation transforms observables and linkages in the natural system via observation and measurement into symbols and propositions that represent 'A' and 'B' in the formal world. This defines a modelling relation between the natural and formal worlds. Within the formal system 'A' and 'B' can be linked in various mathematical ways, but of interest to this study is if they are linked by inference rules. That is, **IF** antecedent conditions are true **THEN** consequents fire an action (**IF** A **THEN** B). If a set of related entities from the natural system, which define some body of knowledge, are represented in the formal system by a set of inference rules, then the encoding relation can be termed knowledge-based modelling. Examples of knowledge-based systems from the medical domain which illustrate variations on this type of modelling can be found in Appendix I. The decoding relation can be of a multipurpose nature: for description, explanation, control and/or prediction. Most knowledge-based measurement systems (including AIRS) describe, explain and control (in open-loop mode) the processes that they represent. However, predictive models provide a powerful tool when it is important to gain information about a future state of a natural system. For example, being able to predict future patient state on the basis of some control action.

To be aware of how these philosophical issues relate to day-to-day activity within a critical care unit, the measurement system can be viewed from another perspective. Figure 7.3 shows the elements of a general measurement system, which when errors are ignored comprises the system of interest, the measuring system and the method of display. However, errors are introduced inherently into the measuring system, so it becomes important to realise that the measured value of an observation will not be exactly the same as its true value. A full account and classification of these errors can be found in the literature (for example, Barry, 1978; Hofmann, 1982).

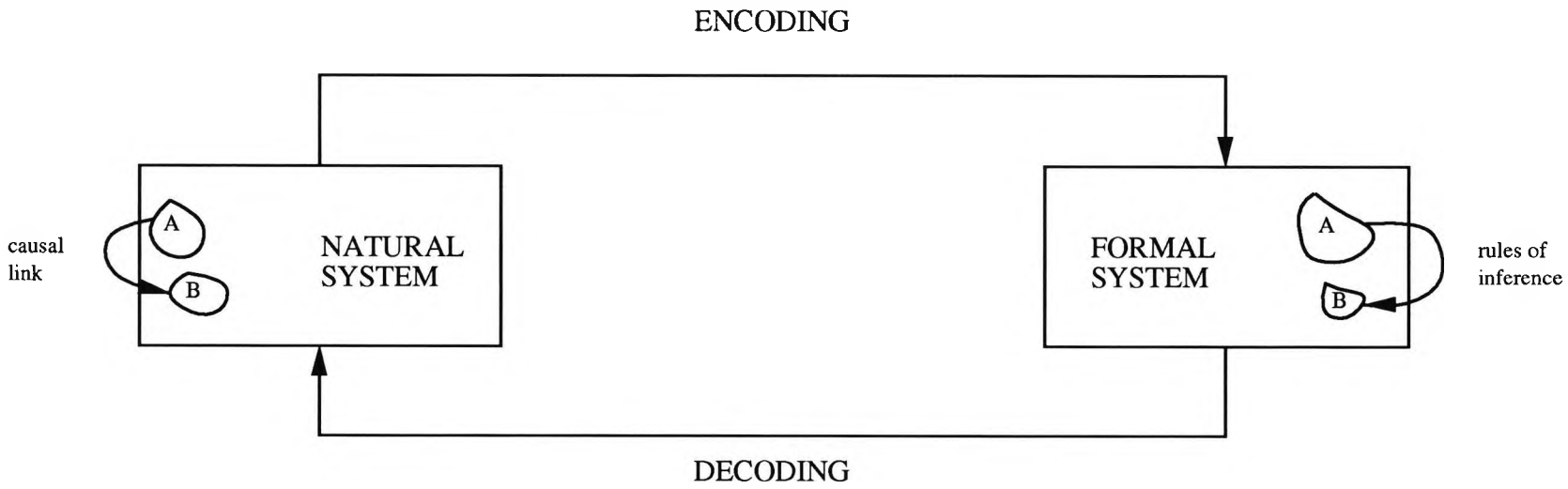


FIGURE 7.2 RELATIONS BETWEEN NATURAL AND FORMAL SYSTEMS

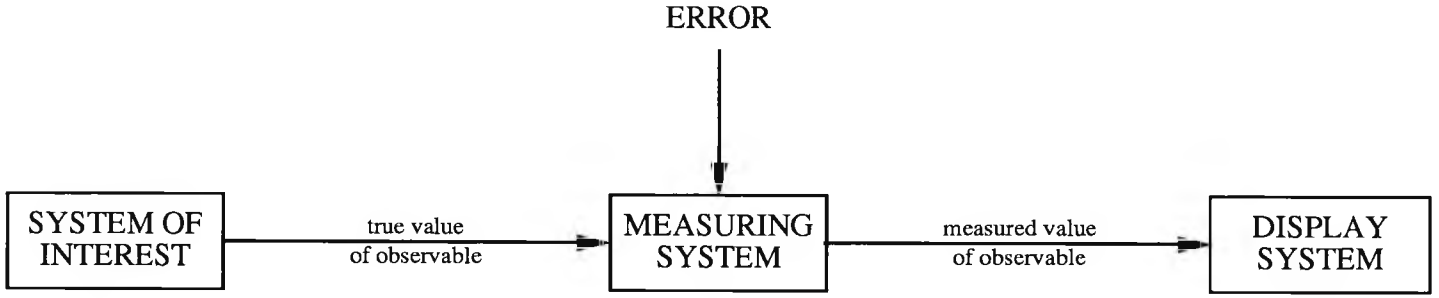


FIGURE 7.3 GENERAL MEASUREMENT SYSTEM

For the management of patients who require intensive care a large and diverse range of physiological data are required. There are also a range of measuring techniques to cope with the different types of data encountered (see Table 7.1). A set of sensors is required which can detect changes in pressures, flows, volumes, temperatures, inputs, outputs, input-output relations, biochemical processes and electrophysiological signals. Some sensors, such as those found in most modern ventilators (flow, pressure, volume and temperature), are in direct contact with the system of interest. Others have indirect contact, with varying degrees to which they are abstracted from the physiological process of interest. For example, for the on-line measurement of arterial blood pressure a saline-filled cannula is introduced into (say) the brachial artery of the arm, the other end being attached to a cuvette in which the pressure transducer resides and is external to the patient. The pulsatile nature of the arterial blood pressure can be observed through the transparent cannula as oscillations in the level of the blood-saline interface. Blood pressure is therefore recorded as the action of the saline fluid on the transducer. The accuracy of this measure depends on factors such as the compressability of the saline fluid, the rigidity of the walls of the cannula, and the characteristics of the pressure transducer itself.

For monitoring purposes a basic set of observations can be identified which allows the state of the patient to be inferred at any point in time. This set of variables differs slightly between institutions dependent upon the varying clinical specialities catered for within each hospital. When a time series has been collected the detection of trends becomes possible, allowing the patient-specific treatment plan to be optimised (Blom et al., 1985).

In critical care medicine it is often desirable to keep certain key physiological variables within well defined limits. If these limits are exceeded a control action brings about a change so that the system returns to within the accepted levels. AIRS demonstrated this feedback principle in open-loop mode: if minute volume decreased to an unacceptable level, advice was given to the user to raise the respiratory rate and/or tidal volume. Closed-loop feedback systems have also been used in clinical situations, for example to control arterial blood pressure in hypertensive patients by intravenous administration of nitroprusside (Sheppard, 1980).

As measurement technology has advanced there has been a

PRESSURES

- Arterial blood pressure (systolic, diastolic)
- Venous blood pressure
- Pulmonary capillary wedge pressure
- Intra-cranial pressure

FLOWS

- Blood flow
- Respiratory gas flow

VOLUMES

- Tidal volume
- Minute volume

TEMPERATURE

- Core temperature
- Peripheral temperature (big toe)

INPUTS

- drugs / doses

OUTPUTS

- Cardiac output

INPUT - OUTPUT RELATIONSHIP

- Fluid - electrolyte balance

ELECTROPHYSIOLOGICAL SIGNALS

- Electrocardiogram
- Electroencephalogram
- Evoked responses

BIOCHEMISTRY

- Haematological
- Liver function tests

TABLE 7.1 EXAMPLES OF PATIENT DATA

shift towards integrating microprocessors or microcomputers into biomedical instrumentation systems. This is evident in the ventilator component of AIRS, where a microprocessor sends and receives control signals from pressure, flow and temperature sensors mounted at strategic places in the ventilator system. This data processing functionality has been defined previously in the Introduction as giving rise to a 'smart' instrument; for an instrumentation system to be described as 'intelligent' there must be incorporated within it an information processing requirement for a knowledge-based component. Thus, this type of instrumentation system differs from conventional technology by having as a necessary requirement mechanisms to elicit expert-level knowledge and to represent that knowledge in a way that is programmable by computer. The way this is achieved in AIRS is discussed below, comparisons can also be made with other methods of knowledge elicitation and representation.

7.3 Knowledge Elicitation

It has been shown previously that the difference between 'smart' microprocessor controlled instruments and intelligent advisory systems is the level of knowledge embodied in the latter. This knowledge does not appear *de novo*, rather it is acquired from the literature and human experts. The process of knowledge acquisition has been described as a "bottle-neck in expert system construction" (Hayes-Roth et al., 1983), so it is not suprising to see a plethora of techniques becoming available to try and address this issue. This section will identify some of the most popular techniques for knowledge elicitation, that is, that branch of knowledge acquisition which deals with human experts. It is common for experts to fall into one of three categories: academics, practitioners and domain 'Tsars'. Generally, academics provide a theoretical base for a given domain; practitioners have experience of using a particular body of knowledge and therefore provide a pragmatic base; and domain Tsars have the ability to combine theory and practice. Knowledge engineers must be aware of the scope and limitations of their human experts, and be prepared to accept that the perception one expert has to a particular problem may not be the same as another. In situations where there is a clash of opinion, a 'gold standard' should emerge when knowledge is elicited from a range of experts. It is recognised that this method of acquiring a gold standard may not be feasible in some domains. For example, in situations where there is a lack of agreement between experts, problem solving may be considered an art rather than a

science, and therefore any attempt at formal representation of the problem domain is unlikely to succeed.

To compound the difficulties faced by the knowledge engineer there are different types of knowledge, given by the following epistemological classification of expertise (Shadbolt and Burton, 1989):

i) Domain level knowledge

This is defined as knowledge that describes the concepts and elements in the domain and the relations between them. That is, a declarative description is obtained about what is known for each concept in the domain.

ii) Inference level knowledge

This is defined as knowledge about how the individual components of expertise in the domain are organised and used in the overall system; this type of knowledge may be implicit from a pragmatic viewpoint.

iii) Task level knowledge

This is defined as procedural knowledge which deals with how goals should be reached.

iv) Strategic knowledge

This is defined as knowledge which monitors and controls the systemic problem-solving strategy. For example, strategic knowledge deals with problems associated with conflict resolution.

Thus, if the overall aim of knowledge elicitation is to specify a body of knowledge which is complete, consistent and as correct as possible, then several overarching questions must be borne in mind:

* What is the role of the intelligent advisory system?
* How is the domain mapped out in terms of significant concepts and relationships?
* What is the task structure of the domain? Are there any 'special' relationships between concepts?
* Is there a need for more than one type of strategic knowledge?

Knowledge elicitation can therefore be a complex process, best dealt with by a range of techniques. In this study a 'structured

interview' elicitation technique is used and is discussed below in some detail. Other elicitation techniques are discussed for comparative purposes and indicate potential areas for future development in this field.

7.3.1 Structured Interviews

The most popular knowledge elicitation techique is the structured interview. There are various formats that can be used within a structured interview, two of the simplest methods are outlined below. Using the first method the expert is given a list of variables that are used to describe the domain and a list of conclusions which describe all possible outcomes of the system. Production rules of the format:

IF <variable 1> <value>
and <variable 2> <value>
⋮
and <variable N> <value>

THEN <conclusion A>

can then be derived which link the two lists. It is important for the knowledge engineer to attempt to formalise all rules stated, whether they be explicit or implicit in nature.

In the second method the expert is given a brief verbal outline of the target task, such as 'weaning patients off ventilators using synchronised intermittent mandatory ventilation'. Topics covered in the outline should (as before) include a description of the possible solutions to the problem and a description of the variables which may affect each solution. However, in this method a list of major rules which connect the variables to the solution should also be stated explicitly. The task for the knowledge engineer is then to take each elicited rule in turn and enquire about the conditions required which increase or decrease its appropriateness. This procedure reveals the scope of each rule, and in its test for relevancy new rules may be generated. This cyclic nature of rule generation continues until it is clear that the expert will not produce any additional information.

It is possible to use both of these methods in conjunction with one another, where the first method produces the rule list for the second method. In both cases the knowledge engineer should attempt to keep his contribution to the structured interview process

to a minimum. This can be achieved by careful phrasing of the trigger questions. A **"Why..."** query is useful as it converts an assertion made by the expert into a rule. Similarly, a **"How..."** query generates rules at a lower level in the knowledge hierarchy. A **"What if..."** question simulates a forward scenario for which new rules may be generated. To reveal the scope of a rule, or of a set of rules, a **"When..."** query could be used which could also act as a catalyst for a new rule set. Finally, if the goal of the knowledge engineer is to generate further dialogue when the expert has come to a temporary lull, the **"Can you tell me more about..."** is a good device to use.

In the development of AIRS both methods of performing structured interviews were employed to elicit knowledge. For example, after identification of the seven variables which appear on the AIRS-maintain screen, the possible action and explanation outcomes were elicited by forward scenario simulation of each alarm state. Conversely, the full set of trigger question types were required to elicit knowledge for AIRS-wean. Concepts used in weaning patients off ventilators were eventually organised into a hierarchy, described previously in Section 6.3.3.

The advantages of the structured interview approach to knowledge elicitation are threefold: the expert only has to put aside a small amount of time in what is usually a busy schedule; the experts generally enjoy this method of knowledge elicitation; and the method can be used to elicit knowledge from any domain. However there are also a number of disadvantages which can be split into two sub-groups: those which deal with the expert as subject; and those which deal with the knowledge engineer.

In the former category it is sometimes difficult to get experts to obey instructions as generally they do not respond in a systematic manner and are prone to give anecdotes rather than rules. In some domains it is difficult to establish clear rules in production rule format. To obtain maximum efficiency of transfer between the human expert and the computer-based system it is essential for the expert and the knowledge engineer to have a good rapport. To engage in any meaningful knowledge transfer it is worth spending some preliminary sessions mapping out the knowledge domain, which also enables the knowledge enginer to have an opportunity to get to know the expert and his way of conceptualising a problem.

From the point of view of the knowledge engineer the

structured interview technique requires a priori knowledge of the domain so that trigger questions are meaningful and lead to the capture of a large amount of rules. For the knowledge transfer sessions to go well the knowledge engineer needs to be astute and alert to changes in theme. From a resource stand-point this method is very time consuming for the knowledge engineer. To aid the ease of knowledge elicitation most meetings are tape-recorded from which a written transcript is obtained, and many iterations of the technique are necessary before a 'complete' rule set is obtained.

7.3.2 Other Techniques for Knowledge Acquisition

Although the structured interview technique was favoured for the elicitation of knowledge for AIRS, various other techniques have become available in recent years, including: Protocol analysis; 20 questions; Laddered grids; Repertory grids; and Automatic techniques. A brief description of each follows.

Protocol analysis is a generic term used to describe several similar methods of knowledge elicitation. Their common theme is that the knowledge engineer records the routine activity of an expert during the solving of a particular problem task. Protocols are then made from the written transcripts of the record (for example, on audio/visual tapes) from which rules can be extracted. To be successful the knowledge engineer must be acquainted with the domain of interest; if this is not the case an alternative strategy can be used where another expert 'shadows' the expert performing the problem-solving task in order to explain the nuances of what is happening. In general protocol analysis elicits the **"when..."** and **"how..."** of using different items of the knowledge base as well as revealing the various reasoning strategies. A problem with protocol analysis is that it is very time consuming, and only a small section of the domain can be addressed each session.

The **20 Questions** knowledge elicitation technique is useful in the initial stages of knowledge structuring. Initially the expert is provided with little or no information about a particular problem to be solved. The expert must then ask the knowledge engineer for specific data or information that is required to solve the problem, (the knowledge engineer can qualify any answer given). Using this format, processing rules can be formed from the requests for information from the expert, where not only **what** is requested but also in **what order** the information is requested, are important knowledge

sources.

An obvious disadvantage of this technique is that the knowledge engineer has to be an expert in the domain to answer questions from the expert. Other disadvantages include: the technique is time consuming to prepare, it is sometimes difficult to infer rules, and experts do not like being relegated to role of subordinate.

The **laddered grid** technique provides a graphical representation for a hierarchy of domain knowledge. A qualitative two-dimensional graph is produced which shows 'concept nodes' connected by 'labelled arcs'. To construct this graph the knowledge engineer and expert confer, starting with a seed item which is mutually agreed. The position of the nodes in the domain map is an important item of information and several verbal devices can be used to distinguish the place for each node. These include, "**How....**" or "**Can you give an example....**" to move **down** the conceptual map; "**What....**" to move **up** the conceptual map; and "**Give me alternatives....**" to move across. The knowledge engineer can choose the order in which the map is drawn by using the appropriate verbal cue.

A **repertory grid** is a technique in which the expert is presented with a set of cards. Each card has a single concept written on it and represents an element in the domain of interest. To construct the repertory grid the expert chooses three cards, two of which are similar and one of which is different. A label is given to the feature which provides the reason for discrimination. This process continues with different triads of elements until the expert can think of no further discriminating constructs. The outcome of this technique is a matrix of 'similarity ratings' which relate the elements and their labels. To analyse the results the statistical technique of cluster analysis is used. Rules can be elicited directly from the grid.

The advantage of this technique is that it uses a formal method to elicit information which may not have been forthcoming using a conventional interview technique. However, analysis can be very time consuming and sometimes difficult to interpret. To aid the knowledge engineer, there is a software program available which performs the repertory grid analysis interactively with the domain expert (Shaw and Gaines, 1987). This technique is best suited to small scale domains even when the computer-aided method is used.

In each of the knowledge elicitation techniques discussed thus far, a limiting feature has been that the knowledge engineer acts a filter for information and if a control engineering analogy is taken some signal is invariably lost. To counter this problem **automatic knowledge elicitation** procedures are becoming popular. These are techniques where the expert imparts knowledge directly into the computer-based system. There are two main methods in use: machine induction and knowledge-base browsing.

Machine induction has the longer history, with perhaps the best known algorithm for rule capture being ID3 (Quinlan, 1979). The ID3 algorithm is implemented on a computer and uses a statistical pattern recognition technique to infer rules from training examples that the experts provide. These training examples are solved problems from the domain of interest. An expert system shell is available which is based on this algorithm ('First Class', Programs in Motion Co.). The machine induction technique does not provide a complete solution to automatic knowledge elicitation as the knowledge engineer together with the expert have to decide what metrics to measure to reach particular conclusions. A major disadvantage of machine induction is its way of handling causality. The fact that a particular set of precursors co-occurs with a particular conclusion does not necessarily imply a causal relationship.

A more recent development in automatic knowledge elicitation is the knowledge base browser approach (Chelsom, 1990). This method has become possible due to the rapid development of computer hardware and software, where a major emphasis has involved the human-computer interface. In this technique the expert first establishes a taxonomy of classes in the form of a hierarchy. The data which are required to provide information about the whole domain are established. The task of the expert is to then go through each 'knowledge node' systematically, detailing which items of data are important for that particular node and to indicate at what value each item of data change their levels of significance. A subjective measure of 'belief' is also added at this stage. The outcome of this procedure is the definition of a 'problem hypothesis space' which can be interrogated at will. A disadvantage of this technique is that the problem to be studied has to be classifiable into a hierarchy. The expert must also be familiar with state of the art technology used at the human-computer interface.

7.4 Knowledge Representation

Knowledge representation can be achieved in numerous ways: frame-based systems (eg. PIP); systems which use semantic nets (eg. CASNET); and the use of a blackboard architecture for software control and knowledge representation (eg. HEARSAY-II). Further discussion of these techniques and systems, together with an operational description of MYCIN which uses a rule-based formalism, can be found in Appendix I. As AIRS uses a rule-based representation itself, this method of knowledge representation is discussed below.

7.4.1 Rule-based Systems

There are three necessary constituents which make up a rule-based system : a database of pertinent facts which can be updated and modified; a knowledge base consisting of rules which relate items of data; and an inference engine which matches rules in the knowledge base with facts about a particular event as they are entered by the user. Thus, the information model that this type of system uses is to infer intermediate data findings from the rules executed from data input by the user, which has the effect of gaining extra information about a particular problem. The rule execution cycle continues until an impasse is reached, that is, when no conclusions can be formed from the data input by the user (the system then may request more data), or the top-goal of the system can be concluded from the data already available, this stops the execution cycle.

Rule-based systems are typified by a production rule inference mechanism, shown in Table 7.2:

IF	<antecendent 1> <antecendent 2> ⋮ <antecedent N>
THEN	<consequent >

TABLE 7.2 The Format for a Production Rule

Each system consists of many rules of this type, these may be chained together by allowing rule antecedents in one part of the rule-base to be a consequent of another rule in another part of the rule-base. For example, let a mythical list of antecedents and consequents for a particular application be represented by the letters A to G; further, let two rules be defined as described in Table 7.3:

RULE 1			RULE 2		
IF	i)A	and	IF	i)E	and
	ii)B	and		ii)F	and
	iii)C			iii)G	
THEN	D		THEN	C	

TABLE 7.3 Two General Production Rules

This example illustrates the principle of rule chaining, as in evaluating the antecedents of Rule 1 a consequent of another rule (that is, Rule 2) is met. Thus, for the consequent D to be asserted as a fact, sub-goals A,B,E,F and G have to be evaluated. In a simple system these sub-goals could all be user queries about patient findings which establish the existence of a pathphysiological disorder. In some systems, including MYCIN, the consequents can also have weightings attached to them which indicate the likelihood of a particular event occurring. When rules chain together these likelihood values interact in a way to reflect the conditional probabilities of two or more events happening simultaneously.

Another way of conceptualising the rule-base is to think of the rules organised into hierarchical sets, sometimes referred to as 'goal trees' because of the order of rule execution. In the simple two rule example above this corresponds to the representation illustrated in Table 7.4:

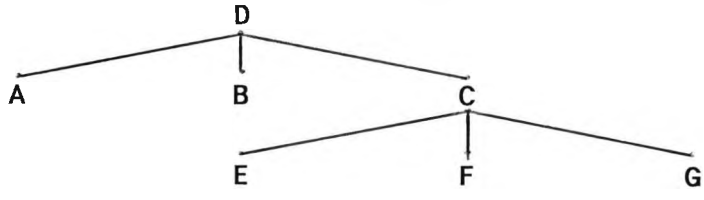


TABLE 7.4 A 'Goal Tree' Representation

By considering the rule-base as an ordered hierarchical set, both the control strategy and explanation facility of rule-based systems can be formulated.

When the inference engine matches the consequents of rules in the knowledge base with data in the database a forward-chaining inference mechanism is defined. Conversely, if the match is with the antecedents of the rules, then a backward chaining inference mechanism

is the outcome.

In more detail, the forward-chaining mechanism starts with data in the database from which all antecedents of satisfiable rules are fired in order of appearance in the rule-base. The successful antecedents then become further items of data and are added to the dynamic database. Of course, the antecedents which succeed may be the consequents of rules at a higher level in the goal tree, and so the inference mechanism is reiterated. This process continues until more data are requested or the top-goal is reached. In systems which use a forward chaining inference mechanism, for example the OPS5 environment (Forgy, 1981), the efficency of the system is determined by the order of rules in the rule-base.

Backward chaining inference systems start with the top goal in the goal tree from which its consequents must be evaluated. These invoke further rules whose antecedents are the consequents of the top goal. Again, this procedure continues until confirmatory data required by the system are requested. If these data already exist in the database the leaf nodes succeed, that is, the goal nodes at the base of the goal tree are satisfied. These in turn send a wave of goal successes back up the goal tree, ultimately leading to the success of the top goal rule. However, if the confirmatory data does not exist in the database, goals which require user interaction are fired in order that the required data may be input. The EMYCIN shell used by the MYCIN system uses this form of inference mechanism, where the top goal is a therapy rule which asks if there are any organisms present which require therapy (Shortliffe, 1976). The normal execution cycle of PROLOG also mimics a backward chaining mechanism.

In AIRS there are three types of rule : weaning, regression and progression. The top goal of the weaning rule set is given in Table 7.5:

```
premise  (0.1,  weaning,  [fit-to-wean])  :-
equals   (0,1,  weaning,  fit-to-wean, yes).
```

where
first argument is the Rule Number
second argument is the Premise Number
third argument is the Rule Type
fourth argument is/are the Dependent Variables.

TABLE 7.5 The Top-goal of the Weaning Rule Set

To satisfy the goal 'fit-to-wean' six further rule hierarchies have to be satisfied (see Section 6.3.3). A backward chaining inference mechanism is used where the leaf nodes fire questions to the user in order to add items of data to the database.

Two types of explanation are possible in these systems : WHY explanations and HOW explanations. WHY queries are dealt with by ascending the goal tree and show which items of data are required to determine the antecedents of sub-goals. Conversely, HOW explanations descend the goal tree, chaining together the events which are necessary to reach a particular conclusion. In AIRS, both types of explanation are given in terms of a list of the goal nodes passed through to reach a particular conclusion.

One problem encountered in rule-based systems is that of conflict resolution; that is, if a condition arises that both negates and satisfies a particular rule. The most common solution to this impasse is to invoke a meta-rule. In AIRS there is one meta-rule, which is shown below in Table 7.6:

IF	**progression rules succeed and regression rules succeed**	
THEN	**regress**	

TABLE 7.6 AIRS Meta-rule

Thus, AIRS errs on the side of safety. This is an important principle which many researchers take one step further (Table 7.7):

IF	**no conclusion(s) apparent**
THEN	**do nothing**

TABLE 7.7 A General Meta-rule

Here, if the system is unable to come to any conclusion then current opinion dictates that the system takes no action rather than an action which may be erroneous. Although this maxim can be applied to any type of knowledge representation it is particularly true for production rule systems where there is usually only one consequential action.

7.5 The Human-Computer Interface

There are many synonyms which describe the interface between the (human) user and computer, including 'man-machine interface' or more simply 'user interface'. These are attributable perhaps to the relevant newness of the area of research. The choice here of 'Human-Computer Interface' (HCI) is deliberate, for it is a phrase which describes the juncture between two of the components featured previously in the design phase of AIRS (see Figure 5.1).

Research themes which include the investigation of the HCI are popular due to the rapid proliferation of computer systems in all sectors of society. Whereas there may be an increasing number of people able to program computers, there is still an order of magnitude greater number of people who can be described simply as computer operators. This multi-perspective view of the HCI becomes more apparent when dealing with knowledge-based systems in highly complex environments such as that exhibited by a High Dependency Environment. For example, at certain times there may be a conflict of interest between efficient programming capability and the interface requirement. To identify the pertinent issues involved in HCI design, the user requirements for the system are an important source of information. It is essential that computer programmers (system designers) take into account these issues when creating computer-based systems for widespread use, for if not it is probable that the system, when implemented, will remain unused.

An integrated approach to HCI issues is required which matches what the user wants from the system to the type of interface available using the programming environment. Traditionally the HCI consists of a keyboard for input into the computer system, and a visual display unit (VDU) or printer for display of the output. By using an integrated approach, the HCI can be categorised in terms of type of alternative device for input/output; graphics capability of the system; and other more specific configurable features.

A need for peripheral devices which provide an alternative method for input into the system became apparent as soon as computer-based technology reached the business community. Many computer operators have limited keyboard skills which means that as a consequence data input is a rate-limiting step. To counter this problem two types of input device have evolved : the pointing device which must be used with associated software; and devices which use

pressure-sensitive sensors. The most popular pointing device is the mouse, although it is becoming evident that control of a trackerball requires less dexterity. The trackerball also has the advantage of having a smaller 'footprint'. A transparent pressure-sensitive overlay placed on a VDU screen is the sensor for another pointing device - the finger of the user. More recently the sensing element has evolved to an infra-red matrix, this method of location is more reliable than its predecessor. In the design phase of AIRS an external pressure-sensitive graphics pad is specified as an alternative input medium (described in Chapter 5).

Changes in input/output device have heralded further changes in the quality of screen interface. Graphics packages are becoming the norm, a 'windows interface' with associated menus and pull-down screens being a popular configuration. It is also possible to perform multi-tasking with advanced graphics packages. This has obvious advantages in complex environments, for example, data can be collected continuously from on-line monitoring devices whilst historical trends or trend predictions are being computed and displayed simultaneously.

Other features of the HCI include its capability for handling communications, being user-friendly for all types of end-user, being maintainable and its robustness. Increasingly, stand-alone computers are being networked into larger communications systems, either via a local network or as part of a wider commercial communications system. This is an attractive feature for computer systems employed in High Dependency Environments, as messages relating to patient care can be left on the system or sent to/from a remote source. If the system is connected to a commercial network it is possible to form diagnostic-related databases for the benefit of future patient care. For example, by its very nature a rare clinical condition occurs infrequently, the chance that the same hospital will admit further similar cases is therefore also rare. However, if a clinical database for all such cases is created at one national site, clinicians can send electronically the details of particular patients which will complement the database. In return the clinician can receive management guidelines generated from previous attempts at treating similar conditions. Indeed, there is nothing to prevent the formation of an international database for particular clinical findings. Creation of such databases would enable research into the clinical time course of the disease and could lead to improved therapy planning.

Research into how users use computer-based management systems has revealed that expert users may look at specific items whereas the novice user requires a more general picture. The notion here is that the expert has a 'mind model' based on experience of looking at similar cases, and uses the system to fine-tune some of the parameters. However, the novice user has no such experiential knowledge to draw upon, and therefore uses the system to help form a model. Any HCI must therefore cater for both types of user - the expert 'vertical-mode' user and the novice 'horizontal-mode' user.

For the HCI to be maintainable there is a requirement for modular programming. This allows redundant features of the HCI to be removed and new features or upgrades of existing features to be added. The HCI must also be robust for users to gain confidence in using the system.

Although the current 'windows-type' HCI of AIRS is attractive and easy to use, it is too slow. The requirement for fast access into the system and faster screen management can now be achieved using commercial software packages written in real-time programming environments. Extra facilities are also available, for example, a notepad data entry screen for free-text input together with a word-processing functionality. Both of these facilities are useful for generation of patient care reports.

The HCI of the future may be more general than those available at present, sculpted by a user interface management system (UIMS). Essentially the UIMS is a software device which is placed between the user and the application program and graphical presentation manager. It comprises a set of software tools which support the implementation and evaluation of interactive human-computer dialogues. The overwhelming advantage of this type of system is that high quality user interfaces can be generated by the end-user. The result of this collaboration should be that the software application will be used effectively and efficiently because the user has defined the HCI rather than the applications programmer.

7.6 Evaluation

In the formulation of a strategy for evaluation, analogies have been drawn from evaluation studies in domains which have a richer history than knowledge-based technology. These are: mathematical modelling; clinical trials of a new pharmaceutical; and the social

sciences. Figure 7.4 shows a different perspective of knowledge base development than used previously in this study, its emphasis is on knowledge processing rather than whole system development. The figure illustrates the mappings between a basic process model of knowledge engineering and the evaluation issues that can be 'covered' by strategies in the analogical systems.

The validation of mathematical models can be defined as the extent to which a model satisfies the objectives for which it was formulated. Two types of criteria can be applied to such models; internal and external (Leaning, 1980). The mappings in Figure 7.4 indicate that the internal criteria cover the development of the intelligent decision aid, whilst the external criteria cover program tuning and the goal state where data are required for validation purposes.

Internal criteria for mathematical model validation comprises consistency, completeness and algorithmic validity. Consistency and completeness refers to the finding that the model should not contain any logical, mathematical or conceptual contradictions, and should represent all possible aspects of model configuration. Algorithmic validity refers to the requirement that any algorithm used in the model is appropriate and leads to accurate solutions. The analogies to knowledge-based technology drawn from this work are clear - consistency and completeness can be applied directly to the development of the knowledge base, and algorithmic validity can be applied to the adequacy of knowledge representation.

External validation comprises empirical validity, theoretical validity, pragmatic validity and heuristic validity. Empirical validity describes the extent to which the mathematical model corresponds to available data over the intended range of application; theoretical validity requires any model to be consistent with accepted theory; pragmatic validation describes the extent to which the model satisfies its initial objectives; and heuristic validation assesses the potential of the model for scientific explanation. The external validation criteria for development of mathematical models map into the issues of system effectiveness, acceptability and usability when applied in the knowledge-based system domain.

Clinical trial protocols can be divided into three phases. In Phase I the safety aspect of the drug is determined, for example, the measurement of maximum tolerated dose. Phase II tests the efficacy of the drug, and Phase III uses a multi-centre trial to assess the

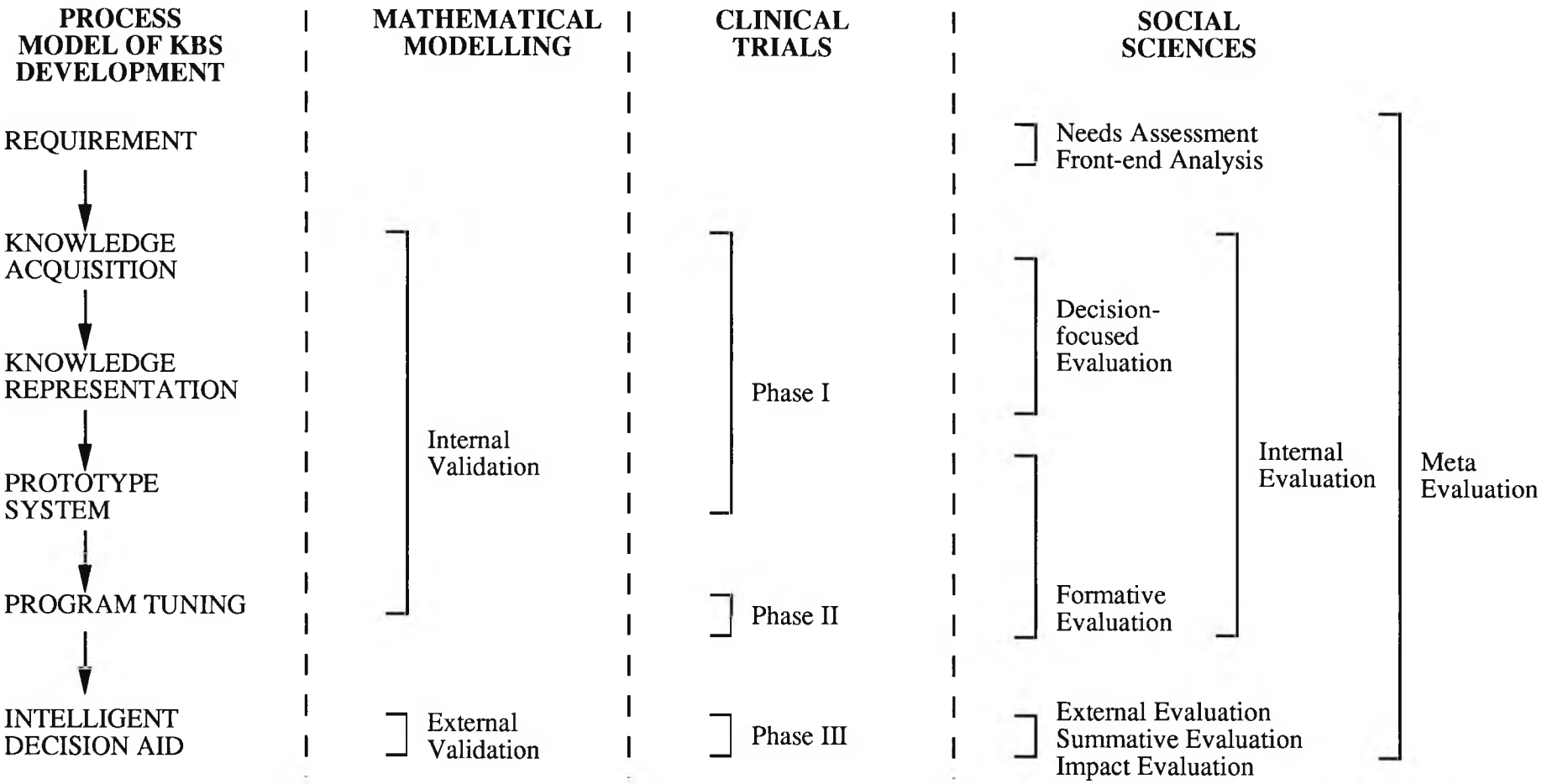


FIGURE 7.4 EVALUATION STRATEGY COVERAGE

effects of the new treatment regime and compares it to the best available existing therapy. From Figure 7.4, it can be seen that the safety aspect has been mapped into the process for producing the first prototype system from the initial stage of knowledge acquisition. Clearly, the Phase II objective of producing the desired effect can be mapped into the program tuning phase of system development. The third phase of multi-centre trial maps to the evaluation of the goal state, and can be considered as summative evaluation.

The methodology described in this thesis suggests one way of introducing an intelligent information system into a pre-existing data processing environment. Its emphasis is on the evaluation process, as this has been identified as a route towards ensuring a high user acceptance. Different facets exist in the evaluation process, demonstrated by the lack of consensus for its definition and terminology when applied to intelligent systems. A possible (and candidate) solution to this problem comes from the social sciences, where a 'user-focused' approach has been used to define a multi-perspective view of evaluation, as follows:-

> "...the systematic collection of information about the activities, characteristics, and outcomes of programmes, personnel, and products for use by specific people to reduce uncertainties, improve effectiveness, and make decisions with regard to what those programmes, personnel, or products are doing and affecting. This definition of evaluation emphasizes 1) the systematic collection of information about 2) a broad range of topics 3) for use by specific people 4) for a variety of purposes."
>
> (Patton, 1982)

This definition is favoured because it places emphasis on a systematic approach to the information requirements of the different end-users. The "systematic collection of information" reflects the information gathered at each stage of the methodology; the "broad range of topics" reflect the different types of evaluation which cover different aspects of the underlying phases of the methodology; the specific users of the system are identified at an early stage in the methodology; and the "variety of purposes" are reflected in the interest in the evaluation shown by the different type of user.

Hence, evaluation protocols taken from the Social Sciences have the broadest scope, and the types of evaluation from this domain illustrated in Figure 7.4 are by no means exhaustive. The various strategies shown are also not mutually exclusive, as indicated by the

high degree of integration. Unlike the domains of mathematical modelling and clinical trials, evaluation in the Social Sciences investigates the requirement for a proposed system. A needs assessment matches what the user requires to solve a problem with the resources that are available. Front-end analysis describes a more strategic level of analysis than a needs assessment, providing guidance in the planning and implementation of programmes. Internal evaluation is performed 'in-house' and comprises decision-focused evaluation and formative evaluation. Decision-focused evaluation provides the information required to make a specific decision on progress at a specific juncture in the programme; and formative evaluation investigates how the present system can be improved. The former maps into the development of the knowledge base, whereas the latter covers the iterative process of program tuning. External evaluation is performed by 'third party' assessors, and can consist of summative evaluation and impact evaluation. In the Social Sciences, summative evaluation investigates whether or not the programme should be continued; impact evaluation considers the extent to which there is a change in the environment in which the programme resides. In the knowledge-based domain this can represent the evaluation of the goal state. Also, in the Social Sciences the evaluation process of a programme is sometimes evaluated, this 'meta evaluation' covers the entire development process, from initial requirement to achievement of the final product. Such a concept would also be useful in the knowledge-based domain.

It is evident from this study that the evaluation of knowledge-based systems should be considered as an essential and on-going activity. However, this process is hampered in the medical domain, being partly due to its multi-perspective nature in a clinical setting. Other factors which must be overcome include the volume and complexity of the incoming information, the time and other resource demands made on already busy personnel to perform the evaluation study, and the professional sensitivity of clinical personnel involved in the management of the evaluation study. To combat these difficulties the evaluation process needs a clear and coherent structure. In formulating an evaluation strategy it has been shown how the evaluation process from three different domains have been drawn together in order to synthesise such a structured framework. This process was incorporated into a novel methodology for intelligent system design, implementation and evaluation, as discussed previously.

8: CONCLUSION

Part I of this study provided the necessary background and historical information for the introduction of intelligent instrumentation into a High Dependency Environment. As a key element in this process, a novel methodology was proposed which provides a structure for the incorporation of a knowledge-based component into the measurement process. This methodology covers the entire development process, from the requirement for a particular measurement system to the evaluation of the final product. As a consequence the methodological weaknesses of existing approaches (for example, from either a software or control engineering paradigm) were highlighted and the benefit of using the novel methodology espescially designed for technological change was exposed. For instance, evaluation of the system throughout its development phase was identified as a key component in the procurement of the final product. The methodology is iterative in nature and provides support for system evolution and is therefore necessarily open-ended.

Part II detailed a specification and implementation for a prototype patient management system designed for those patients who require ventilatory therapy as part of their overall management strategy. Three levels of decision-support were implemented to match the three phases of ventilation identified: the start-up phase of ventilatory support was implemented using simple deterministic rules obtained by analysis of a database of retrospective patient data; the maintain phase was equated with a pattern-matching algorithm, the boundary values of the data classifiers in the algorithm were established by consultation with expert clinicians; and the weaning phase was implemented using production rule technology which again required expert clinicians from which knowledge could be elicited.

Evaluation of the prototype system, which was implemented entirely in PROLOG, had some unexpected findings. For example, novice users were more impressed with the windows display of the start-up advice screen than those which were implemented using more advanced knowledge representations. One reason for this conclusion is that the more advanced knowledge technology did not match user expectation in terms of time taken to reach a recommendation for action. A paradox becomes evident here, as although the time taken for a human expert to arrive at the problem location (and then reach a decision for action) may be orders of magnitude greater than the time the computer takes to

reach its decision, users still expect computer-generated advice to be instantaneous. The underlying psychology responsible for this outcome may be due to the fact that time is an important commodity in the Critical Care Unit, and clinical staff should not waste it waiting for the output from slow-acting high technology found at the bedside.

From this phase of the research programme several general conclusions can be made that would enable successful introduction of intelligent instrumentation systems in the Intensive Therapy Unit. These include:

* the system must make life easier for staff in the Intensive Therapy Unit, especially the nurse who is the main care provider.
* the system must improve the quality of data available.
* interpretation of data within the system must be made in context of the clinical situation at that time.
* the system must include a facility for trend prediction
* the system should make clinical audit an easier task.

Also, several technico-clinical problem areas were identified from the use of ventilatory therapy as the application in this study, including:

* data overload and complexity of measurement in the ITU.
* the continuity of patient management practices through the different phases of ventilation.
* an appraisal of the worth of each item of monitored data.

These conclusions emphasise the systematic nature of the systems enquiry. Particular problems were broken down into their constituent parts before problem-solving was attempted.

Part III discussed the nature of intelligent instrumentation and techniques used to incorporate the knowledge component into the measurement process used in the development of AIRS. Comparisons were made with other available methods of knowledge elicitation, knowledge representation schemas, human-computer interface techniques and evaluation issues. The nature of intelligent instrumentation from a wider perspective was also included. From discussion, recommendations for further work became apparent.

In the immediate future a reorganisation of programming tasks should take place, so that PROLOG is used solely for that part of the program where logic is required. That is, the part of the system which is represented by production rule technology. The window environment and associated menu system can be implemented more efficiently in other high level languages such as 'C' or 'PASCAL'. This would also enable the development of a graphical capability. To gain credibility work should also commence on an on-line version of the system. For this an appropriate communications package will be required as well as a ventilator with a digital port. However, in the long term this type of single function program may become redundant, superseded by machines capable of processing many parallel functions at once, the data being pre-processed by transputers at the bedside. A programming environment such as a parallel version of PROLOG, as used in the Japanese Fifth Computer Generation Initiative, will become the software of choice, although dedicated parallel PROLOG machines may become the norm. This may be conjecture, but intelligent instrumentation systems have a bright future and are set to pervade all measurement processes no matter what the domain.

REFERENCES

AIKINS,J.S., KUNZ,J.C., SHORTLIFFE, E.H. and FALLAT, R.J. (1983). PUFF: An Expert System for Interpretation of Pulmonary Function Data. Comput.Biomed.Res., 16, 199-208.

ALFORD,M. (1985). SREM at the Age of Eight; the Distributed Computing Design System. Computer, 18 (4), 36-46.

BALZER,R., CHEATHAM,T.E. and GREEN,C. (1983). Software Technology in the 1990s: Using a New Paradigm. Computer, 16 (11), 39-45.

BARNEY,G.C. (1985). Intelligent Instrumentation: Microprocessor Applications in Measurement and Control. London:Prentice Hall.

BARRY,B.A. (1978). Errors in Practical Measurement in Science, Engineering and Technology. Ed. M.D.Morris. New York: Wiley-Interscience.

BERTRAND,O., VIALE,J.P., ANNAT,G., SEBES,F., DELAFOSSE,B., PERCIVAL,C., BUI-XUAN,B. and MOTIN,J. (1986). Mass Spectrometer System for Long-term Continuous Measurements of vO_2 and vCO_2 During Artifical Ventilation. Med. & Biol Eng. & Comput., 24, 174-181.

BHANSALI,P.V. and ROWLEY,B.A. (1984). A Microcomputer Controlled Servo-ventilator. J.Clin. Eng., 9 (1), 47-51.

BLEICH,H.L. (1972). Computer-based Consultation: Electrolyte and Acid-Base Disorders. Amer.J.Med., 53, 285.

BLOM,J.A. de ROYTER,J.A.F., SARANUMMI.N. and BENEKEN,J.E.W. (1985). Detection of Trends in Monitored Variables. In: Computers and Control in Clinical Medicine. Ed. E.R.Carson and D.G.Cramp. New York:Plenum.

BOOTH,F. (1983). Patient Monitoring and Data Processing in the ICU (editorial). Crit. Care Med., 11 (1), 57-58.

BRATKO,I. (1986). Prolog Programming For Artificial Intelligence. Addison-Wesley: Wokingham.

BRITISH MEDICAL ASSOCIATION.(1967). Intensive Care. Planning Report No.1.

CARSON,E.R., CHELSOM,J.J., CRAMP, D.G., SUMMERS, R. and ZARKADAKIS, G. (1988). Towards the Development of an Intelligent ITU Workstation. In: IEEE Engineering in Medicine and Biology Society 10th Annual Conference. Ed. G. Harris and C. Walker. New York:IEEE, 1414-1415.

CARSON,E.R., CRAMP,D.G., and FINKELSTEIN,L. (1986).
Towards Intelligent Measurement in Critical Care Medicine.
In: IEEE Engineering in Medicine and Biology Society. 8th Annual Conference Ed. G.V.Kondraske and C.J.Robinson. New York:IEEE, 799-801.

CHECKLAND,P.B (1981).
Systems Thinking, System Practice.
Chichester: John Wiley.

CHELSOM,J.J.L. (1990).
The Interpretation of Data in Intensive Care Medicine: An Application of Knowledge-based Techniques. PhD Thesis. City University, London, UK

CLANCEY,W.J. and LETSINGER,R. (1981).
NEOMYCIN : Reconfiguring a Rule-based Expert System for Application to Teaching. In: Proc. 7th IJCAI, 829-836.

DAVIS,R. and KING,J. (1977).
An Overview of Production Systems.
In: Machine Intelligence 8 : Machine Representations of Knowledge.
Ed. E.Elcock and D.Michie. New York : Wiley.

de DOMBAL,F.T., LEAPER,D.J., STANILAND,J.R., McCANN,A.P. and HORROCKS,J.C. (1972).Computer-Aided Diagnosis of Acute Abdominal Pain. British Medical Journal, 2, 9-13.

EASTERBY-SMITH,M (1980).
The Design, Analysis and Interpretation of Repertory Grids.
Int.J.Man-Machine Studies, 13, 3-24.

ERMAN,L.D., HAYES-ROTH,F., LESSER,V.R. and REDDY,D.R. (1980).
The HEARSAY-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty. Computing Surveys, 12, 213-253.

FAGAN,L.M. (1980).
V.M: Representing Time-dependent Relations in a Medical Setting.
PhD Thesis.Stanford: Stanford University, Dept. Computer Science.

FAGAN,L.M., SHORTLIFFE,E.H. and BUCHANAN,G.G. (1980).
Computer-based Medical Decision Making: from MYCIN to VM.
Automedica, 3, 97-106.

FEIGNEBAUM,E.A. (1982).
Knowledge Engineering for the 1980s. Dept. Computer Science, Stanford University, USA.

FINCHAM,W. and BEISHON,J. (1973).
The Human Respiratory System (T241 13/14). Walton Hall: OUP.

FINKELSTEIN,L. (1982).
The Theory and Philosophy of Measurement. In Handbook of Measurement Science : Volume 1. Ed. P.H. Sydenham. Chichester: Wiley

FINKELSTEIN,L. and CARSON,E.R. (1986).
Intelligent Measurement in Clinical Medicine. Proc. of the 5th IMEKO Symposium on Measurement Theory,Jena. International Meas. Confed.: Budapest.

FINKELSTEIN,L. and FINKELSTEIN,A.C.W. (1983).
Review of Design Methodology. IEE Proc., 130 Pt A (4), 213-221.

FIRST,M.B., SOFFER,L.J. and MILLER,R.A. (1985).
QUICK (Quick Index to Caduceus Knowledge): Using the Internist-I/Caduceus. Knowledge Base as an Electronic Text-book of Medicine. Comput.Biomed.Res., 18, 137-165.

FLAGG,P.J. (1928).
Treatment of Asphyxia in the New-born. JAMA, 91, 788-791.

FORGY,C.L. (1981).
OPS5 User's Manual. Carnegie-Mellon University.

GAMMACK,J.G. and YOUNG,R.M. (1985).
Psychological Techniques for Eliciting Expert Knowledge.
In: Research and Development in Expert Systems. Ed. M.A.Bramer
Cambridge : Cambridge University Press.

GLEASON,G.J. and AGIN,G.J. (1979).
A Modular System for Sensor-Controlled Manipulation and Inspection.
Proc. IX Int. Symp. of Industrial Robots. Washington D.C. 57-70.

GORRY,G.A., KASSIRER,J.P., ESSIG,A. and SCHWARTZ,W.B. (1973).
Decision Analysis as the Basis for Computer-Aided Management of Acute Renal Failure. Amer.J. Med., 55, 473-484.

GORRY,G.A., SILVERMAN,H. and PAUKER,S.G. (1978).
Capturing Clinical Expertise: A Computer Programme that considers Clinical Responses to Digitalis. Amer.J. Med., 64, 452-460.

GREGORY,G.A. (1983).
Who Should Receive Intensive Care? (Presidential Address).
Crit. Care Med., 1 (10), 767-8.

GROVER,M.D. (1983).
A Pragmatic Knowledge Acquisition Methodology. In: Proc. 8th IJCAI.
Ed. A.Bundy. Los Altos: William Kaufmann 436-438.

HARRISON,T.J. (1986).
Charles and the Computer. Measurement and Control, 19 (3), 84-91.

HARVEY,A.M. (1974)
Neurosurgical Genius - Walter Edward Dandy, John Hopkins Med. J. 135, 358-368.

HAYES-ROTH,F., WATERMAN,D.A. and LENAT,D.B. (1983).
Building Expert Systems. Reading, Mass.: Addison-Wesley.

HERNANDEZ-SANDE,C., MORET-BONILLO,V. and ALONSO-BETANZOS,A. (1989).
ESTER: An Expert System for Management of Respiratory Weaning Therapy.
IEEE Trans. Biomed. Eng., BME-36 (5), 559-564.

HILBERMAN,M. (1975).
The Evolution of Intensive Care Units.
Crit.Care. Med., 3 (4), 159-165.

HODGES,A. (1983).
Alan Turing: The Enigma of Intelligence. London: Counterpoint.

HOFMANN, D. (1982).
Measurement Errors, Probability, and Information Theory
In: Handbook of Measurement Science Vol.1 Ed. P.H. Sydenham.
Chichester : Wiley-Interscience.

HOLMDAHL,M.H. (1962).
The Respiratory Care Unit. Anesthesiology,23 (4), 559-568.

HORROCKS,J.C., McCANN,A.P., STANILAND,J.R., LEAPER,D.J., and de DOMBAL,F.T. (1972).
Computer-aided Diagnosis: Description of an Adaptable System, and Operational Experience with 2,034 cases. British Medical Journal, 2, 5-9.

HUNTER,J.R.W. (1986).
Artifical Intelligence in Medicine. A Tutorial Survey. Part 2: Comparisons. Biomedical Measurement, Informatics and Control, 1 (3).

IBSEN,B. (1954).
The Anaesthetists Viewpoint on Treatment of Respiratory Complications in Poliomyelitis During the Epidemic in Copenhagen,1952.
Proc.Roy.Soc.Med., 47, 52.

JENSEN,R.E.,SHUBIN,H., MEAGHER,P.F. and WEIL,M.H. (1966).
On-line Monitoring of the Seriously Ill Patient.
Med.Biol.Engng., 4, 265-272.

KARI,A. (1988).
Benefits and Disadvantages of Automated Data Management Systems in Intensive Care. In: Proc. Medical Informatics 88: International Conference on Computers in Clinical Medicine. London: British Medical Informatics Society.

KATONA,P.G. (1983).
Automated Control of Physiological Variables and Clinical Therapy.
C.R.C. Critical Reviews in Biomedical Engineering, 8 (4), 281-310.

KAWAKAMI,Y., YOSHIKAWA,T. and ASANUMA,Y. (1981).
A Control System for Arterial Blood Gases.
J.Appl. Physiol., 50, 1362.

KIRSCHNER,M. (1930).
Zum Neubau der Chirurgischen. Universitatsklinik Tuebingen.
Der Chirurg, 2, 54-61.

KLAUS,M.H. and KENNELL,J.H. (1970).
Mothers Separated from their Newborn Infants.
Pediat.Clin.N.Amer., 17, 1015-1037.

KNAUSS,W.A., DRAPER, C.A., WAGNER,D.P. and ZIMMERMAN,J.E. (1985).
APACHE-II: A Severity of Disease Classification System.
Crit.Care Med., 13 (10), 818.

KOLATA,G. (1982).
How Can Computers Get Common Sense? Science, 217, 1237-1238.

LASSEN,H.C.A. (1953).
Preliminary Report on the 1952 Epidemic of Poliomyelitis in Copenhagen. With Special Reference to the Treatment of Acute Respiratory Insufficiency. Lancet, 1, 37.

LEANING,M.S. (1980).
The Validity and Validation of Mathematical Models. PhD Thesis, Dept. Systems Science, City University, London.

LEDLEY,R.S. and LUSTED, L.B. (1959).
Reasoning Foundations of Medical Diagnosis.
Science, 130, 9-21.

LIGHTHILL,J. (1972).
Artificial Intelligence: A General Survey.
Report SRC-72-72. Science Research Council, UK.

LINDSAY,R.K., BUCHANAN,B.G, FEIGENBAUM,E.A. and LEDERBERG,J. (1980).
Applications of Artificial Intelligence for Organic Chemistry:
The DENDRAL Project. New York: McGraw-Hill.

MABRY,J.C., THOMPSON,H.K., HOPWOOD,M.D. and BAKER,W.R. (1977).
A Prototype Data Management and Analysis System - CLINFO : System Description and User Experience. In: Proc. MEDINFO-77. Ed. D.B.Shires and H.WOLF. Amsterdam: North-Holland, 71-75.

McCLEAVE,D.J., GILLIGAN,J.E. and WORTHLEY,L.I.G. (1977).
The Role and Function of an Australian Intensive Care Unit.
Crit. Care.Med., 5 (5), 245-251.

MEEHL,P.E. (1954).
Clinical vs Statistical Prediction.
Minnesota: University of Minnesota Press.

MENDELSON,Y. and PEURA,R.A. (1984).
Non-invasive Transcutaneous Monitoring of Arterial Blood Gases.
IEEE Trans.Biomed.Eng., BME-31 (12), 792-800.

MENN,S.J., BARNETT,G.O., SCHMECHEL,D., OWENS,W.D. and PONTOPPIDAN,H. (1973).
A Computer Program to Assist in the Care of Acute Respiratory Failure.
JAMA, 223 (3), 308-312.

MILLER,P.L. (1984).
Goal-directed Critriquing by Computer: Ventilator Management.
Comput.Biomed.Res., 18, 422-438.

MILLER,R.A., POPLE,H.E. and MYERS,J.D. (1982).
INTERNIST-I, an Experimental Computer-based Diagnostic Consultant for General Internal Medicine. New Engl. J. Med., 307, 468-476.

MINSKY,M. (1975).
A Framework for Representing Knowledge.
In: The Psychology of Computer Vision, Ed. P.Winston.
New York: McGraw-Hill, 211-277.

NEWELL,A., SHAW,J.C. and SIMON,H.A. (1957).
Empirical Explorations of the Logic Theory Machine.
Proc.West. Jt. Computer Conf., 218-239.

NEWELL,A., SHAW,J.C. and SIMON,H.A. (1960).
A Variety of Intelligent Learning in a General Problem Solver.
In: Self Organizing Systems. Ed. M.C.Yovits and S.Cameron.
Elmsferd, New York: Pergamon, 153-189.

NEWELL,A. and SIMON,H. (1972).
Human Problem-Solving. Englewood Cliffs : Prentice Hall.

NIGHTINGALE,F. (1863).
Notes on Hospitals
Longman,Green,Longman,Roberts and Green (3rd Ed).

OHLSON,K.B., WESTENSKOW,D.R. and JORDAN,W.S. (1982).
A Microprocessor-based Feedback Controller for Mechanical Ventilation.
Ann.Biomed.Eng., 10 (1), 35-48.

PATTON,M.Q. (1982).
Practical Evaluation.
Beverley Hills: Sage.

PAUKER,S.G., GORRY,G.A., KASSIRER,J.P. and SCHWARTZ,W.B. (1976).
Towards the Simulation of Clinical Cognition: Taking a Present Illness by Computer. Am. J. Med., 60, 981-996.

PERLMAN,F., McCUE,J.D. and FRIEDLAND,G. (1974).
Urinary Tract Infectia (UTI)/ Vaginitis Protocol, Introduction.
Ambulatory Care Project, Lincoln Laboratory, M.I.T. and Beth Israel Hospital, Harvard Medical School.

POPLE,H.E. (1982)
Heuristic Methods for Imposing Structure on Ill-structured Problems: The Structuring of Medical Diagnostics.
In: Artifical Intelligence in Medicine, Ed. P.Szolovitz.
Boulder, Co: Westview Press, 119-190.

PRICE,D.J. and MASON,J. (1986).
Resolving the Numerical Chaos at the Bedside.
In: Proc. Current Perspectives in Health Computing.
Edited by J.Bryant, J.Roberts and P.Windsor.
Weybridge: BJHC.

PRYOR,T.A., GARDNER,R.M., CLAYTON,P.D. and WARNER,H.R. (1983).
The HELP System. J.Med. Sys., 7, 87-101.

QUINLAN,J.R. (1979).
Rules by Induction from Large Collections of Examples.
In : Expert systems in the Micro-Electronic Age,
Ed.D.Michie. Edinburgh : Edinburgh University Press.

REGGIA,J.A. NAU,D.S. and WANG,P.Y. (1983).
Diagnostic Expert Systems Based on a Set Covering Model.
Int. J. Man-Machine Studies, 19, 437-460.

ROBERTS,L. (1965).
Machine Perception of Three-dimensional Solids.
In: Optical and Electro-optical Information Processing, Ed. J.Tippett.
Cambridge,USA: MIT Press, 159-197.

ROSEN,R. (1985).
Anticipatory Systems : Philosophical, Mathematical and Methodological Foundations. Oxford : Pergamon.

ROSS,D.T. (1985).
Applications and Extensions of SADT.
Computer, 18 (4), 25-34.

ROSS,D.T. (1977).
Structured Analysis (SA): A Language for Communicating Ideas.
IEEE Trans. Soft.Eng., SE-3 (1), 16-34.

ROSS,D.T. and SCHOMAN,K.E. (1977).
Structured Analysis for Requirements Definition.
IEEE Trans. Soft.Eng., SE-3 (1), 6-15.

ROMAN,G-C, (1985).
A Taxonomy of Current Issues in Requirements Engineering.
Computer, 18 (4), 14-22.

ROUSSEL,P. (1975).
PROLOG: Manual de reference et d'utilisation.
Groupe d'Intelligence Artificielle, Marseille-Luminy, September, 1975.

RUDOWSKI,R., FROSTILL,C. and GILL,H. (1988).
A Knowledge-based Support System for Mechanical Ventilation of the Lungs. The KUSIVAR Concept and Prototype.
Comput.Meth. Prog. Biomed., 30, 59-70.

SAFAR,P., DEKORNFELD,T.J., PEARSON,J.W. and REDDING,J.S. (1961).
The Intensive Care Unit. A Three Year Experience at Baltimore City Hospitals. Anaesthesia, 16 (3), 275-284.

SAMUEL,A.L. (1963).
Some Studies in Machine Learning Using the Game of Checkers.
In: Computers and Thought Ed. E.A.Feigenhaum and J.Feldman.
New York: McGraw Hill.

SANDELL,H.S.H. (1984).
GENIE User's Guide and Reference Manual.
Technical Report No. 84-003, Electrical and Biomedical Engineering, Vanderbilt University, Nashville, Tennessee.

SAVAGE,L.J. (1954).
The Foundations of Statistics. New York: Wiley.

SCHEFFER,P.A., STONE,A.H. and RZEPKA,W.E. (1985).
A Case Study of SREM. Computer, 18 (4), 47-54.

SCHWEICKERT,R., BURTON,A.M., TAYLOR,N.K., CORLETT,E.N., SHADBOLT,N.R. and HEDGECOCK,A.P. (1987).
Comparing Knowledge Elicitation Techniques : A Case Study.
Artificial Intelligence Review, 1, 245-253.

SHADBOLT,N. and BURTON,M. (1989).
Knowledge Elicitation.
In : Evaluation of Human Work : Practical Ergonomics Methodology,
Ed. J.Wilson and N.Corlett. Taylor and France.

SHANNON,C.E. (1950).
A Chess-playing Machine. Sci.Amer., 182 (2), 48-51.

SHAW,M.L.G. and GAINES,B.R. (1983).
A Computer Aid to Knowledge Engineering.
In : Proc. 3rd BCS Conference on Expert Systems.
Ed. J.Fox. Cambridge : Cambridge University Press.

SHAW,M.L.G. and GAINES,B.R. (1987).
KITTEN : Knowledge Initiation and Transfer Tools for Experts and Novices. Int.J. Man-Machine Studies, 27, 251-280.

SHEPPARD,L.C. (1980).
Computer Control of the Infusion of Vaso-active Drugs.
Ann. Biomed.Eng. 8, 431.

SHOEMAKER,W.C., APPEL,P.L., BLAND,R., HOPKINS,J.A. and CHANG,P. (1982). Clinical Trial of an Algorithm for Outcome Prediction in Acute Circulatory Failure. Crit.Care Med., 10 (6), 390-397.

SHORTLIFFE,E.H., DAVIS.R., BUCHANAN,B., AXLINE,S., GREEN,C. and COHEN,S. (1975). Computer-based Consultations in Clinical Therapeutics - Explanation and Rule Acquisition Capabilities of the MYCIN System. Comput.Biomed.Res., 8, 303-320.

SHORTLIFFE,E.H. (1976).
Computer-based Medical Consultations: MYCIN.
New York: Elsevier.

SIEGAL,J.H. (1981).
Relations Between Circulatory and Metabolic Changes in Sepsis.
Ann.Rev. Med. 32, 175-194.

SIEVERT,G.E. and MIZELL,T.A. (1985).
Specification-based Software Engineering with TAGS.
Computer, 18 (4), 56-65.

SITTIG,D.F. (1988).
COMPAS: A Computerised Patient Advice System to Direct Ventilatory Care. Ph.D. Thesis. Dept. Medical Informatics, University of Utah.

SLAGLE,J.R. (1961).
A Heuristic Program that Solves Symbolic Integration in Freshman Calculus: Symbolic Automatic Integrator (SAINT).
Report 5G-001, Lincoln Laboratory, MIT, Cambridge, USA.

SMITH,D.M., MERCER,R.R. and ELDRIDGE,F.L. (1978).
Servo Control of End-tidal CO_2 in Paralysed Animals.
J.Appl. Physiol., 45, 133.

SMITH,R.G. and BAKER,J.D. (1983).
The Dipmeter Adviser System. A Case Study in Commercial Expert System Development.
In: Proc. 8th IJCAI Ed.A.Bundy Los Altos : William Kaufmann, 122-129.

SNYDER,J.V., McGUIRK,M., GRENVIK,A. and STICKLER,D. (1981).
Outcome of Intensive Care: An Application of a Predictive Model.
Crit.Care Med., 9 (8), 598-603.

SWANSON,G.D., CARPENTER,T.M., SNIDER,D.E. and BELLVILLE,J.W. (1971).
An On-line Hybrid Computing System for Dynamic Respiratory Response Studies. Comp.Biomed.Res., 4, 205-215.

TAYLOR,J.H. and FREDERICK,D.K. (1984).
An Expert System Architecture for Computer-aided Control Engineering.
IEEE Proceedings, 72 (12), 1795-1805.

TURING,A.M. (1937).
On Computable Numbers, with an Application to the Entscheidungs Problem. Proc.Lond. Math. Soc., 2, 230-265.

TURING,A.M. (1950).
Computing Machinery and Intelligence. Mind, 59, 433-460.

U.S. ARMY MEDICAL SERVICES. (1955).
Surgery in World War II. Vol.II General Surgery.
Office of the Surgeon General, Department of the Army, Washington,D.C.

U.S. ARMY MEDICAL SERVICES. (1963).
Surgery in World War II. Vol.I Thoracic Surgery.
Office of the Surgeon General, Department of the Army, Washington,D.C.

U.S. ARMY MEDICAL SERVICES. (1964).
Surgery in World War II, Activities of Surgical Consultants Vol.II.
Office of the Surgeon General, Department of the Army, Washington,D.C.

WALDROP,M.M. (1984).
The Necessity of Knowledge. Science, 223, 1279-1282.

WEISS,S., KULIKOWSKI,C.A. and SAFIR,A. (1978a).
Glaucoma Consultation by Computer. Comput.Biol.Med., 8, 25-40.

WEISS,S., KULIKOWSKI,C.A., AMAREL,S. and SAFIR,A. (1978b).
A model-based method for computer aided medical decision making.
Artif. Intel., 11, 145-172.

WEIZENBAUM,J. (1966).
ELIZA - A Computer Program for the Study of Natural Language Communication between Man and Machine. Comms. of the ACM,9, 36-45.

WEST,J.B. (1979).
Respiratory Physiology - The essentials.
Baltimore: The Williams and Wilkins Co.

WEYL,S., FRIES,J., WIEDERHOLD,G. and GERMAND,F. (1975).
A Modular Self-describing Clinical Databank System.
Comput. Biomed. Res., 8, 279-293.

WINOGRAD,T. (1972).
Understanding Natural Language.
New York: Academic Press.

APPENDIX I

CRITICAL REVIEW OF ARTIFICIAL INTELLIGENCE IN MEDICINE

I.1 Introduction

Biomedical engineering is a fertile domain for the application of new computer-based techniques that have the potential to enhance patient care. These techniques come in many forms, and can be classified according to the level of processing required before their stated purpose is achieved. Low-level processing include techniques such as signal interpretation, where a threshold value has to be reached before an appropriate action is taken. This type of system uses a shallow data model to transform data into a more useful information-based format. However in this Appendix, the systems of interest are those which provide a further transformation, that of information into knowledge via appropriate explanation and justification of concepts used. These knowledge-based systems have various synonyms including expert systems, intelligent systems and decision-support systems.

I.2 Artificial Intelligence: General Medical Systems

The criterion by which the medical systems reviewed have been chosen is that they each illustrate some novel use of techniques found in the domain of artificial intelligence. They have thus become leaders in their respective fields from which other intelligent systems have evolved.

The Dendral system was one of the first medical expert systems to be implemented, and was the antithesis of the general problem solving strategies of its day. It worked in a narrow and very well-defined domain which paved the way for a plethora of similar systems. These systems were not confined to the medical domain, for example, geological expert systems which were of benefit to the oil industry were constructed.

The MYCIN system is included because it is a prime example of the use of a production rule methodology, this describes how the knowledge required by the system is captured. A normal representation of a production rule, (IF....THEN....), is supplemented by a quantitative qualifier, as follows:-

IF premise
THEN action
WITH a confidence factor X.

This allows a clinicians degree of confidence in a decision to be captured by the system, and at the same time aid in its transparency to the end-user.

CASNET is a system which uses a causal-associational model to represent the embodied knowledge. This knowledge is separated into three different entities : manifestations present in the patient; pathophysiological processes caused by the manifestations; and disease states caused by the pathophysiological processes. If this is thought as a three-plane hierarchy then the causal-associational links can be both inter- and intra-planal. A fourth 'therapy' plane also links to this hierarchy at the level of the disease state plane.

The PIP system is included because it was one of the first intelligent systems to include a frame-type representation. This allows the knowledge engineer to keep all similar knowledge together, rather than have it dispersed throughout the program. Frames are therefore useful in the development stage of the system, as the knowledge embodied in the program remains transparent to the user and is readily obtainable for editing purposes.

HEARSAY-II, although not strictly a medical system, is the product of a large research programme which started in the early 1970s and is still on-going, (HEARSAY-III is now under development). A feature of this system is its blackboard architecture for control of the knowledge processes intermediate to final output. Like DENDRAL it is used for signal understanding rather than as a diagnostic system.

The INTERNIST research programme was an ambitious attempt to combine all knowledge relevant to general internal medicine in one system. It is included in this review to illustrate the problems involved when techniques employed for a narrow clinical domain are exploited for a much wider domain.

I.3 DENDRAL

DENDRAL is a system which interprets the data emergent from the process of mass spectography, and works at a high intellectual level. The domain is therefore vey narrow but presumes expert knowledge in what is a difficult task domain (Lindsay et al., 1980).

At the time of its first implementation DENDRAL was a precursor to intelligent knowledge-based systems and was responsible for the founding of many of the techniques used in knowledge engineering. These include both knowledge elicitation and knowledge

representation. The former was identified to be a 'bottleneck' in the design and building of intelligent systems, as although development time could be continual it needed the prior knowledge of the expertise to be embodied. This expertise could only be elicited in discrete stages due to the time demands made on the human experts involved.

The evaluation of DENDRAL involved the development of a set of analytical rules utilising a training-set of similar compounds, (for example, cyclic ketones). Once these rules were obtained they were applied to another five members of the same chemical group for the purpose of rule refinement. The refined system was then applied to the remaining compounds of the same group. The full implementation of DENDRAL is used in both academia and industry, making it not only the first 'expert system', but also one of the very few in actual clinical use.

I.4 MYCIN

MYCIN is perhaps the most cited expert system of the 1970s, and was developed by Shortliffe and others at Stanford University, U.S.A., (Shortliffe et al., 1975; Shortliffe, 1976). The domain of interest is aiding in the identification process and recognising the significance of organisms causing microbial infection, then recommending an optimal treatment protocol. This domain is well chosen as there is a need for such consultative advice, as microbial infection is often secondary to the major complaint of the patient, and the clinician responsible for the welfare of such patients may not be an expert on infectious diseases. The treatment protocols yield information not only on what drugs to recommend to combat particular infections, but also on what dose should be administered, thereby cutting down on inappropriate prescribing and antibiotic misuse.

MYCIN is an example of an expert system which uses 'production rules' to represent the causal relationships between individual items of factual knowledge held on the knowledge base. Such rule-based deduction systems are procedural by nature of the way they are constructed, that is, **IF** a premise condition is true **THEN** one can deduce that the consequent action(s) is true. An example of a typical MYCIN production rule is found in Figure A-I.1, and can be seen to have the general form:-

IF premise assertions are true
THEN consequent assertions are true
with confidence weight X.

RULE 160

If:
1) The timeframe of the patient's headache is acute.
2) The onset of the patient's headache is abrupt, and
3) The headache severity (using a scale of 0 to 4; maximum is 4) is greater than 3.

Then:
1) There is suggestive evidence (.6) that the patient's meningitis is bacterial.
2) There is weakly suggestive evidence (.4) that the patients's meningitis is viral and
3) There is suggestive evidence (.6) that the patient has blood within the subarachnoid space.

Thus this rule has three conclusions. It is represented internally in LISP as follows:

```
PREMISE   (SAND SAME CNTXT HEADACHE-CHRONICITY ACUTE)
          (SAND CNTXT  HEADACHE-ONSET ABRUPT)
          (GREATERP (VALI ONTXT HEADACHE-SEVERITY 3))

ACTION;   (DO-ALL (CONCLUDE ONTXT MENINGITUS
          BACTERIAL-MENINGITUS)
                    TALLY 600)
          (CONCLUDE CNTXT MENINGITUS VIRAL-MENINGITUS
                    TALLY 400)
          (CONCLUDE CNTXT SUBARACHNOID-HEMORRAGE
          YES
                    TALLY 600)
```

FIGURE A-I.1 A TYPICAL PRODUCTION RULE FOUND IN THE MYCIN SYSTEM

Notice that each production rule has associated with it a measure of certainty. This aids in the reasoning strategy of the system, where a value of -1 represents complete disbelief, and a value of +1 represents complete belief in the consequent assertion(s). The assertions can be Boolean combinations of clauses each of which consists of a predicate statement triple:-

(attribute,object,value).

For example,

(Gramstain,E.Coli,Gramneg),

which when translated means that the Gramstain of the E.Coli organism is Gram-negative.

The uniformity of representation for both domain-specific inferences and reasoning goals makes it possible for MYCIN to use a very general and simple control strategy, that is, a goal-directed backward-chaining of the production rules. This approach can be described in the following way. The first rule to be evaluated is the one which contains the highest level goal, which for MYCIN is "To determine if there are any organisms, or classes of organisms, that require therapy". To deduce the need for therapy requires knowledge of the infections, which is usually unknown in the first instance. Therefore the system tries to satisfy sub-goals which originate in the premise of the top-goal that will allow the infections to be inferred. Rule chaining is the name given to the process by which the production rule hierarchy is linked together; the premise portion of each sub-goal rule fires a new set of sub-goals. This process is repeated until the most fundamental level of the hierarchy is reached, where the rules become assertions that can only be confirmed or denied by directly questioning the user for the appropriate information.

After MYCIN determines the significant infections, found by assessing the overall certainty factor which combines the individual degrees of confidence associated with each production rule, the organisms which account for the infections are found deterministically. Then, if appropriate, the system proceeds to recommend an antimicrobial regimen. To reach its decision the MYCIN therapy selector uses a description of the infection(s) present; the causal organisms together with a ranking of drugs by their sensitivity; and a set of drug-preference categories. The algorithm used within the therapy selector also calculates the drug-dose required, and contains knowledge to modify the value if, for example,

the patient is in renal failure. An advantage of this therapy selection system is that it can accept and critique a treatment protocol proposed by the user. For this function the therapy selector has an appropriate facility to generate explanation and justification for its choice of action. "WHY?" queries are dealt with by displaying the rule it is trying to imply, and if "WHY?" is asked again the query is answered by ascending the goal-tree hierarchy. "HOW?" queries are interpreted as the chain of rules which are fired to get to that particular conclusion, and if "HOW?" is asked again the query is answered by descending the goal tree hierarchy.

An advantage of using a production rule system is that each rule is a small 'packet' of knowledge, each one being independent of all the others. This has two consequences: first, changing or adding knowledge to the MYCIN knowledge base is relatively easy; and second, addition of new rules is facilitated by a having modular data structure. The MYCIN system is also user-friendly, characterised by its natural language interface which translates the knowledge encoded in the system to a form that is easy to understand and examine at the user interface.

The MYCIN system can deal with both inexact and incomplete information. Inexact information is inherent in this domain as many test results are qualitative in nature, and relatively few statements can be made with absolute certainty. Incomplete information may arise from the time constraints involved in the identification of an organism, that is, the time taken for an identifying laboratory test to be completed and the result of it made known to the clinician.

A disadvantage of the system is that disease states can not always be adequately described by a rule. Also, it may not always be possible to map a series of desired actions into a set of production rules. Another disadvantage is that although new knowledge can be added by inserting a new rule, this may not interact with the existing rules in the anticipated way.

The MYCIN system is encoded in LISP and runs under the TENEX operating system. When compiled the system takes up approximately 50 kbytes of disk space, which includes 16 kbytes to hold the knowledge base and 28 kbytes to hold clinical parameters, tables and working space. A normal consultation takes on average 20 minutes, which includes time allowed for the optional use of the explanation facility. MYCIN has been evaluated as having a diagnostic success rate

of 72%. This relatively poor performance (for an expert system) combined with clinician suspicion and resistance to new technology, contributed to the fact that the system never went into clinical operation. However, as a research tool it was the subject of extensive interest in the artificial intelligence fraternity, being responsible for a number of successful descendants. One of these, the PUFF system developed to interpret pulmonary function test results (Aikens et al., 1983), is one of the few medical expert systems in daily clinical use.

I.5 CASNET

CASNET is an expert system for consultation in the diagnosis and therapy of glaucoma (Weiss et al., 1978 a,b). A feature of this system is that the medical knowledge used in the patient-specific reasoning process is encompassed in a causal-associational network model of the specific disease process. Such a network, termed a 'semantic net', allows the structure of the medical knowledge covered by the system to be more coherent. A semantic network comprises nodes connected by links, where nodes correspond to either the condition or action part of the rules and the links are the inferences between the two. The model of disease is separate from the decision making strategy which allows the up-dating of both data structures to be facilitated more easily.

The CASNET model has a descriptive component which consists of four sets of elements, as follows:-

i) Observations - these consist of symptoms and laboratory test results, etc., and form the direct evidence that a disease is present

ii) Pathophysiological states - these describe internal abnormal conditions or mechanisms that can directly cause the observed findings. The causal relations between states are of the form

$$n_i \xrightarrow{a_{ij}} n_j$$

where n_i and n_j are states and a_{ij} is the causal frequency with which state n_i, when present in a patient, leads to state n_j

iii) Disease categories - each category consists of a pattern of states and observations, and is therefore conceptually at the highest level of abstraction

iv) Treatment plans and therapies - composed of sets of related treatments or treatment plans.

Figure A-I.2 shows a three level description of a disease process, the

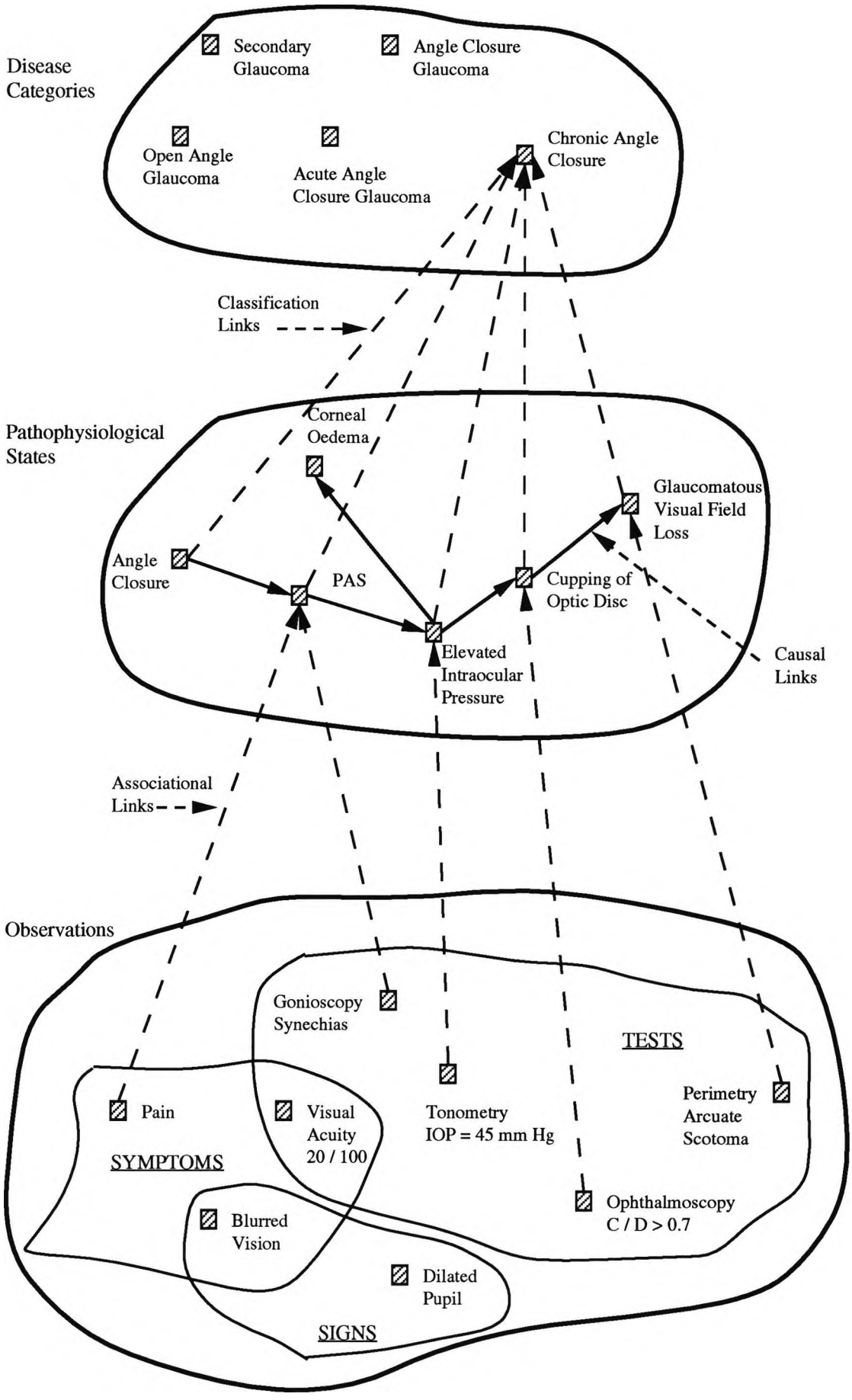


FIGURE A-I.2 THREE-LEVEL DESCRIPTION OF DISEASE PROCESS
(Kulikowski & Weiss, 1982)

fourth component (the treatment plans) are associated with the disease-state plane.

Other components of the CASNET model are the decision rules, which state:-

i) The degree of confidence with which an inference of a pathological state can be made from an observed pattern of findings. In rule-form this translates to :-

$$t_i \xrightarrow{Q_{ij}} n_j$$

where t_i is a finding (or observation) or Boolean combination of findings, n_j is a state, and Q_{ij} is a number in the range -1 to +1 representing the confidence with which t_i is believed to be associated with n_j. The value of Q is then transposed to a certainty factor which indicates how certain is the belief that the patient is in state n_j. A threshold function is then used to determine whether or not the certainty factor confirms, denies, or leaves undetermined a particular state.

ii) Rules for associating disease categories and pathophysiological states to treatment protocols are in the form of a classification table which consists of ordered triples,

$$(n_1,D_1,T_1),\ (n_2,D_2,T_2),.....(n_i,D_i,T_i),$$

where n_i is the pathophysiological state, D_i is the disease process arising from it, and T_i are the preferred treatment regimes for disease D_i.

The pathogenesis and mechanisms of a disease process are described in terms of cause-and-effect relationships between pathophysiological states. For example, Figure A-I.3 shows a partial causal network for glaucoma, where each box represents a node, n_i, and is a pathophysiological state, and each arrow represents the causal associations between states. In this way complete or partial disease processes can be characterised by pathways through the network. When a set of cause-and-effect relationships are specified the resulting network can be described as an acyclic graph of states (Weiss et al., 1978b). This state network is defined by a four-tuple, (S,F,N,X), where **S** is the set of starting states (that is, those states which have no antecedant causes); **F** is the set of final states; **N** are the number of states visited between **S** and **F**; and **X** are the causal relationships between the states visited (in the form of a list).

An unusual feature of CASNET is the language with which it was implemented. The authors of the system considered that an efficient program would lead to decreased computer response times and

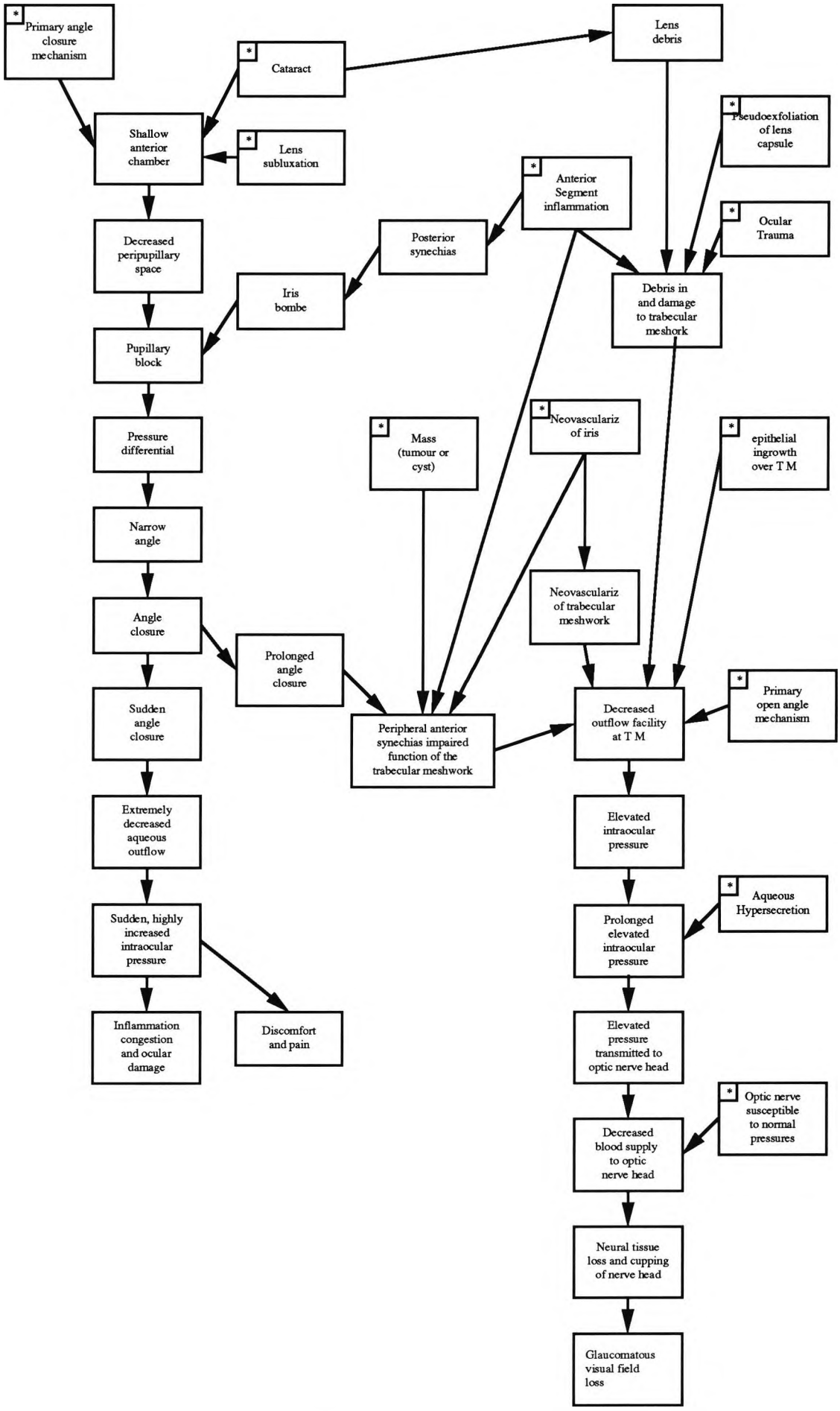


FIGURE A-1.3 PARTIAL CAUSAL NETWORK FOR GLAUCOMA. STATES WITH NO ANTECEDENT CAUSES ARE MARKED BY ASTERISKS (*) (Kulikowski and Weiss, 1982)

therefore an increase in user acceptance, therefore FORTRAN was chosen for its implementation. This choice had repercussions in the development of CASNET, as any modification to the program meant that the source code had to be obtained, changed and recompiled. In the 1978 implementation of CASNET, the knowledge base consisted of more than 100 states, 75 classification tables and 200 diagnostic and treatment statements. Thus, a most comprehensive knowledge representation had been achieved. However glaucoma is an example of a well-defined clinical problem, (that is, not many disease processes can overlap with it, which would make differential diagnosis more difficult). If a more complex domain of interest were chosen one would expect a great increase in the number of states, classification tables and treatment statements. This may make the causal-associational model unworkable for large-scale systems, due to the increase in complexity.

An important test for any expert system is how it copes with contradictory information. This can occur in the CASNET system, for example, when a particular state is confirmed even though all of its potential causes in the network are denied. When such a situation occurs the CASNET system advises the user that this has happened. There are two explanations for the origin of this contradictory information : it could be that the model of the disease process may be incomplete (that is, one or more nodes together with their respective causal relationships are missing from the knowledge base); or it may be that the threshold function associated with one of the existing causal links has been set at an inappropriately high level. The modular structure of the knowledge base enables easy access to the underlying reason for the contradiction, so up-dating of the knowledge base can be facilitated.

An advantage of the CASNET system is that it can present alternative expertise derived from different consultants. It is this fact, coupled with the fact that glaucoma is a well-defined clinical problem which results in an accuracy of diagnosis which contributed to its relative success. The diagnostic accuracy of CASNET has been evaluated at greater than 75 % for particularly difficult cases, and greater than 90 % for cases which constitute a broad clinical spectrum. Unfortunately the clinical utility of the system was not assessed as high, which resulted in CASNET never being used in a routine clinical environment. However, the system remains as a very useful tool for research into artificial intelligence in medicine.

I.6 PIP

The task of the Present Illness of a Patient (PIP) expert system is to diagnose oedematous patients (Pauker et al., 1976). It was developed more for understanding the cognitive processes involved in medical decision-making rather than for use as a clinical tool. Accordingly, a less well-defined clinical task was chosen, as oedema represents a complex diagnostic domain.

Conceptually, there are four components to the program (see Figure A-I.4), which are: the patient-specific data; the supervisory program module; the 'short term' memory; and the 'long term' memory. Clinical data about a specific patient is entered and passes to the supervisory program, which in turn delivers it to the 'short term' memory. The supervisory program then generates hypotheses about the given set of facts, using information stored in both the 'short term' and 'long term' memories, transferring all relevant information to the 'short term' memory. Additional patient-specific questions are then generated by the supervisory program, and the data entered starts the control cycle once more. Thus, the program alternates between asking questions and integrating new information into a developing picture of patient state. A typical cycle consists of characterisation of the observations, seeking advice on how to proceed, generating hypotheses, testing those hypotheses and selecting new questions to ask the user. It is thought that this test and hypothesis cycle more closely mimics the way in which clinicians make decisions than the more rigid production rule type inference mechanisms. Whereas the procedural production rule inference methodology is deductive by nature, the test and hypothesis inference cycle is both declarative and abductive.

A feature of the PIP system is that the long-term memory uses a 'frame representation' for its collection of facts about a particular disease, clinical state or physiological state. Using frames to represent knowledge was first proposed in the mid-1970s (Minsky, 1975), and since then they have become a popular medium used by knowledge engineers to conceptualise various types of expert system. Each frame has a number of 'slots' for the inclusion of specific sub-categories, such as observations, non-causal information and rules for judging how closely a given patient might match the disease state that the frame describes. An advantage of using a frame-type representation is that all relevant information about a particular disease sub-class is held close together and not spread over the entire knowledge base as is true of production rule systems.

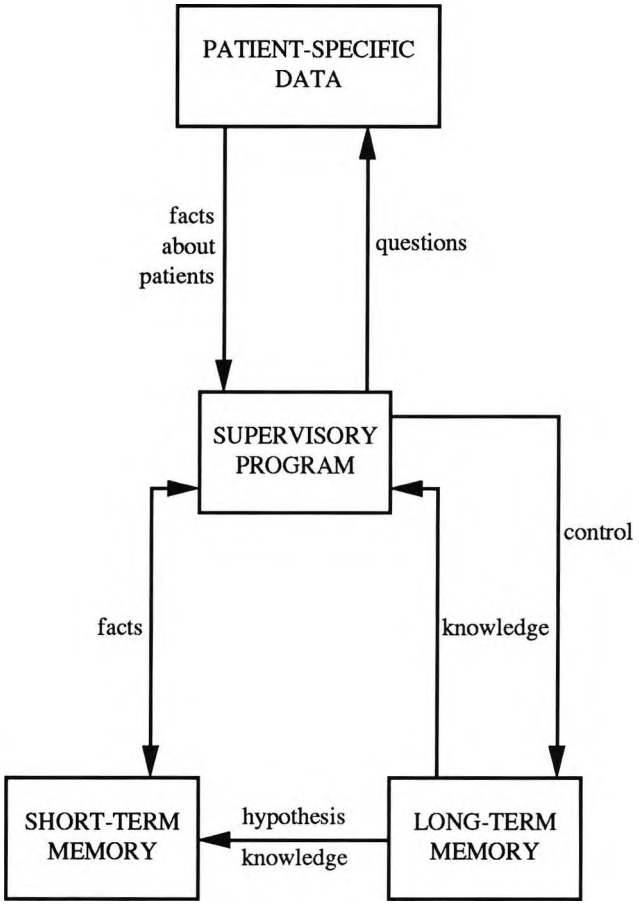


FIGURE A-I.4 PROGRAM ORGANISATION OF PIP
(After Pauker et al., 1976)

PIP uses a scoring system for each hypothesis generated, and then classifies the score with the appropriate use of threshold functions. The score originates from the uncertainty rules found within each of the frames in the long-term memory. It has two components: the first is a measure of 'goodness-of-fit' of observed to expected manifestations for the hypothesis under examination; the second is a value which corresponds to the 'cover' of the frame, that is, the ratio between the number of findings explained by the hypothesis to the total number of findings found in the frame. Once this score, termed the 'belief function', has been established the focus of the system then turns to other frames where the test and hypothesis cycle continues. This process continues until all of the reported manifestations have been covered.

One of the problems associated with this type of system is the maintenance of a sufficiently focused and clinically acceptable line of reasoning. Deviation away from the line of reasoning can be accounted for from the use of a generalised scoring system. To keep this problem to a minimum, it was suggested by the authors that only a well-defined and narrow clinical domain should be chosen for this type of methodology.

The PIP system developed a knowledge base of about 70 frames, which included frames for 20 different diseases, the rest being representations of various clinical and physiological states which are associated with those diseases.

I.7 HEARSAY-II

The HEARSAY-II speech understanding system is the product of a well-financed research programme from Carnegie-Mellon University, U.S.A. (Erman et al., 1980). A feature of this system is its general problem solving framework, termed the 'blackboard architecture', which has become the basic control model for real-time intelligent signal understanding systems (Hayes-Roth, 1983). The blackboard architecture contains four elements:-

i) 'entries', which is the name given to intermediate by-products of the problem-solving strategy.

ii) 'knowledge sources', these are diverse and independent sub-programs which are capable of solving a specific problem efficiently. They can be thought of as a condition-action duplex, where the condition premise describes the situations to which each knowledge source can contribute, and the action component specifies the interaction of that particular knowledge source in the

overall problem-solving strategy. It is the activated knowledge sources which produce the entries.

iii) the 'blackboard' itself, which can be considered as a structured database. It serves two roles : first, it represents intermediate states of problem solving activity; and second, as all communication between the individual knowledge sources is carried out via the blackboard, it can accommodate hypotheses from one particular knowledge source that may activate a number of others.

iv) an 'intelligent control mechanism', which has sufficient knowledge embodied within it to decide if and when any particular knowledge source should be activated, thus generating entries which are recorded on the blackboard.

The blackboard control structure supports a hierarchical arrangement of knowledge sources. For instance, in the HEARSAY-II system the top-level of the hierarchy is the spoken sentence, followed by phrase, word sequence, word, syllable, segment and phoneme respectively. The control postulate is that of 'establish-and-refine', that is, each activated knowledge source in the hierarchy tries to confirm or reject itself. If it is confirmed then it refines itself by calling on the next immediate layer in the hierarchy to enter the same cycle. This procedure continues until (if successful) the bottom layer of the hierarchy is reached. Once the phonemes have established themselves, the spoken sentence has been modelled completely. In this way the spoken sentence is recognised as syntactically valid, which is one of the definitions used as speech 'understanding'. Further signal processing techniques can then be applied to obtain a more comprehensive understanding, such as translation into another language or paraphrasing an argument.

HEARSAY-II is implemented in the SAIL programming environment, which is an ALGOL-60 dialect, and includes list and set membership functions. About 40 knowledge sources were developed, each one being a one or two person effort for a period of between 2 to 36 months. This work produced between 5 and 100 pages of source code (with an average of 30), and on average each knowledge source has 50 kbytes of data in its local database. The kernel program comprises 300 pages of code, and was the product of a continual development process, carried out by two full-time programmers over a period of three years.

It is interesting to investigate the development of the HEARSAY-II speech understanding system in terms of its evaluation. The first version of the kernel program took two persons 4 months to complete; the first knowledge source implementation took 16 months to

work, although using an incomplete knowledge base; the first complete implementation (kernel plus knowledge source plus interface development) took 27 months, when the system had a 10 % success rate of signal understanding with a 250 word working vocabulary; 10 months later, (that is, after 35 months of development), the initial very poor performance had been transformed into a 90 % success rate working with a 1000 word vocabulary.

A disadvantage of the architecture as it is implemented in the HEARSAY-II system, is that the blackboard has to be accessed at each decision step. This is desirable for structuring communication between different knowledge sources, but consumes a lot of time when it occurs at intermediate decision points within an individual knowledge source. Therefore a different problem solving strategy is suggested for dealing with knowledge from within a single knowledge source.

More processing time could also be saved if knowledge-based meta-programming tasks were included in the kernel system. In its present implementation time is lost in the choice process from which the system decides which knowledge source to implement. An intelligent decision-making module which 'sits' on top of the kernel system could alleviate that problem.

I.8 INTERNIST

The INTERNIST project of the University of Pittsburgh, U.S.A. is an ambitious attempt to represent medical knowledge which would enable diagnosis of general internal medicine (Miller et al., 1982). A total of 15 person-years of research effort has produced an extensive knowledge base, which in 1982 consisted of over 500 individual disease profiles and over 3550 manifestations of disease. A disease profile is defined as a list of manifestations (for example, patient history, signs, symptoms, laboratory test results, as well as demographic data and predisposing factors), which have been reported to occur in patients with a specified disease. The profile is compiled by reviewing the medical literature as well as by consultation with human 'experts'. An example of part of a disease profile for aortic dissection is shown in Figure A-I.5. The knowledge base, because of its extensive domain, is also in the form of a hierarchy of diseases. For example, acute viral hepatitis is classified as a hepatocellular infection, hepatocellular infection is classified as a sub-class of diffuse hepatic parenchymal disease, and this in turn falls into the

The leftmost number beside each manifestation is the evoking strength and the rightmost number is the frequency. The list is abridged from a comprehensive INTERNIST disease profile comtaining over 200 findings.

AGE 16 TO 25...0 1
AGE 26 TO 55...0 3
AGE GTR THAN 55...0 3
CHEST TRAUMA RECENT IIX...1 1
HEART CATHETERIZATION RECENT IIX...1 1
HYPERTENSIONS IIX...1 4
MARFANS SYNDROME FAMILY IIX...2 2
SEX FEMALE...0 2
SEX MALE...0 4
SYNSCOPE OR SYNSCOPE RECENT IIX...1 2

CHEST PAIN LATERAL EXACERBATION WITH BREATHING....1 1
CHEST PAIN SUBSTERNAL AT REST...1 3
CHEST PAIN SUBSTERNAL BURNING...1 2
CHEST PAIN SUBSTERNAL CRUSHING...1 2
CHEST PAIN SUBSTERNAL KNIFE-LIKE OR TEARING...2 3
CHEST PAIN SUBSTERNAL LASTING GTR THAN 20 MINUTE <S>...2 3
CHEST PAIN SUBSTERNAL MIGRATING TO BACK OR ABDOMEN...3 3
CHEST PAIN SUBSTERNAL RADIATING TO BACK...2 2
CHEST PAIN SUBSTERNAL SEVERE...2 3
CHEST PAIN SUBSTRENAL SEVERITY MAXIMAL AT ONSET...3 2
ARTERY <HES> CAROTID SYSTOLIC BRUIT...1 2
ARTERY <HES> FEMORAL SYSTOLIC BRUIT...1 2
COMA WITH LUCID INTERVAL... 1 1
CYANOSIS ACRAL PART <S> ONLY...1 2
FEVER...0 3
NEUROLOGIC SIGN <S> TRANSHENT...1 2
PRESSURE ARTERIAL DIASTOLIC 95 TO 125...1 3
PRESSURE ARTERIAL SYSTOLIC 90 TO 100
PRESSURE ARTERIAL SYSTOLIC GTR ARM <S> THAN LEG<S>...2 2

AORTA XRAY DILATATION ASCENDING AORTA... 2 3
AORTA XRAY DILATATION DESCENDING THORACIC...2 3
AORTA XRAY DUPLICATION OF LATERAL BORDER...3 2
AORTA XRAY HUMP ON ARCH <LATERAL VIEW>...2 3
AORTA XRAY RAPID SERIAL ENLARGEMENT...3 2
AORTOGRAPHY ABDOMINAL AORTA IRREGULARITY OF LUMEN...2 3
AORTOGRAPHY ABDOMINAL AORTA NARROWING...2 2
AORTAGRAPHY DOUBLE CONTRAST COLUMN...4 3
AORTOGRAPHY THORACIC DESCENDING AORTA NARROWING...2 2
AORTOGRAPHY THORACIC DESCEBDING IRREGULARITY OF LUMEN...2 4
AORTOGRAPHY THORACIC FAILURE OF CONTRAST TO FILL MULTIPLE BRANCH ARTERY <IES>...3 2

FIGURE A-I.5 SAMPLE INERNIST I DISEASE PROFILE FOR AORTIC DISSECTION
(Miller R A, 1984)

category of hepatic parenchymal disease which itself is a major sub-class of the hepatobiliary system. Links exist in the knowledge base between findings and disease state and there are also disease-to-disease causal associations. A novel use (sic) of the INTERNIST knowledge base is to use the list of disease profiles as an electronic text book of medicine (First et al., 1985), although the original authors themselves estimate that the knowledge base is only 75 % complete.

Associated with each manifestation in a disease profile are two critical parameters, these are its 'evoking strength' and its measure of clinical frequency. The evoking strength is a measure of how important the manifestation is for the purpose of differential diagnosis. It is measured on a nominal scale of 0 to 5, that is, from non-specific (the finding occurs too commonly to be useful for differential diagnosis) to the other extreme where the finding is pathognomic for the diagnosis. The clinical frequency is an estimate of how often any patient would exhibit a particular finding for a particular disease. It is measured on a scale of 1 to 5, which corresponds to 'very rarely' to 'always' respectively. Both of these measures are extremely subjective and are assigned by the human experts.

The reasoning strategy of INTERNIST begins in an event-driven fashion, data entered via the keyboard evokes a set of related disease hypotheses. For each hypothesis the system creates a patient-specific model which consists of four lists, as follows :-

i) observed findings consistent with the disease
ii) observed findings unexplained by the disease
iii) other findings of the disease profile not observed in the patient
iv) findings that ought to be observed if the disease is the correct diagnosis.

A scoring system ranks the competing hypotheses and in this way the most favoured hypothesis is found. A partitioning heuristic then divides the remaining hypotheses into those which compete and those whch complement the most highly ranked one. If there are more than four competitors, the system will try to reduce the number by asking questions about the findings that are in the disease profile of the most highly ranked hypothesis. When the number of competitors has been reduced to four or less, discriminant analysis is employed, which uses the evoking strength of each manifestation in the disease profiles to

obtain the optimal hypothesis. In the event of a single hypothesis with no competitors, the system will ask for data which will 'strongly confirm' it. When this process has been completed and if there are still some unexplained findings, then the next most highly ranked hypothesis is partitioned, and the cycle continues until all findings are covered. INTERNIST is therefore another example of an abductive system, using a reasoning cycle that begins with observed findings and ends when the best hypothesis is found.

A disadvantage of using an evoking strength parameter is that the more favoured hypotheses usually lead to the most common diseases being considered responsible for the set of patient-specific findings. This would contradict one of the important functionalities of expert systems, as their anticipated use is for the input of the more difficult cases, (that is, where considerable diagnostic expertise is required). Another problem encountered with the scoring regime employed by INTERNIST is that it attaches no level of importance to the in-coming data, which can lead to inappropriate task definition. For example, if a wealth of data is forthcoming about a relatively unimportant finding, the system may spend the majority of its execution time trying to make sense of all of this data rather than more important tasks. This will be frustrating to the user and may erode user confidence in the system.

The system does not always reason correctly about causality. For example, at times the system gives weight to disease hypothesis because of a presumption concerning its ability to explain data, but if a pathophysiological analysis was carried out it would show the hypothesis to be incorrect. This can lead to a lot of time being wasted as well as inaccurate decisions being made. Indeed, evaluation of INTERNIST showed that the weakness of the system was its inability to link findings to their proper causes.

Preliminary evaluation of the INTERNIST knowledge base was performed using difficult cases taken from medical journals. Of the 43 patient histories reviewed, INTERNIST correctly diagnosed 58 % (25/43) of the cases. This compares with a diagnostic accuracy of 65 % (28/43) for non-expert clinicians, and 81 % (35/43) for expert clinicians. It proved difficult to improve on this success rate, so research effort concentrated on a successor program to INTERNIST called CADUCEUS. This new system is still in its developmental stage. Other problems associated with the INTERNIST project were its limited user interface and its lack of an explanatory facility.

One of the original collaborators sums up his experience with INTERNIST in a rather paradoxical fashion, as follows:-

> ".....a diagnostic reasoning program must have access to detailed pathophysiologic knowledge in order to permit the test of hypothesised attributions; however, if the program is forced into consideration of the detailed pathophysiology, there is a danger that unifying gestalts may fail to emerge."
>
> (Pople,1982)

I.9 Summary

Six medical expert systems have been critically reviewed, they have illustrated many of the knowledge engineering concepts and techniques currently employed. One of the goals of the early knowledge-based systems was to prove that the concept of a computer-based diagnostic or signal understanding system was indeed a valid one. This type of system was thought to have a sound practical future after the success of DENDRAL. However, subsequent experience has shown that this has not been the case, as only very few systems have reached full clinical use. Possible reasons for this lack of achievement include over zealous expectation from the end-users and technical issues, such as the best way to combine uncertainties in the in-coming data.

MYCIN was given as an example of a procedural expert system, to be compared with CASNET, PIP and INTERNIST which are all examples of declarative expert systems. The distinction between the two types hinges upon the relationship between the encoded knowledge representation and the control algorithm. In systems where the knowledge representation and control algorithm are combined, then a procedural definition applies. This is exemplified by the production rule methodology, where causality is implicit in the **IF .... THEN** rules. The ability to provide explanation of action, one of the prerequisites of an expert system, can not naturally be accommodated. However, to overcome this problem 'canned' explanations can be used. These are a set of fixed files, each one specific to a particular production rule, which can be fired whenever the production rule they represent is used. A disadvantage of this approach is that the explanation can appear 'stale' to an experienced user of the system. In systems where the knowledge representation and control algorithm are separate, then a declarative definition applies. The knowledge representation is explicitly defined, allowing a set of algorithms to act on it. This can include the test and hypothesis diagnostic cycle,

an explanation generating algorithm, and an algorithm which produces text which constitutes computer-based advice. Both frame-based knowledge representations and semantic networks are examples of declarative systems.

APPENDIX II

ANNOTATED PROGRAM LISTING OF AIRS

AII.1 Creating the window environment

To illustrate how a window environment is created, the program code shown in Figure A-II.1 is taken from the top-level module of AIRS, which describes the opening screen format. All the predicates used in this listing are predefined by the PROLOG-2 environment.

```
make_windows:- c_stream, c_screen, c_windows.

c_stream :-
create_stream(banner,readwrite,byte,
                          window(20,78,bright yellow on black)),
create_stream(header,readwrite,byte,window(1,80,white on blue)),
create_stream(blacking,readwrite,byte,window(25,80,black on black)),
create_stream(top_menu,readwrite,byte,
                           window(7,14,bright yellow on cyan)),
open(banner,readwrite),
open(header,readwrite),
open(blacking,readwrite),
open(top_menu,readwrite).

c_screen:-
screen(blacking,
 create(0,0,blacking,0,0,0,none,black on black,25,80,hidden)),
screen(header,
 create(0,0,header,0,0,0,none,white on blue,1,80,revealed)),
screen(banner,
 create(4,1,banner,0,0,0,all,yellow on black,20,78,revealed)),
screen(top_menu,
 create(5,65,top_menu,0,0,0,all,yellow on cyan,7,14,hidden)).

c_windows:-
screen(blacking,unhide),
window(banner,cursor_address(10,20)),
window(banner,
      text("AIRS - Artificial Intelligent Respirator System")),
window(banner,cursor_address(15,36)),
window(banner,text("WELCOME !")),
window(header,cursor_address(0,2)),
window(header,text(" AIRS Off-Line v2.0 ")).
*
*
*
close_and_delete :-
close(top_menu),delete_stream(top_menu),
close(header),delete_stream(header),
close(banner),delete_stream(banner),
close(blacking),delete_stream(blacking),
halt.
```

FIGURE A-II.1 Creating the Window Environment

The essence of any PROLOG program is to satisify logical

goals. In the program excerpt listed above the top-goal is **make_windows/0** (the "0" indicates that **make_windows** has no arguments), which will succeed only if **c_stream/0**, **c_screen/0** and **c_windows/0** all succeed in turn. For **c_stream/0** to succeed the 'built-in' predicate used to define a 'stream' is called. In PROLOG-2 'streams' are the data input and output channels to and from devices of various kinds. A window environment is considered by the system to be a device, so a means of accessing data has to be defined. This is accomplished by using the predicate

create_stream(Streamname,Access,Datatype,Description).

'Streamname' is the name of the stream to be created.
'Access' specifies the type of access allowed to the stream, and can be one of read (read only), write (write only), or readwrite (the stream can be read from and written to).
'Datatype' is either specified as ASCII or byte, depending on the data's binary form.
'Description' describes the stream, which can be a file, a window, or a special descriptor such as a printer.

The window descriptor also has three arguments, the first two describe the number of rows and columns the window is to cover on the screen, and the third argument describes the colour attributes of the window. For example, a colour attribute of 'yellow on black' indicates yellow coloured text on a black background.

Nothing can be achieved with this window stream until it is opened, using the predicate

open(Streamname,Access).

To display the defined window the screen predicate is used

screen(Name,Operation).

'Name' is the name of the viewport created.
'Operation' is the PROLOG structure which describes the desired operation.

The viewport can cover the entire window or any rectangular portion of it, however an error message will occur if the viewport is larger than its associated window. It is advantageous to give the viewport the same name as its window stream, thereby keeping like with like. To create a viewport the Operation argument of the screen/2 predicate is instantiated to

create(SY,SX,Win,WY,WX,D,M,Matt,H,W,R).

'SY and SX' define the position of the top left hand corner of the viewport on the screen.

'Win' is the name of the associated window stream.
'WY and WX' define the top left hand corner of the viewport on the window.
'D' is the depth of the viewport, that is, a depth of zero will ensure that the whole viewport can be seen.
'M' specifies which side of the viewport are to have margins in Left-Right-Bottom-Top order, although more commonly "all" or "none" are used.
'Matt' describes the colour attributes of the margins.
'H and W' define the size of the viewport in terms of rows and columns.
'R' is either "hidden" or "revealed", depending on whether the viewport is to be seen straightaway.

Once the window has been created there are various operations that can be used for its management, using the predicate

window(Streamname,Operation).

Two of these operations are illustrated in the source code shown, where the operator **cursor_address(Y,X)** is used to position the cursor, and the operator **text(" textstring")** is used to enter a welcome message.

In the code illustrated above, four windows are generated: the "banner" displays an opening message; the "header" displays a one line window at the top of the screen, which is used subsequently to enter the patient's name and identification number; the "blacking" window is a useful utility, as its purpose is to cover up any pre-existing windows or text on the screen before the window of interest is displayed; and the "top-menu" window describes the options available for continuation of the computer-based consultation.

After processing data, the windows have to be closed if the computer memory that they take up is to be re-used. This is achieved by using the predicate

close(Streamname).

To tidy up the memory completely, the window stream can be deleted using

delete_stream(Streamname).

The **halt** predicate returns the user to the underlying disk-operating system.

AII.2 Menu selection

A menu-driven user interface has been designed which allows

the full potential of the windows environment to be explored. AIRS comprises a hierarchy of menus. The code which describes the highest level menu is illustrated in Figure A-II.2 below.

```
display_topmenu :-
retractall(menu_selection(top_menu,_)),
assert(menu_selection(top_menu,0)),
repeat,
once(menu_selection(top_menu,Selection)),
once(menu(top_menu,"TOP-MENU",
                  ["Patient name"-"@"-true-1-help,
                   "Input data"-"@"-true-2-help,
                   "Database"-"@"-true-3-help,
                   "Analyse data"-"@"-true-4-help,
                   "Action"-"@"-true-5-help,
                   "Explanation"-"@"-true-6-help,
                   "eXit"-"@"-true-7-help],Option,Selection)),
once(screen(top_menu,unhide)),
once(retract(menu_selection(top_menu,Selection))),
once(assert(menu_selection(top_menu,Option))),
once(top_menu_action(Option)),
Option == 7,!.                          /* fails until exit selected */

top_menu_action(1) :-
screen(blacking,pull_up),
reconsult("rsptname.dat"),
ptname,
screen(header,pull_up).
*
*
*
top_menu_action(7):-
(store;true).

store :-
 current_data(demographic,"identity",Id),
 Display is_string "Archiving " & Id,
 decision_box(3,55,Display,Result),
 archive_patient(Result).
```

FIGURE A-II.2 Program Code for the Pull-down Menus

The code shown can be split into two functional units: **display_topmenu/0** creates the menu; and **top_menu_action/1** effects the menu choice. Before the **menu/5** predicate is encountered **for the first time** in **display_topmenu/0**, the internal PROLOG database (the 'workspace') is set to a dummy menu selection variable. This ensures that no conflict of choice occurs when the menu selection is made. The **'repeat'** predicate, in conjunction with the **'once'** built-in predicate, allows re-satisfaction of the menu options. This has the effect of keeping cursor control within the menu until menu option 7 is chosen ("exit"). When this occurs, immediately after **'top_menu_action(7)'** has been satisfied the final term of **display_topmenu/0** succeeds, the **'cut'**

operator (denoted by the exclamation mark) eliminates any back-tracking, and **display_topmenu/0** is exited.

Central to the menu selection routine is the **menu/5** predicate, which has the general format

menu(Window,Title,Menu,Answer,Start).

'Window' is the name of the window and associated viewport created to accommodate the menu. These must be of sufficient dimensions to enclose the menu options.
'Title' is the title of the window, and is displayed in the top margin of the menu selection box. This is an optional feature.
'Menu' is a list with each entry having five fields, as follows. (1) the selection descriptor, which must be a string; (2) an accelerator key, which must be a single character string, and has the same function as choosing the menu selection with the cursor keys (or mouse); (3) the enabling condition, which is a PROLOG goal that has to succeed before the menu is displayed; (4) the term which fires the action sequence; and (5) the string which is passed to the help system for user clarification.
'Answer' is the selected item.
'Start' is the menu option where the cursor bar appears, and is in reverse video mode.

To illustrate an application of this general format, consider the **menu/5** predicate found in the example above. The window and viewport are called "top_menu"; the menu window has the title "TOP MENU". The seven menu options are the first fields of the entries in the list; no accelerator keys are used, so these are all matched to "@"; the enabling condition is the predicate **true/0**, which always succeeds when called; the selected item is indicated by an integer; and the help system is not activated. The **Answer** and **Start** arguments of the **menu/5** predicate are matched to the variables **Option** and **Selection** respectively. This allows the starting position (Selection) of the cursor bar to be placed on the previous choice, and instantiates "Option" to the integer of the present menu selection. This integer is then used in the **top_menu_action/1** predicate to effect the appropriate action.

The predicate **top_menu_action/1** consults one of the second level files and initiates action by calling the top goal of the opened file. This activates a second level menu system which operates in the same way as the top level system. In some instances third and fourth

level menus are used, but each menu node must eventually exhibit some action. For example, this may be for data capture, displaying a database or displaying an advice screen. The file is closed automatically when exiting from the lower level menu system, enabling cursor control to revert to the top-menu. This successive use of menus is the way in which control is exhibited within AIRS.

When "exit" is chosen from the top-menu, **top_menu_action(7)** is fired which terminates the menu-driven action of the system. Before returning to the Disk Operating System any patient details are either updated and stored on disk or (the ";" operator is interpreted as "or") the **true/0** predicate succeeds and the AIRS consultation is terminated immediately.

AII.3 First level of intelligence

The initialisation of the patient on the ventilator assumes that diagnostic state is known, as it is this knowledge which determines the information that is shown on the screen. In the program excerpt shown in Figure A-II.3 only one of the diagnostic states is shown. This is indicative of all the others.

```
init_menu_action(1) :-
analysis_init(1),
action(cardiac,init),
explain(cardiac,init).
*
*
init_menu_action(16) :-
screen(analysis,unhide),
screen(action,unhide).

init_menu_action(17) :-
screen(explanation,unhide).

analysis_init(1) :-
screen(analysis,unhide),
window(analysis,cursor_address(0,1)),
window(analysis,text("Suggested Ventilator Settings")),
window(analysis,cursor_address(1,1)),
window(analysis,text("for Cardiac Patients :-")),
window(analysis,cursor_address(3,1)),
window(analysis,text("RR      12 per min")),
window(analysis,cursor_address(4,1)),
window(analysis,text("TV      10 ml per Kg")),
window(analysis,cursor_address(5,1)),
window(analysis,text("I:E      1")),
window(analysis,cursor_address(6,1)),
window(analysis,text("PEEP     5 cmH2O ***")),
window(analysis,cursor_address(7,1)),
window(analysis,text("FIO2     0.5")).
```

FIGURE A-II.3 Initialisation of a Cardiac Patient

The predicate **init_menu_action/1** is the action product of the initialisation menu and is responsible for firing the goal predicates which display the suggested ventilator settings (**analysis_init/1**), the action screen (**action/2**), and the explanation screen (**explain/2**). These latter two predicates are found in the files "rsact.pro" and "rsexpln.pro" respectively.

A facility to switch between the analysis/action screen and the explanation screen has been added as a feature in this menu (**init_menu_action(16)** and **init_menu_action(17)** being the explicit goals to achieve this action).

AII.4 Second level of intelligence

At the centre of the sub-program which exhibits the second level of intelligence is a value-matching algorithm. The program code illustrated in Figure A-II.4 shows an example of the value-matching algorithm being used in the determination of the set_point alarm status of PCO_2.

```
check_status(cmv,set_pt) :-
check_pco2(cmv,set_pt).

check_pco2(cmv,set_pt):-
lab_data(_,_,Index,"PCO2  ",V1),
Value1 is value(V1,ir),              /*  numerical string to atom  */
Value1 < 3,!,                                 /*  low < 3 KPa  */
fill_top_blue(pco2_status),
action(cmv,set_pt,pco2,low), explanation(cmv,set_pt,pco2,low).

check_pco2(cmv,set_pt) :-
lab_data(_,_,Index,"PCO2  ",V1),
Value1 is value(V1,ir),              /*  numerical string to atom  */
Value1 > 9,!,                                 /*  high > 9 KPa  */
fill_top_red(pco2_status),
action(cmv,set_pt,pco2,high), explanation(cmv,set_pt,pco2,high).

check_pco2(cmv,set_pt) :-
fill_top_green(pco2_status).
                          /*  Not HIGH or LOW, therefore NORMAL.
                              Screen command required for reset. */
check_pco2(cmv,set_pt).                                /*  catch all  */
```

FIGURE A-II.4 Value-matching Algorithm for PCO_2 Limit Alarm

The predicate **check_status/2** comprises twenty-one clauses, one for each of the seven data variables in the three modes of ventilation recognised by the system. P_aCO_2 is the data variable in the illustrative example above, where the value-matching algorithm shows the data values for changing set-point alarm status when the

mode of ventilation is continuous mandatory ventilation (CMV). There are four clauses for **check_pco2/2**, in the first two of these the value for P_aCO_2 is obtained from the internal PROLOG database using **lab_data/5**. This value is then converted from a string to an atom using **value/2**, and tested against the set-point criterion. If the **Value1** term is passed the colour attribute of the upper half of the window is changed accordingly. The order in which the clauses for **check_pco2/2** occur is important, as if the value fails the first two clauses it is automatically assumed to reside within the normal range. A fourth clause is added for **check_pco2/2** which has no goal, this acts as a software safety mechanism and returns all failures.

AII.5 Third level of intelligence

AII.5.1 Structure of the rulebase

The rule-based knowledge of the weaning element of AIRS is represented in PROLOG by the clauses **premise/4** and **action/3**. A rule of the form:

IF condition 1
AND condition 2
OR condition 3
AND condition 4
THEN action 1
AND action 2

is represented in a general form as

```
premise(rule_number_1,premise_number_1,rule_type,Data):-
     condition 1,
     condition 2.
premise(rule_number_1,premise_number_2,rule_type,Data):-
     condition 3,
     condition 4.
action(rule_number_1,Premise_variable,Data):-
     action 1,
     action 2.
```

The premise conditions can be any PROLOG goal, but several standard conditions are defined:

equals/5
not_equal/5
greater_than_or_equal/5
less_than_or_equal/5
one_of/5

Figure A-II.5 shows an example of program code, the 4th rule of the weaning rule-set, which has two premises that deal with impaired energy supply. The premises indicate that this situation can occur when the patient has impaired oxygen delivery or a nutritional deficiency. A full listing of the program code for the third level of intelligence is found in Section AII.8.

```
premise(4,1,weaning,[impaired_oxygen_delivery]):-
equals(4,1,weaning,impaired_oxygen_delivery,yes).

premise(4,2,weaning,[nutritional_deficiency]):-
equals(4,2,weaning,nutritional_deficiency,yes).

action(4,P,[impaired_energy_supply]):-
conclude(4,P,impaired_energy_supply,transient,yes).
```

FIGURE A-II.5 Rule for Impaired Energy Supply

The predicates listed above test the current values of data stored in a dynamic database defined by PROLOG facts **current_value/3**:

current_value(Dataname,Datatype,Datavalue).

For the action goals, one standard predicate is used to update the **current_value/3** facts, that is:

conclude/5.

AII.5.2 The role of premise requirements

The 4th argument of the **premise/4** clauses is a list of all the Data-names that appear in **equals/5** conditions for the premise. The values of these data items must exist in the database if the premise is to evaluate successfully. Hence, the list of Data-names is called the 'premise requirements' list for the rule premise.

Before the rules which comprise the inference engine are fired, the predicate **generate_premise_requirements/0** checks each premise clause and asserts a fact:

premise_requirements(Rule,Premise,Type,Requirements)

for each rule premise, where the four arguments are the same as those in the head of the **premise/4** clause.

Whenever the value of a data item is concluded with **conclude/5** the **premise_requirements** facts are searched and the name of the data item whose value is concluded is removed from the

requirements list (this is achieved by the predicate **reduce_premise_requirements/2**).

Whenever the value of a data item is tested using one of the standard premise conditions (**equals/5** etc.) and the test fails, the name of the data item is added to the requirements list in the **premise_requirements/4** fact for that rule premise.

The rules for which the requirements list is empty are suitable for testing on the next cycle of the inference engine.

AII.5.3 The inference engine

The inference engine operates in a simple cycle, shown in Figure A-II.6, activated by the predicate **control_cycle/0**. The set of active rules is formed in two stages. First a set is formed of all rules whose requirements list is empty (in **premise_requirements/4** facts). Then for each rule in this set the premise is evaluated (by calling the **premise/4** goal) and the set of active rules is formed from all rules whose premise succeeds. The two phases of rule activation are achieved by the predicates **first_pass_activation/1** and **second_pass_activation/2**.

The predicate **schedule_rules/3** is passed the list of active rules and returns them to be executed one at a time. The order that the rules are scheduled is controlled by a simple algorithm:

goal — **top priority**
weaning
regression
progression
request — **lowest priority**

On each cycle the rule returned by **schedule_rule/3** is executed (that is, the **action/3** predicate is called.

AII.6 Data capture

A menu-type interface is used to enter data into AIRS. At one level the menu options are the type of data (for example, ventilator, monitor, nurse observations), and at a lower level the menu options are the data items themselves (for example,tidal volume and respiratory rate). The program excerpt in Figure A-II.7 illustrates data capture from the ventilator, and shows two predicates : **vent_data/0** and **input_data/2**.

```
vent_data :-
repeat,
once(get_archive_time(Date,Time)),
once(vent_display([Day,Month,Year],[Hour,Min])),
once(create_data_display(Date,Time,"RR    ",Display_RR)),
once(menu(input_ventdata,"Data Input",
[Display_RR-"@"-create_data_display(Date,Time,"MV    ",
                    Display_MV)-input_data("RR    ",Index)-help,
 Display_MV-"@"-create_data_display(Date,Time,"MawP  ",
                    Display_MawP)-input_data("MV    ",Index)-help,
 Display_MawP-"@"-create_data_display(Date,Time,"IE    ",
                    Display_IE)-input_data("MawP  ",Index)-help,
 Display_IE-"@"-create_data_display(Date,Time,"TV    ",
                    Display_TV)-input_data("IE    ",Index)-help,
 Display_TV-"@"-create_data_display(Date,Time,"SMV   ",
                    Display_SMV)-input_data("TV    ",Index)-help,
 Display_SMV-"@"-create_data_display(Date,Time,"PawP  ",
                    Display_PawP)-input_data("SMV   ",Index)-help,
 Display_PawP-"@"-create_data_display(Date,Time,"PP    ",
                    Display_PP)-input_data("PawP  ",Index)-help,
 Display_PP-"@"-true-input_data("PP    ",Index)-help,
 "exit"-"@"-true-exit-help],  Option,0)),
once(call(Option)),
Option == exit,
window(ventdb,scroll_up),
scroll_window_down(ventdatdb),
close_all.

input_data(Measurand,Index):-
date(Day,Month,Year), time(Hour,Minute,Sec),
Date = [Day,Month,Year], Time = [Hour,Minute],
pt_index(Index),
screen(input_ventdata,unhide),
Prompt is_string Measurand,
Value_string is_string ?(Value,
   fedit(5,1,30,"Input Data",Prompt," ",green on black,Value)),
asserta(vent_data(Date,Time,Index,Measurand,Value_string)),
display_data(Measurand,Value_string).
```

FIGURE A-II.7 Program Code for Data Capture

In the **vent_data/0** predicate **repeat/0** is again used to keep cursor control within the menu until the "exit" option is chosen. The predicates **get_archive_time/2**, **vent_display/2** and **create_data_display/4** are involved with the on-screen presentation of the captured data and are discussed in the next section.

The **menu/5** predicate within **vent_data/0** describes a more sophisticated menu system than previously encountered. For any option except the last one in the menu list, the enabling condition is the **create_data_display/4** predicate of the next item in the menu. The enabling condition of the last listed item is the predicate **true/0** which automatically succeeds. To complicate the understanding of the

program even further, the display token of the chosen menu item is the last argument of the previous **create_data_display/4** predicate. The corresponding argument for the first item in the list is defined before the **menu/5** predicate is fired. The return token for every menu option is the predicate **input_data/2**. The data capture menus have been implemented in this way to facilitate data processing.

When it is called, the first argument of **input_data/2** is instantiated to the chosen measureand. The date and time are then taken from the system clock to act as a time-stamp for the data file and the index number of the patient is checked to ensure that the correct data is transferred. A built-in predicate **fedit/8** is used as a means to enter the data, and has the general format

fedit(SY,SX,Length,Title,Prompt,Def,Att,Ans).

'SY and SX' define the position of the top left corner of the window and associated viewport.
'Length' is the width of the window (by default an **fedit/8** window is specified as having only one row).
'Title' is the title of the window, and is optional.
'Prompt' is the prompt which is written to the left of the window.
'Def' is the data string to be entered.
'Att' is the colour attribute of the window.
'Value' is the output string.

Hence, the program excerpt shows a window that is thirty characters long and the top left corner appears five rows down and one column across. The window is entitled "Input Data", the prompt is the required measurand (to be instantiated) and the characters are green on a black background. When the input window first appears the data entry area is blank and awaits input by the user. The value entered can be edited as necessary, as it is not matched to "Value" until the return key is pressed. The number inputted is converted to a string before saving in the PROLOG workspace in the form

vent_data(Date,Time,Index,Measurand,Value_string).

A typical entry could be

vent_data([01,03,89],[14,30],Pt3,"RR ","12").

This can be interpreted as, "The patient whose identity is Pt3 had a respiratory rate of 12 breaths per minute at 14:30 hours on 1st March, 1989."

There are similar clauses for monitor data, nurse observations etc. which have the same argument format to **vent_data/5** above. It is these clauses which are saved to disk when exiting from

AIRS. They are stored in files called "PTX.ARC", where X is the appropriate patient number.

AII.7 Data presentation

The predicates shown in Figure A-II.8 are used for data capture and for data presentation.

```
get_archive_time(Date,Time) :-
vent_data(Date,Time,_,_,_).

get_archive_time(Date,Time):-
date(Day,Month,Year), time(Hour,Minute,Sec),
Date = [Day,Month,Year], Time = [Hour,Minute].

vent_display(_,_):-
date_display(Date), time_display(Time),
window(ventdb,cursor_address(5,1)),
Datadisplay is_string Date & " " & Time,
window(ventdb,text(Datadisplay)),
window(ventdatdb,cursor_address(0,1)),
window(ventdatdb,text(Datadisplay)).

create_data_display(Date,Time,Measurand,Data_display) :-
vent_data(Date,Time,Index,Measurand,Value),
output_format(Value,Display_value),
units(Measurand,Units),
Data_display is_string Measurand & Display_value & Units.

create_data_display(Date,Time,Measurand,Data_display) :-
units(Measurand,Units),
Data_display is_string Measurand & "      " & Units.

units("RR    "," per min"). units("MV    "," litres").
units("MawP  "," cmH2O"). units("IE    "," ").
units("TV    "," litres"). units("SMV   "," litres").
units("PawP  "," cmH2O"). units("PP    "," cmH2O").

display_data(Measurand,Value) :-
clause(vent_tab_head/1,vent_tab_head(Measurand),Pos),
Xpos is 10 + Pos*8,
Xposdb is 10 + Pos*8,
window(ventdb,inquire_cursor_address(CY,X)),
window(ventdb,cursor_address(CY,Xpos)),
window(ventdb,text(Value)),
window(ventdatdb,inquire_cursor_address(CYdb,Xdb)),
window(ventdatdb,cursor_address(CYdb,Xposdb)),
window(ventdatdb,text(Value)).

vent_tab_head("RR    "). vent_tab_head("MV    ").
vent_tab_head("MawP  "). vent_tab_head("IE    ").
vent_tab_head("TV    "). vent_tab_head("SMV   ").
vent_tab_head("PawP  "). vent_tab_head("PP    ").
```

FIGURE A-II.8 Program Code for Data Presentation

There are two clauses for the predicate **get_archive_time/2** which is called from within the data capture menu. The first succeeds

if the date and time from the system clock has remained the same from the previous call to this predicate. However if a time boundary (minutes are the most significant time boundary) has been passed, the first clause fails and the second clause is used to update the time (and date, if the time is midnight). The predicate **vent_display/2** is used to display the date and time in the data capture and database windows.

The predicate **create_data_display/4** also has two clauses, which one of them succeeds also depends on whether the time has been updated. Its use is to return the term **Data_display**, which shows the data entry options available. Within this clause, **output_format/2** is used to convert all numbers to an accuracy of two decimal places; and **units/2**, as illustrated in the code above, returns the appropriate units of measurement for the entered data.

The predicate **display_data/2** is used to enter the captured data entries into their respective positions in the data capture and database windows. Date and time require ten columns for their entry, so the position of the data entries are off-set by that amount. The clauses for **vent_tab_head/1** are position sensitive, the variable term **Pos** being matched to the numerical order of the entered **Measurand**. Each data item has a field of eight columns in the database, so the cursor position for data entry into the data capture window is given by the term **Xpos**. The same algorithm is used for placing data at the appropriate position within the database window.

AII.8 Program listing for the third level of intelligence

```
/* _________________ rswean rulebase _______________ */

/* Form of rule premises :
   1st argument - Rule_no
   2nd argument - Premise_no
   3rd argument - Rule_type
   4th argument - Dependent variables */
/* ____________________________________________________ */

premise(goal,1,goal,[]):-
true.
      /* top goal always fires */

premise(terminate,1,terminate,[]):-
true.
     /* end goal always fires */

/* ------- Rule premises - weaning -------- */

premise(0,1,weaning,[fit_to_wean]):-
equals(0,1,weaning,fit_to_wean,yes).

premise(1,1,weaning,[respiratory_muscle_fatigue]):-
equals(1,1,weaning,respiratory_muscle_fatigue,yes).

premise(1,2,weaning,[f_e_imbalance]):-
equals(1,2,weaning,f_e_imbalance,yes).

premise(1,3,weaning,[system_failure]):-
equals(1,3,weaning,system_failure,yes).

premise(1,4,weaning,[anxiety]):-
equals(1,4,weaning,anxiety,yes).

premise(1,5,weaning,[feeding_problems]):-
equals(1,5,weaning,feeding_problems,yes).

premise(2,1,weaning,[excessive_muscle_demands]):-
equals(2,1,weaning,excessive_muscle_demands,yes).

premise(2,2,weaning,[impaired_energy_supply]):-
equals(2,2,weaning,impaired_energy_supply,yes).

premise(3,1,weaning,[increased_respiratory_resistance]):-
equals(3,1,weaning,increased_respiratory_resistance,yes).

premise(3,2,weaning,[fever_infection]):-
equals(3,2,weaning,fever_infection,yes).

premise(3,3,weaning,[increased_co2_production]):-
equals(3,3,weaning,increased_co2_production,yes).

premise(4,1,weaning,[impaired_oxygen_delivery]):-
equals(4,1,weaning,impaired_oxygen_delivery,yes).

premise(4,2,weaning,[nutritional_deficiency]):-
equals(4,2,weaning,nutritional_deficiency,yes).
```

```
premise(5,1,weaning,[breathing_circuit_inappropriate]):-
equals(5,1,weaning,breathing_circuit_inappropriate,yes).

premise(5,2,weaning,[bronchospasm]):-
equals(5,2,weaning,bronchospasm,yes).

premise(5,3,weaning,[ascites]):-
equals(5,3,weaning,ascites,yes).

premise(5,4,weaning,[obesity]):-
equals(5,4,weaning,obesity,yes).

premise(6,1,weaning,[reduced_oxygen_consumption]):-
equals(6,1,weaning,reduced_oxygen_consumption,yes).

premise(6,2,weaning,[increased_work_of_breathing]):-
equals(6,2,weaning,increased_work_of_breathing,yes).

premise(7,1,weaning,[catabolic]):-
equals(7,1,weaning,catabolic,yes).

premise(8,1,weaning,[sensitivity]):-
equals(8,1,weaning,sensitivity,yes).

premise(9,1,weaning,[increased_dynamic_resistance]):-
equals(9,1,weaning,increased_dynamic_resistance,yes).

premise(10,1,weaning,[hypovolaemia]):-
equals(10,1,weaning,hypovolaemia,yes).

premise(10,2,weaning,[anaemia]):-
equals(10,2,weaning,anaemia,yes).

premise(10,3,weaning,[pulmonary_oedema]):-
equals(10,3,weaning,pulmonary_oedema,yes).

premise(10,4,weaning,[potassium_level_out]):-
equals(10,4,weaning,potassium_level_out,yes).

premise(10,5,weaning,[phosphate_level]):-
equals(10,5,weaning,phosphate_level,yes).

premise(11,1,weaning,[c_v_pressure]):-
equals(11,1,weaning,c_v_pressure,yes).

premise(12,1,weaning,[haemoglobin_count]):-
equals(12,1,weaning,haemoglobin_count,yes).

premise(13,1,weaning,[p_a_wedge_pressure]):-
equals(13,1,weaning,p_a_wedge_pressure,yes).

premise(13,2,weaning,[colloid_osmotic_pressure]):-
equals(13,2,weaning,colloid_osmotic_pressure,yes).

premise(14,1,weaning,[potassium_high]):-
equals(14,1,weaning,potassium_high,yes).

premise(14,2,weaning,[potassium_low]):-
equals(14,2,weaning,potassium_low,yes).
```

```
premise(15,1,weaning,[phosphate_high]):-
equals(15,1,weaning,phosphate_high,yes).

premise(16,1,weaning,[respiratory_failure]):-
equals(16,1,weaning,respiratory_failure,yes).

premise(16,2,weaning,[cardiac_failure]):-
equals(16,2,weaning,cardiac_failure,yes).

premise(16,3,weaning,[neurological]):-
equals(16,3,weaning,neurological,yes).

premise(16,4,weaning,[metabolic_acid_base]):-
equals(16,4,weaning,metabolic_acid_base,yes).

premise(17,1,weaning,[ventilatory_failure]):-
equals(17,1,weaning,ventilatory_failure,yes).

premise(17,2,weaning,[inefficient_pulmonary_gas_exchange]):-
equals(17,2,weaning,inefficient_pulmonary_gas_exchange,yes).

premise(18,1,weaning,[tidal_volume]):-
equals(18,1,weaning,tidal_volume,yes).

premise(19,1,weaning,[capAaDO2]):-
equals(19,1,weaning,capAaDO2,yes).

premise(20,1,weaning,[cardiac_output_low]):-
equals(20,1,weaning,cardiac_output_low,yes).

premise(20,2,weaning,[acute_lvf]):-
equals(20,2,weaning,acute_lvf,yes).

premise(21,1,weaning,[cardiac_index]):-
equals(21,1,weaning,cardiac_index,yes).

premise(22,1,weaning,[p_a_wedge_pressure]):-
equals(22,1,weaning,p_a_wedge_pressure,yes).

premise(23,1,weaning,[metabolic_alkalosis]):-
equals(23,1,weaning,metabolic_alkalosis,yes).

premise(23,2,weaning,[respiratory_alkalosis]):-
equals(23,2,weaning,respiratory_alkalosis,yes).

premise(23,3,weaning,[renal_problem]):-
equals(23,3,weaning,renal_problem,yes).

premise(23,4,weaning,[hepatic_problem]):-
equals(23,4,weaning,hepatic_problem,yes).

premise(24,1,weaning,[creatinine_high]):-
equals(24,1,weaning,creatinine_high,yes).

premise(25,1,weaning,[lft_deranged]):-
equals(25,1,weaning,lft_deranged,yes).

premise(26,1,weaning,[sleep_deprivation]):-
equals(26,1,weaning,sleep_deprivation,yes).
```

```
premise(26,2,weaning,[primary_anxiety]):-
equals(26,2,weaning,primary_anxiety,yes).

premise(26,3,weaning,[pain]):-
equals(26,3,weaning,pain,yes).

premise(26,4,weaning,[sedation]):-
equals(26,4,weaning,sedation,yes).

premise(27,1,weaning,[sleeping_draught]):-
equals(27,1,weaning,sleeping_draught,yes).

premise(28,1,weaning,[anxiolytic_agent]):-
equals(28,1,weaning,anxiolytic_agent,yes).

premise(29,1,weaning,[local_blocker]):-
equals(29,1,weaning,local_blocker,yes).

premise(29,2,weaning,[infiltration]):-
equals(29,2,weaning,infiltration,yes).

premise(32,1,weaning,[sedation]):-
equals(32,1,weaning,sedation,yes).

premise(33,1,weaning,[atelactasis]):-
equals(33,1,weaning,atelactasis,yes).

premise(33,2,weaning,[decreased_ventilatory_response]):-
equals(33,2,weaning,decreased_ventilatory_response,yes).

premise(34,1,weaning,[ineffective_cough]):-
equals(34,1,weaning,ineffective_cough,yes).

premise(35,1,weaning,[hypoxia]):-
equals(35,1,weaning,hypoxia,yes).

premise(35,2,weaning,[hypercapnia]):-
equals(35,2,weaning,hypercapnia,yes).
```

```
/* ---------- rule premises - regression -------- */

premise(100,1,regression,[regress]):-
equals(100,1,regression,regress,yes).

premise(101,1,regression,[respiratory_muscle_weakness_fatigue]):-
equals(101,1,regression,respiratory_muscle_weakness_fatigue,yes).

premise(101,2,regression,[decreased_respiratory_drive]):-
equals(101,2,regression,decreased_respiratory_drive,yes).

premise(101,3,regression,[increased_respiratory_drive]):-
equals(101,3,regression,increased_respiratory_drive,yes).

premise(102,1,regression,[rr_gt_25_and_tv_vlow]):-
equals(102,1,regression,rr_gt_25_and_tv_vlow,yes).

premise(102,2,regression,[abnormal_breathing]):-
equals(102,2,regression,abnormal_breathing,yes).

premise(103,1,regression,[abdominal_paradox]):-
equals(103,1,regression,abdominal_paradox,yes).

premise(103,2,regression,[respiratory_alternans]):-
equals(103,2,regression,respiratory_alternans,yes).

premise(104,1,regression,[increased_tv_inspiratory_time_ratio]):-
equals(104,1,regression,increased_tv_inspiratory_time_ratio,yes).

premise(105,1,regression,[p01_gt_6]):-
equals(105,1,regression,p01_gt_6,yes).

premise(105,2,regression,[increased_co2_production]):-
equals(105,2,regression,increased_co2_production,yes).



/* ---------- rule premises - progression -------- */

premise(200,1,progression,[progress]):-
equals(200,1,progression,progress,yes).

premise(200,2,progression,[regress]):-
equals(200,2,progression,regress,yes).

premise(200,3,progression,[review]):-
equals(200,3,progression,review,yes).

premise(201,1,progression,[ph_same_paco2_up]):-
equals(201,1,progression,ph_same_paco2_up,yes).

premise(201,2,progression,[ph_same_paco2_same]):-
equals(201,2,progression,ph_same_paco2_same,yes).

premise(201,3,progression,[ph_same_paco2_down]):-
equals(201,3,progression,ph_same_paco2_down,yes).
```

```
premise(202,1,progression,[ph_up_paco2_up]):-
equals(202,1,progression,ph_up_paco2_up,yes).

premise(202,2,progression,[ph_down_paco2_up]):-
equals(202,2,progression,ph_down_paco2_up,yes).

premise(203,1,progression,[ph_up_paco2_same]):-
equals(203,1,progression,ph_up_paco2_same,yes).

premise(203,2,progression,[ph_up_paco2_down]):-
equals(203,2,progression,ph_up_paco2_down,yes).

premise(203,3,progression,[ph_down_paco2_same]):-
equals(203,3,progression,ph_down_paco2_same,yes).

premise(203,4,progression,[ph_down_paco2_down]):-
equals(203,4,progression,ph_down_paco2_down,yes).

premise(204,1,progression,[ph_same1]):-
equals(204,1,progression,ph_same1,yes).

premise(204,2,progression,[paco2_up1]):-
equals(204,2,progression,paco2_up1,yes).

premise(205,1,progression,[ph_same2]):-
equals(205,1,progression,ph_same2,yes).

premise(205,2,progression,[paco2_same1]):-
equals(205,2,progression,paco2_same1,yes).

premise(206,1,progression,[ph_same3]):-
equals(206,1,progression,ph_same3,yes).

premise(206,2,progression,[paco2_down1]):-
equals(206,2,progression,paco2_down1,yes).

premise(207,1,progression,[ph_up1]):-
equals(207,1,progression,ph_up1,yes).

premise(207,2,progression,[paco2_up2]):-
equals(207,2,progression,paco2_up2,yes).

premise(208,1,progression,[ph_down1]):-
equals(208,1,progression,ph_down1,yes).

premise(208,2,progression,[paco2_up3]):-
equals(208,2,progression,paco2_up3,yes).

premise(209,1,progression,[ph_up2]):-
equals(209,1,progression,ph_up2,yes).

premise(209,2,progression,[paco2_same2]):-
equals(209,2,progression,paco2_same2,yes).

premise(210,1,progression,[ph_up3]):-
equals(210,1,progression,ph_up3,yes).

premise(210,2,progression,[paco2_down2]):-
equals(210,2,progression,paco2_down2,yes).
```

```
premise(211,1,progression,[ph_down2]):-
equals(211,1,progression,ph_down2,yes).

premise(211,2,progression,[paco2_same3]):-
equals(211,2,progression,paco2_same3,yes).

premise(212,1,progression,[ph_down3]):-
equals(212,1,progression,ph_down3,yes).

premise(212,2,progression,[paco2_down3]):-
equals(212,2,progression,paco2_down3,yes).

premise(213,1,progression,[ph_is_same]):-
equals(213,1,progression,ph_is_same,yes).

premise(214,1,progression,[paco2_is_up]):-
equals(214,1,progression,paco2_is_up,yes).

premise(215,1,progression,[ph_is_same]):-
equals(215,1,progression,ph_is_same,yes).

premise(216,1,progression,[paco2_is_same]):-
equals(216,1,progression,paco2_is_same,yes).

premise(217,1,progression,[ph_is_same]):-
equals(217,1,progression,ph_is_same,yes).

premise(218,1,progression,[paco2_is_down]):-
equals(218,1,progression,paco2_is_down,yes).

premise(219,1,progression,[ph_is_up]):-
equals(219,1,progression,ph_is_up,yes).

premise(220,1,progression,[paco2_is_up]):-
equals(220,1,progression,paco2_is_up,yes).

premise(221,1,progression,[ph_is_down]):-
equals(221,1,progression,ph_is_down,yes).

premise(222,1,progression,[paco2_is_up]):-
equals(222,1,progression,paco2_is_up,yes).

premise(223,1,progression,[ph_is_up]):-
equals(223,1,progression,ph_is_up,yes).

premise(224,1,progression,[paco2_is_same]):-
equals(224,1,progression,paco2_is_same,yes).

premise(225,1,progression,[ph_is_up]):-
equals(225,1,progression,ph_is_up,yes).

premise(226,1,progression,[paco2_is_down]):-
equals(226,1,progression,paco2_is_down,yes).

premise(227,1,progression,[ph_is_down]):-
equals(227,1,progression,ph_is_down,yes).

premise(228,1,progression,[paco2_is_same]):-
equals(228,1,progression,paco2_is_same,yes).
```

```
premise(229,1,progression,[ph_is_down]):-
equals(229,1,progression,ph_is_down,yes).

premise(230,1,progression,[paco2_is_down]):-
equals(230,1,progression,paco2_is_down,yes).




/* ------------------------------------------------------------ */
/* ------------ negative weaning conclusions ----------------- */

premise(500,1,weaning,[fit_to_wean]):-
equals(500,1,weaning,fit_to_wean,no).

premise(501,1,weaning,[respiratory_muscle_fatigue,f_e_imbalance,
                       system_failure,anxiety,feeding_problems]):-
equals(501,1,weaning,respiratory_muscle_fatigue,no),
equals(501,2,weaning,f_e_imbalance,no),
equals(501,3,weaning,system_failure,no),
equals(501,4,weaning,anxiety,no),
equals(501,5,weaning,feeding_problems,no).

premise(502,1,weaning,[impaired_energy_supply,
                       excessive_muscle_demands]):-
equals(502,1,weaning,impaired_energy_supply,no),
equals(502,2,weaning,excessive_muscle_demands,no).

premise(503,1,weaning,[increased_respiratory_resistance,
                       fever_infection,increased_co2_production]):-
equals(503,1,weaning,increased_respiratory_resistance,no),
equals(503,2,weaning,fever_infection,no),
equals(503,3,weaning,increased_co2_production,no).

premise(504,1,weaning,[impaired_oxygen_delivery,
                       nutritional_deficiency]):-
equals(504,1,weaning,impaired_oxygen_delivery,no),
equals(504,2,weaning,nutritional_deficiency,no).

premise(505,1,weaning,[breathing_circuit_inappropriate,
                       bronchospasm,ascites,obesity]):-
equals(505,1,weaning,breathing_circuit_inappropriate,no),
equals(505,2,weaning,bronchospasm,no),
equals(505,3,weaning,ascites,no),
equals(505,4,weaning,obesity,no).

premise(506,1,weaning,[reduced_oxygen_consumption,
                       increased_work_of_breathing]):-
equals(506,1,weaning,reduced_oxygen_consumption,no),
equals(506,2,weaning,increased_work_of_breathing,no).

premise(507,1,weaning,[catabolic]):-
equals(507,1,weaning,catabolic,no).

premise(508,1,weaning,[sensitivity]):-
equals(508,1,weaning,sensitivity,no).
```

```
premise(509,1,weaning,[increased_dynamic_resistance]):-
equals(509,1,weaning,increased_dynamic_resistance,no).

premise(510,1,weaning,[hypovolaemia,anaemia,pulmonary_oedema,
                    potassium_level_out,phosphate_level]):-
equals(510,1,weaning,hypovolaemia,no),
equals(510,2,weaning,anaemia,no),
equals(510,3,weaning,pulmonary_oedema,no),
equals(510,4,weaning,potassium_level_out,no),
equals(510,5,weaning,phosphate_level,no).

premise(511,1,weaning,[c_v_pressure]):-
equals(511,1,weaning,c_v_pressure,no).

premise(512,1,weaning,[haemoglobin_count]):-
equals(512,1,weaning,haemoglobin_count,no).

premise(513,1,weaning,[p_a_wedge_pressure_high,
                    colloid_osmotic_pressure]):-
equals(513,1,weaning,p_a_wedge_pressure_high,no),
equals(513,2,weaning,colloid_osmotic_pressure,no).

premise(514,1,weaning,[potassium_high,potassium_low]):-
equals(514,1,weaning,potassium_high,no),
equals(514,2,weaning,potassium_low,no).

premise(515,1,weaning,[phosphate_high]):-
equals(515,1,weaning,phosphate_high,no).

premise(516,1,weaning,[respiratory_failure,cardiac_failure,
                    neurological,metabolic_acid_base]):-
equals(516,1,weaning,respiratory_failure,no),
equals(516,2,weaning,cardiac_failure,no),
equals(516,3,weaning,neurological,no),
equals(516,4,weaning,metabolic_acid_base,no).

premise(517,1,weaning,[ventilatory_failure,
                    inefficient_pulmonary_gas_exchange]):-
equals(517,1,weaning,ventilatory_failure,no),
equals(517,2,weaning,inefficient_pulmonary_gas_exchange,no).

premise(518,1,weaning,[tidal_volume]):-
equals(518,1,weaning,tidal_volume,no).

premise(519,1,weaning,[capAaDO2]):-
equals(519,1,weaning,capAaDO2,no).

premise(520,1,weaning,[cardiac_output_low,acute_lvf]):-
equals(520,1,weaning,cardiac_output_low,no),
equals(520,2,weaning,acute_lvf,no).

premise(521,1,weaning,[cardiac_index]):-
equals(521,1,weaning,cardiac_index,no).

premise(522,1,weaning,[p_a_wedge_pressure_low]):-
equals(522,1,weaning,p_a_wedge_pressure_low,no).
```

```
premise(523,1,weaning,[metabolic_alkalosis,respiratory_alkalosis,
                    renal_problem,hepatic_problem]):-
equals(523,1,weaning,metabolic_alkalosis,no),
equals(523,2,weaning,respiratory_alkalosis,no),
equals(523,3,weaning,renal_problem,no),
equals(523,4,weaning,hepatic_problem,no).

premise(524,1,weaning,[creatinine_high]):-
equals(524,1,weaning,creatinine_high,no).

premise(525,1,weaning,[lft_deranged]):-
equals(525,1,weaning,lft_deranged,no).

premise(526,1,weaning,[sleep_deprivation,primary_anxiety,
                    pain,sedation]):-
equals(526,1,weaning,sleep_deprivation,no),
equals(526,2,weaning,primary_anxiety,no),
equals(526,3,weaning,pain,no),
equals(526,4,weaning,sedation,no).

premise(527,1,weaning,[sleeping_draught]):-
equals(527,1,weaning,sleeping_draught,no).

premise(528,1,weaning,[anxiolytic_agent]):-
equals(528,1,weaning,anxiolytic_agent,no).

premise(529,1,weaning,[local_blocker,infiltration]):-
equals(529,1,weaning,local_blocker,no),
equals(529,2,weaning,infiltration,no).

premise(532,1,weaning,[sedation]):-
equals(532,1,weaning,sedation,no).

premise(533,1,weaning,[atelactasis,
                    decreased_ventilatory_response]):-
equals(533,1,weaning,atelactasis,no),
equals(533,2,weaning,decreased_ventilatory_response,no).

premise(534,1,weaning,[ineffective_cough]):-
equals(534,1,weaning,ineffective_cough,no).

premise(535,1,weaning,[hypoxia,hypercapnia]):-
equals(535,1,weaning,hypoxia,no),
equals(535,2,weaning,hypercapnia,no).
```

```
/* ------------------------------------------------------- */
/* ---------  negative regression conclusions ------------- */

premise(600,1,regression,[regress]):-
equals(600,1,regression,regress,no).

premise(601,1,regression,[respiratory_muscle_weakness_fatigue,
                          decreased_respiratory_drive,
                          increased_respiratory_drive]):-
equals(601,1,regression,respiratory_muscle_weakness_fatigue,no),
equals(601,2,regression,decreased_respiratory_drive,no),
equals(601,3,regression,increased_respirarory_drive,no).

premise(602,1,regression,[rr_gt_25_and_tv_vlow,
                          abnormal_breathing]):-
equals(602,1,regression,rr_gt_25_and_tv_vlow,no),
equals(602,2,regression,abnormal_breathing,no).

premise(603,1,regression,[abdominal_paradox,
                          respiratory_alternans]):-
equals(603,1,regression,abdominal_paradox,no),
equals(603,2,regression,respiratory_alternans,no).

premise(604,1,regression,[increased_tv_inspiratory_time_ratio]):-
equals(604,1,regression,increased_tv_inspiratory_time_ratio,no).

premise(605,1,regression,[p01_gt_6,
                          increased_co2_production]):-
equals(605,1,regression,p01_gt_6,no),
equals(605,2,regression,increased_co2_production,no).




/* ---------- negative rule premises - progression -------- */

premise(200,1,progression,[progress,regress,review]):-
equals(200,1,progression,progress,no),
equals(200,2,progression,regress,no),
equals(200,3,progression,review,no).

premise(201,1,progression,[ph_same_paco2_up,
                           ph_same_paco2_same,
                           ph_same_paco2_down]):-
equals(201,1,progression,ph_same_paco2_up,no),
equals(201,2,progression,ph_same_paco2_same,no),
equals(201,3,progression,ph_same_paco2_down,no).

premise(202,1,progression,[ph_up_paco2_up,
                           ph_down_paco2_up]):-
equals(202,1,progression,ph_up_paco2_up,no),
equals(202,2,progression,ph_down_paco2_up,no).
```

```
premise(203,1,progression,[ph_up_paco2_same,
                           ph_up_paco2_down,
                           ph_down_paco2_same,
                           ph_down_paco2_down]):-
equals(203,1,progression,ph_up_paco2_same,no),
equals(203,2,progression,ph_up_paco2_down,no),
equals(203,3,progression,ph_down_paco2_same,no),
equals(203,4,progression,ph_down_paco2_down,no).


premise(204,1,progression,[ph_same1,paco2_up1]):-
equals(204,1,progression,ph_same1,no),
equals(204,2,progression,paco2_up1,no).

premise(205,1,progression,[ph_same2,paco2_same1]):-
equals(205,1,progression,ph_same2,no),
equals(205,2,progression,paco2_same1,no).

premise(206,1,progression,[ph_same3,paco2_down1]):-
equals(206,1,progression,ph_same3,no),
equals(206,2,progression,paco2_down1,no).

premise(207,1,progression,[ph_up1,paco2_up2]):-
equals(207,1,progression,ph_up1,no),
equals(207,2,progression,paco2_up2,no).

premise(208,1,progression,[ph_down1,paco2_up3]):-
equals(208,1,progression,ph_down1,no),
equals(208,2,progression,paco2_up3,no).

premise(209,1,progression,[ph_up2,paco2_same2]):-
equals(209,1,progression,ph_up2,no),
equals(209,2,progression,paco2_same2,no).

premise(210,1,progression,[ph_up3,paco2_down2]):-
equals(210,1,progression,ph_up3,no),
equals(210,2,progression,paco2_down2,no).

premise(211,1,progression,[ph_down2,paco2_same3]):-
equals(211,1,progression,ph_down2,no),
equals(211,2,progression,paco2_same3,no).

premise(212,1,progression,[ph_down3,paco2_down3]):-
equals(212,1,progression,ph_down3,no),
equals(212,2,progression,paco2_down3,no).

premise(213,1,progression,[ph_is_same]):-
equals(213,1,progression,ph_is_same,no).

premise(214,1,progression,[paco2_is_up]):-
equals(214,1,progression,paco2_is_up,no).

premise(215,1,progression,[ph_is_same]):-
equals(215,1,progression,ph_is_same,no).

premise(216,1,progression,[paco2_is_same]):-
equals(216,1,progression,paco2_is_same,no).

premise(217,1,progression,[ph_is_same]):-
equals(217,1,progression,ph_is_same,no).
```

```
premise(218,1,progression,[paco2_is_down]):-
equals(218,1,progression,paco2_is_down,no).

premise(219,1,progression,[ph_is_up]):-
equals(219,1,progression,ph_is_up,no).

premise(220,1,progression,[paco2_is_up]):-
equals(220,1,progression,paco2_is_up,no).

premise(221,1,progression,[ph_is_down]):-
equals(221,1,progression,ph_is_down,no).

premise(222,1,progression,[paco2_is_up]):-
equals(222,1,progression,paco2_is_up,no).

premise(223,1,progression,[ph_is_up]):-
equals(223,1,progression,ph_is_up,no).

premise(224,1,progression,[paco2_is_same]):-
equals(224,1,progression,paco2_is_same,no).

premise(225,1,progression,[ph_is_up]):-
equals(225,1,progression,ph_is_up,no).

premise(226,1,progression,[paco2_is_down]):-
equals(226,1,progression,paco2_is_down,no).

premise(227,1,progression,[ph_is_down]):-
equals(227,1,progression,ph_is_down,no).

premise(228,1,progression,[paco2_is_same]):-
equals(228,1,progression,paco2_is_same,no).

premise(229,1,progression,[ph_is_down]):-
equals(229,1,progression,ph_is_down,no).

premise(230,1,progression,[paco2_is_down]):-
equals(230,1,progression,paco2_is_down,no).
```

```
/* ------------- Rule actions --------------- */
/*
   Form of rule action
   1st argument - Rule_no
   2nd argument - Premise_no
   3rd argument - Concluded variables
                                          */
action(goal,P,[state]):-
conclude(goal,P,state,intransient,"CMV").

action(terminate,P,[give_advice]):-
give_advice.


/* --------- Rule actions - weaning --------- */

action(0,P,[state,give_advice]):-
conclude(0,P,state,intransient,"SIMV"),
give_advice.

action(1,P,[fit_to_wean]):-
conclude(1,P,fit_to_wean,transient,no).

action(2,P,[respiratory_muscle_fatigue]):-
conclude(2,P,respiratory_muscle_fatigue,transient,yes).

action(3,P,[excessive_muscle_demands]):-
conclude(3,P,excessive_muscle_demands,transient,yes).

action(4,P,[impaired_energy_supply]):-
conclude(4,P,impaired_energy_supply,transient,yes).

action(5,P,[increased_respiratory_resistance]):-
conclude(5,P,increased_respiratory_resistance,transient,yes).

action(6,P,[fever_infection]):-
conclude(6,P,fever_infection,transient,yes).

action(7,P,[nutritional_deficiency]):-
conclude(7,P,nutritional_deficiency,transient,yes).

action(8,P,[breathing_circuit_inappropriate]):-
conclude(8,P,breathing_circuit_inappropriate,transient,yes).

action(9,P,[bronchospasm]):-
conclude(9,P,bronchospasm,transient,yes).

action(10,P,[f_e_imbalance]):-
conclude(10,P,f_e_imbalance,transient,yes).

action(11,P,[hypovolaemia]):-
conclude(11,P,hypovolaemia,transient,yes).

action(12,P,[anaemia]):-
conclude(12,P,anaemia,transient,yes).

action(13,P,[pulmonary_oedema]):-
conclude(13,P,pulmonary_oedema,transient,yes).
```

```
action(14,P,[potassium_level_out]):-
conclude(14,P,potassium_level_out,transient,yes).

action(15,P,[phosphate_level]):-
conclude(15,P,phosphate_level,transient,yes).

action(16,P,[system_failure]):-
conclude(16,P,system_failure,transient,yes).

action(17,P,[respiratory_failure]):-
conclude(17,P,respiratory_failure,transient,yes).

action(18,P,[ventilatory_failure]):-
conclude(18,P,ventilatory_failure,transient,yes).

action(19,P,[inefficient_pulmonary_gas_exchange]):-
conclude(19,P,inefficient_pulmonary_gas_exchange,transient,yes).

action(20,P,[cardiac_failure]):-
conclude(20,P,cardiac_failure,transient,yes).

action(21,P,[cardiac_output_low]):-
conclude(21,P,cardiac_output_low,transient,yes).

action(22,P,[acute_lvf]):-
conclude(22,P,acute_lvf,transient,yes).

action(23,P,[metabolic_acid_base]):-
conclude(23,P,metabolic_acid_base,transient,yes).

action(24,P,[renal_problem]):-
conclude(24,P,renal_problem,transient,yes).

action(25,P,[hepatic_problem]):-
conclude(25,P,hepatic_problem,transient,yes).

action(26,P,[anxiety]):-
conclude(26,P,anxiety,transient,yes).

action(27,P,[sleep_deprivation]):-
conclude(27,P,sleep_deprivation,transient,yes).

action(28,P,[primary_anxiety]):-
conclude(28,P,primary_anxiety,transient,yes).

action(29,P,[pain]):-
conclude(29,P,pain,transient,yes).

action(32,P,[sedation]):-
conclude(32,P,sedation,transient,yes).

action(33,P,[feeding_problems]):-
conclude(33,P,feeding_problems,transient,yes).

action(34,P,[atelactasis]):-
conclude(34,P,atelactasis,transient,yes).

action(35,P,[decreased_ventilatory_response]):-
conclude(35,P,descreased_ventilatory_response,transient,yes).
```

```
/* --------- Rule actions - regression ---------- */

action(100,P,[state,give_advice]):-
conclude(100,P,state,intransient,"SIMV"),
give_advice.

action(101,P,[regress]):-
conclude(101,P,regress,transient,yes).

action(102,P,[respiratory_muscle_weakness_fatigue]):-
conclude(102,P,respiratory_muscle_weakness_fatigue,transient,yes).

action(103,P,[abnormal_breathing]):-
conclude(103,P,abnormal_breathing,yes).

action(104,P,[decreased_respiratory_drive]):-
conclude(104,P,decreased_respiratory_drive,transient,yes).

action(105,P,[increased_respiratory_drive]):-
conclude(105,P,increased_respiratory_drive,transient,yes).



/* --------- Rule actions - progression ---------- */

action(200,P,[state,give_advice]):-
conclude(200,P,state,intransient,"SIMV"),
give_advice.

action(201,P,[progress]):-
conclude(201,P,progress,transient,yes).

action(202,P,[regress]):-
conclude(202,P,regress,transient,yes).

action(203,P,[review]):-
conclude(203,P,review,transient,yes).

action(204,P,[ph_same_paco2_up]):-
conclude(204,P,ph_same_paco2_up,transient,yes).

action(205,P,[ph_same_paco2_same]):-
conclude(205,P,ph_same_paco2_same,transient,yes).

action(206,P,[ph_same_paco2_down]):-
conclude(206,P,ph_same_paco2_down,transient,yes).

action(207,P,[ph_up_paco2_up]):-
conclude(207,P,ph_up_paco2_up,transient,yes).

action(208,P,[ph_down_paco2_up]):-
conclude(208,P,ph_down_paco2_up,transient,yes).

action(209,P,[ph_up_paco2_same]):-
conclude(209,P,ph_up_paco2_same,transient,yes).

action(210,P,[ph_up_paco2_down]):-
conclude(210,P,ph_up_paco2_down,transient,yes).
```

```
action(211,P,[ph_down_paco2_same]):-
conclude(211,P,ph_down_paco2_same,transient,yes).

action(212,P,[ph_down_paco2_down]):-
conclude(212,P,ph_down_paco2_down,transient,yes).

action(213,P,[ph_same1]):-
conclude(213,P,ph_same1,transient,yes).

action(214,P,[paco2_up1]):-
conclude(214,P,paco2_up1,transient,yes).

action(215,P,[ph_same2]):-
conclude(215,P,ph_same2,transient,yes).

action(216,P,[paco2_same1]):-
conclude(216,P,paco2_same1,transient,yes).

action(217,P,[ph_same3]):-
conclude(217,P,ph_same3,transient,yes).

action(218,P,[paco2_down1]):-
conclude(218,P,paco2_down1,transient,yes).

action(219,P,[ph_up1]):-
conclude(219,P,ph_up1,transient,yes).

action(220,P,[paco2_up2]):-
conclude(220,P,paco2_up2,transient,yes).

action(221,P,[ph_down1]):-
conclude(221,1,ph_down1,transient,yes).

action(222,P,[paco2_up3]):-
conclude(222,P,paco2_up3,transient,yes).

action(223,P,[ph_up2]):-
conclude(223,P,ph_up2,transient,yes).

action(224,P,[paco2_same2]):-
conclude(224,P,paco2_same2,transient,yes).

action(225,P,[ph_up3]):-
conclude(225,P,ph_up3,transient,yes).

action(226,P,[paco2_down2]):-
conclude(226,P,paco2_down2,transient,yes).

action(227,P,[ph_down2]):-
conclude(227,P,ph_down2,transient,yes).

action(228,P,[paco2_same3]):-
conclude(228,P,paco2_same3,transient,yes).

action(229,P,[ph_down3]):-
conclude(229,P,ph_down3,transient,yes).

action(230,P,[paco2_down3]):-
conclude(230,P,paco2_down3,transient,yes).
```

```
/* --- negative action conclusions - weaning --- */

action(500,P,[give_advice]):-
give_advice.

action(501,P,[fit_to_wean]):-
conclude(501,P,fit_to_wean,transient,yes).

action(502,P,[respiratory_muscle_fatigue]):-
conclude(502,P,respiratory_muscle_fatigue,transient,no).

action(503,P,[excessive_muscle_demands]):-
conclude(503,P,excessive_muscle_demands,transient,no).

action(504,P,[impaired_energy_supply]):-
conclude(504,P,impaired_energy_supply,transient,no).

action(505,P,[increased_respiratory_resistance]):-
conclude(505,P,increased_respiratory_resistance,transient,no).

action(506,P,[fever_infection]):-
conclude(506,P,fever_infection,transient,no).

action(507,P,[nutritional_deficiency]):-
conclude(507,P,nutritional_deficiency,transient,no).

action(508,P,[breathing_circuit_inappropriate]):-
conclude(508,P,breathing_circuit_inappropriate,transient,no).

action(509,P,[bronchospasm]):-
conclude(509,P,bronchospasm,transient,no).

action(510,P,[f_e_imbalance]):-
conclude(510,P,f_e_imbalance,transient,no).

action(511,P,[hypovolaemia]):-
conclude(511,P,hypovolaemia,transient,no).

action(512,P,[anaemia]):-
conclude(512,P,anaemia,transient,no).

action(513,P,[pulmonary_oedema]):-
conclude(513,P,pulmonary_oedema,transient,no).

action(514,P,[potassium_level_out]):-
conclude(514,P,potassium_level_out,transient,no).

action(515,P,[phosphate_level]):-
conclude(515,P,phosphate_level,transient,no).

action(516,P,[system_failure]):-
conclude(516,P,system_failure,transient,no).

action(517,P,[respiratory_failure]):-
conclude(517,P,respiratory_failure,transient,no).

action(518,P,[ventilatory_failure]):-
conclude(518,P,ventilatory_failure,transient,no).
```

```
action(519,P,[inefficient_pulmonary_gas_exchange]):-
conclude(519,P,inefficient_pulmonary_gas_exchange,transient,no).

action(520,P,[cardiac_failure]):-
conclude(520,P,cardiac_failure,transient,no).

action(521,P,[cardiac_output_low]):-
conclude(521,P,cardiac_output_low,transient,no).

action(522,P,[acute_lvf]):-
conclude(522,P,acute_lvf,transient,no).

action(523,P,[metabolic_acid_base]):-
conclude(523,P,metabolic_acid_base,transient,no).

action(524,P,[renal_problem]):-
conclude(524,P,renal_problem,transient,no).

action(525,P,[hepatic_problem]):-
conclude(525,P,hepatic_problem,transient,no).

action(526,P,[anxiety]):-
conclude(526,P,anxiety,transient,no).

action(527,P,[sleep_deprivation]):-
conclude(527,P,sleep_deprivation,transient,no).

action(528,P,[primary_anxiety]):-
conclude(528,P,primary_anxiety,transient,no).

action(529,P,[pain]):-
conclude(529,P,pain,transient,no).

action(532,P,[sedation]):-
conclude(532,P,sedation,transient,no).

action(533,P,[feeding_problems]):-
conclude(533,P,feeding_problems,transient,no).

action(534,P,[atelactasis]):-
conclude(534,P,atelactasis,transient,no).

action(535,P,[decreased_ventilatory_response]):-
conclude(535,P,descreased_ventilatory_response,transient,no).
```

```
/* -- negative action conclusions - regression -- */

action(600,P,[give_advice]):-
give_advice.

action(601,P,[regress]):-
conclude(601,P,regress,transient,no).

action(602,P,[respiratory_muscle_weakness_fatigue]):-
conclude(602,P,respiratory_muscle_weakness_fatigue,transient,no).

action(603,P,[abnormal_breathing]):-
conclude(603,P,abnormal_breathing,no).

action(604,P,[decreased_respiratory_drive]):-
conclude(604,P,decreased_respiratory_drive,transient,no).

action(605,P,[increased_respiratory_drive]):-
conclude(605,P,increased_respiratory_drive,transient,no).



/* -- Negative action conclusions - progression -- */

action(700,P,[state,give_advice]):-
conclude(700,P,state,intransient,"SIMV"),
give_advice.

action(701,P,[progress]):-
conclude(701,P,progress,transient,yes).

action(702,P,[regress]):-
conclude(702,P,regress,transient,yes).

action(703,P,[review]):-
conclude(703,P,review,transient,yes).

action(704,P,[ph_same_paco2_up]):-
conclude(704,P,ph_same_paco2_up,transient,yes).

action(705,P,[ph_same_paco2_same]):-
conclude(705,P,ph_same_paco2_same,transient,yes).

action(706,P,[ph_same_paco2_down]):-
conclude(706,P,ph_same_paco2_down,transient,yes).

action(707,P,[ph_up_paco2_up]):-
conclude(707,P,ph_up_paco2_up,transient,yes).

action(708,P,[ph_down_paco2_up]):-
conclude(708,P,ph_down_paco2_up,transient,yes).

action(709,P,[ph_up_paco2_same]):-
conclude(709,P,ph_up_paco2_same,transient,yes).


action(710,P,[ph_up_paco2_down]):-
conclude(710,P,ph_up_paco2_down,transient,yes).

action(711,P,[ph_down_paco2_same]):-
conclude(711,P,ph_down_paco2_same,transient,yes).
```

```
action(712,P,[ph_down_paco2_down]):-
conclude(712,P,ph_down_paco2_down,transient,yes).

action(713,P,[ph_same1]):-
conclude(713,P,ph_same1,transient,yes).

action(714,P,[paco2_up1]):-
conclude(714,P,paco2_up1,transient,yes).

action(715,P,[ph_same2]):-
conclude(715,P,ph_same2,transient,yes).

action(716,P,[paco2_same1]):-
conclude(716,P,paco2_same1,transient,yes).

action(717,P,[ph_same3]):-
conclude(717,P,ph_same3,transient,yes).

action(718,P,[paco2_down1]):-
conclude(718,P,paco2_down1,transient,yes).

action(719,P,[ph_up1]):-
conclude(719,P,ph_up1,transient,yes).

action(720,P,[paco2_up2]):-
conclude(720,P,paco2_up2,transient,yes).

action(721,P,[ph_down1]):-
conclude(721,P,ph_down1,transient,yes).

action(722,P,[paco2_up3]):-
conclude(722,P,paco2_up3,transient,yes).

action(723,P,[ph_up2]):-
conclude(723,P,ph_up2,transient,yes).

action(724,P,[paco2_same2]):-
conclude(724,P,paco2_same2,transient,yes).

action(725,P,[ph_up3]):-
conclude(725,P,ph_up3,transient,yes).

action(726,P,[paco2_down2]):-
conclude(726,P,paco2_down2,transient,yes).

action(727,P,[ph_down2]):-
conclude(727,P,ph_down2,transient,yes).

action(728,P,[paco2_same3]):-
conclude(728,P,paco2_same3,transient,yes).

action(729,P,[ph_down3]):-
conclude(729,P,ph_down3,transient,yes).


action(730,P,[paco2_down3]):-
conclude(730,P,paco2_down3,transient,yes).
```

```
/* ------- data request rules - weaning ----------*/

premise(901,1,request,[state,increased_dynamic_resistance]):-
one_of(901,1,request,state,["CMV"]).

premise(902,1,request,[state,sensitivity]):-
one_of(902,1,request,state,["CMV"]).

premise(903,1,request,[state,obesity]):-
one_of(903,1,request,state,["CMV"]),
not_equal(903,1,request,obesity,A).

premise(904,1,request,[state,ascites]):-
one_of(904,1,request,state,["CMV"]),
not_equal(904,1,request,ascites,A).

premise(905,1,request,[state,impaired_oxygen_delivery]):-
one_of(905,1,request,state,["CMV"]).

premise(906,1,request,[state,catabolic]):-
one_of(906,1,request,state,["CMV"]).

premise(907,1,request,[state,reduced_oxygen_consumption]):-
one_of(907,1,request,state,["CMV"]).

premise(908,1,request,[state,increased_work_of_breathing]):-
one_of(908,1,request,state,["CMV"]).

premise(909,1,request,[state,increased_co2_production]):-
one_of(909,1,request,state,["CMV"]).

premise(910,1,request,[state,c_v_pressure]):-
one_of(910,1,request,state,["CMV"]).

premise(911,1,request,[state,haemoglobin_count]):-
one_of(911,1,request,state,["CMV"]).

premise(912,1,request,[state,p_a_wedge_pressure_high]):-
one_of(912,1,request,state,["CMV"]).

premise(913,1,request,[state,colloid_osmotic_pressure]):-
one_of(913,1,request,state,["CMV"]).

premise(914,1,request,[state,potassium_high]):-
one_of(914,1,request,state,["CMV"]).

premise(915,1,request,[state,potassium_low]):-
one_of(915,1,request,state,["CMV"]).

premise(916,1,request,[state,phosphate_high]):-
one_of(916,1,request,state,["CMV"]).

premise(917,1,request,[state,tidal_volume]):-
one_of(917,1,request,state,["CMV"]).

premise(918,1,request,[state,capAaDO2]):-
one_of(918,1,request,state,["CMV"]).

premise(919,1,request,[state,cardiac_index]):-
one_of(919,1,request,state,["CMV"]).
```

```
premise(920,1,request,[state,p_a_wedge_pressure_low]):-
one_of(920,1,request,state,["CMV"]).

premise(921,1,request,[state,systolic_blood_pressure]):-
one_of(921,1,request,state,["CMV"]).

premise(922,1,request,[state,metabolic_alkalosis]):-
one_of(922,1,request,state,["CMV"]).

premise(923,1,request,[state,respiratory_alkalosis]):-
one_of(923,1,request,state,["CMV"]).

premise(924,1,request,[state,creatinine_high]):-
one_of(924,1,request,state,["CMV"]).

premise(925,1,request,[state,lft_deranged]):-
one_of(925,1,request,state,["CMV"]).

premise(926,1,request,[state,ineffective_cough]):-
one_of(926,1,request,state,["CMV"]).

premise(927,1,request,[state,hypoxia]):-
one_of(927,1,request,state,["CMV"]).

premise(928,1,request,[state,hypercapnia]):-
one_of(928,1,request,state,["CMV"]).

premise(929,1,request,[state,neurological]):-
one_of(929,1,request,state,["CMV"]).

premise(930,1,request,[state,sleeping_draught]):-
one_of(930,1,request,state,["CMV"]).

premise(931,1,request,[state,anxiolytic_agent]):-
one_of(931,1,request,state,["CMV"]).

premise(932,1,request,[state,local_blocker]):-
one_of(932,1,request,state,["CMV"]).

premise(933,1,request,[state,infiltration]):-
one_of(933,1,request,state,["CMV"]).

premise(934,1,request,[state,sedative]):-
one_of(934,1,request,state,["CMV"]).
```

```
/* ------ data request rules - regression ------*/

premise(935,1,request,[state,rr_gt_25_and_tv_vlow]):-
one_of(935,1,request,state,["SIMV"]).

premise(936,1,request,[state,abdominal_paradox]):-
one_of(936,1,request,state,["SIMV"]).

premise(937,1,request,[state,respiratory_alternans]):-
one_of(937,1,request,state,["SIMV"]).

premise(938,1,request,[state,increased_tv_inspiratory_time_ratio]):-
one_of(938,1,request,state,["SIMV"]).

premise(939,1,request,[state,p01_gt_6]):-
one_of(939,1,request,state,["SIMV"]).

premise(940,1,request,[state,increased_co2_production]):-
one_of(940,1,request,state,["SIMV"]).




/* ------ data request rules - progression ------*/

premise(941,1,request,[state,ph_is_same]):-
one_of(941,1,request,state,["SIMV"]).

premise(942,1,request,[state,paco2_is_up]):-
one_of(942,1,request,state,["SIMV"]).

premise(943,1,request,[state,ph_is_up]):-
one_of(943,1,request,state,["SIMV"]).

premise(944,1,request,[state,paco2_is_same]):-
one_of(944,1,request,state,["SIMV"]).

premise(945,1,request,[state,ph_is_down]):-
one_of(945,1,request,state,["SIMV"]).

premise(946,1,request,[state,paco2_is_down]):-
one_of(946,1,request,state,["SIMV"]).
```

```
/* ******************************************************  */
/* --------------- weaning action queries ----------------- */

action(901,1,[increased_dynamic_resistance]):-
request("Has dynamic resistance increased significantly ? ",
                                                    yes-no,R),
conclude(901,1,increased_dynamic_resistance,transient,R).

action(902,1,[sensitivity]):-
request("Is sensitivity at a minimum ? ",yes-no,R),
conclude(902,1,sensitivity,transient,R).

action(903,1,[obesity]):-
request("Is the patient obese ? ",yes-no,R),
conclude(903,1,obesity,intransient,R).

action(904,1,[ascites]):-
request("Does the patient have ascites ? ",yes-no,R),
conclude(904,1,ascites,intransient,R).

action(905,1,[impaired_oxygen_delivery]):-
request("Is oxygen delivery impaired ? ",yes-no,R),
conclude(905,1,impaired_oxygen_delivery,transient,R).

action(906,1,[catabolic]):-
request("Is the patient catabolic ? ",yes-no,R),
conclude(906,1,catabolic,transient,R).

action(907,1,[reduced_oxygen_consumption]):-
request("Is there a marked decrease in oxygen consumption ? ",
                                                    yes-no,R),
conclude(907,1,reduced_oxygen_consumption,transient,R).

action(908,1,[increased_work_of_breathing]):-
request("Is there a marked increase in work of breathing ? ",
                                                    yes-no,R),
conclude(908,1,increased_work_of_breathing,transient,R).

action(909,1,[increased_co2_production]):-
request("Is there an increase in CO2 production ? ",yes-no,R),
conclude(909,1,increased_co2_production,transient,R).

action(910,1,[c_v_pressure]):-
request("Is central venous pressure low ? ",yes-no,R),
conclude(910,1,c_v_pressure,transient,R).

action(911,1,[haemoglobin_count]):-
request("Is the haemoglobin count low ? ",yes-no,R),
conclude(911,1,haemoglobin_count,transient,R).

action(912,1,[p_a_wedge_pressure_high]):-
request("Is pulmonary artery wedge pressure high ? ",yes-no,R),
conclude(912,1,p_a_wedge_pressure_high,transient,R).

action(913,1,[colloid_osmotic_pressure]):-
request("Is colloid osmotic pressure low ? ",yes-no,R),
conclude(913,1,colloid_osmotic_pressure,transient,R).
```

```
action(914,1,[potassium_high]):-
request("Is potassium level high ? ",yes-no,R),
conclude(914,1,potassium_high,transient,R).

action(915,1,[potassium_low]):-
request("Is potassium level low ? ",yes-no,R),
conclude(915,1,potassium_low,transient,R).

action(916,1,[phosphate_high]):-
request("Is phosphate level high ? ",yes-no,R),
conclude(916,1,phosphate_high,transient,R).

action(917,1,[tidal_volume]):-
request("Is tidal volume low ? ",yes-no,R),
conclude(917,1,tidal_volume,transient,R).

action(918,1,[capAaDO2]):-
request("Is AaDO2 gradient low ? ",yes-no,R),
conclude(918,1,capAaDO2,transient,R).

action(919,1,[cardiac_index]):-
request("Is cardiac index low ? ",yes-no,R),
conclude(919,1,cardiac_index,transient,R).

action(920,1,[p_a_wedge_pressure_low]):-
request("Is pulmonary wedge pressure low ?",yes-no,R),
conclude(920,1,p_a_wedge_pressure_low,transient,R).

action(921,1,[systolic_blood_pressure]):-
request("Is systolic blood pressure low ? ",yes-no,R),
conclude(921,1,systolic_blood_pressure,transient,R).

action(922,1,[metabolic_alkalosis]):-
request("Is the patient metabolic alkalotic ? ",yes-no,R),
conclude(922,1,metabolic_alkalosis,transient,R).

action(923,1,[respiratory_alkalosis]):-
request("Is the patient respiratory alkalotic ? ",yes-no,R),
conclude(923,1,respiratory_alkalosis,transient,R).

action(924,1,[creatinine_high]):-
request("Is there an increase in creatinine ? ",yes-no,R),
conclude(924,1,creatinine_high,transient,R).

action(925,1,[lft_deranged]):-
request("Are the liver function tests deranged ? ",yes-no,R),
conclude(925,1,lft_deranged,transient,R).

action(926,1,[ineffective_cough]):-
request("Has the patient got an ineffective cough ? ",yes-no,R),
conclude(926,1,ineffective_cough,transient,R).

action(927,1,[hypoxia]):-
request("Is the patient hypoxic ? ",yes-no,R),
conclude(927,1,hypoxia,transient,R).

action(928,1,[hypercapnia]):-
request("Is the patient hypercapnic ? ",yes-no,R),
conclude(928,1,hypercapnia,transient,R).
```

```
action(929,1,[neurological]):-
request("Is there a neurological involvement ? ",yes-no,R),
conclude(929,1,neurological,transient,R).

action(930,1,[sleeping_draught]):-
request("Is the patient taking a sleeping draught ? ",yes-no,R),
conclude(930,1,sleeping_draught,transient,R).

action(931,1,[anxiolytic_agent]):-
request("Is the patient taking an anxiolytic agent ? ",yes-no,R),
conclude(931,1,anxiolytic_agent,transient,R).

action(932,1,[local_blocker]):-
request("Is the patient receiving a local pain blocker ? ",
                                                    yes-no,R),
conclude(932,1,local_blocker,transient,R).

action(933,1,[infiltration]):-
request
  ("Is the patient receiving an infiltration to relieve pain ? ",
                                                    yes-no,R),
conclude(933,1,infiltration,transient,R).


action(934,1,[sedative]):-
request("Is the patient taking sedatives ? ",yes-no,R),
conclude(934,1,sedative,transient,R).




/* ------------ regression action queries ------------------ */

action(935,1,[rr_gt_25_and_tv_vlow]):-
request("Is RR >25, and TV very low ? ",yes-no,R),
conclude(935,1,rr_gt_25_and_tv_vlow,transient,R).

action(936,1,[abdominal_paradox]):-
request("Is breathing pattern abdominal paradox ? ",yes-no,R),
conclude(936,1,abdominal_paradox,transient,R).

action(937,1,[respiratory_alternans]):-
request("Is breathing pattern respiratory alternans ? ",
                                                    yes-no,R),
conclude(937,1,respiratory_alternans,transient,R).

action(938,1,[increased_tv_inspiratory_time_ratio]):-
request("Has the patient an increased tv-insp time ratio ? ",
                                                    yes-no,R),
conclude(938,1,increased_tv_inspiratory_time_ratio,transient,R).

action(939,1,[p01_gt_6]):-
request("Is the airway occlusion pressure P0.1 >6 ? ",yes-no,R),
conclude(939,1,p01_gt_6,transient,R).

action(940,1,[increased_co2_production]):-
request("Is there an increase in CO2 production ? ",yes-no,R),
conclude(940,1,increased_co2_production,transient,R).
```

```
/* ------------ progression action queries ------------------ */

action(941,1,[ph_is_same]):-
request("Is pH about the same ? ",yes-no,R),
conclude(941,1,ph_is_same,transient,R).

action(942,1,[paco2_is_up]):-
request("Has PaCO2 value gone up significantly ? ",yes-no,R),
conclude(942,1,paco2_is_up,transient,R).

action(943,1,[ph_is_up]):-
request("Has pH value gone up significantly ? ",yes-no,R),
conclude(943,1,ph_is_up,transient,R).

action(944,1,[paco2_is_same]):-
request("Is PaCO2 value about the same ? ",yes-no,R),
conclude(944,1,paco2_is_same,transient,R).

action(945,1,[ph_is_down]):-
request("Has the pH value gone down significantly ? ",yes-no,R),
conclude(945,1,ph_is_down,transient,R).

action(946,1,[paco2_is_down]):-
request("Has the PaCO2 value gone down significantly ? ",yes-no,R),
conclude(946,1,paco2_is_down,transient,R).




/* ---------------------- Inference Engine --------------------- */

/* First pass activation - */
/* find all rules with all data in premise known */

first_pass_activation(Rule_list):-
setof((Rule_no,Premise_no),Rule_type^premise_requirements(Rule_no,
            Premise_no,Rule_type,[]),Rule_list),!.

first_pass_activation([]):-!.


/*Second pass activation - */
/*evaluate premise of rules that survived 1st pass*/

second_pass_activation([],[]):-!.

second_pass_activation([(Rule_no,Premise_no)¦Rule_list],
                    [(Rule_no,Premise_no,Type)¦Activated_rules]):-
premise(Rule_no,Premise_no,Type,Dependents),
second_pass_activation(Rule_list,Activated_rules).

second_pass_activation([(Rule_no,Premise_no)¦Rule_list],
                                                Activated_rules):-
second_pass_activation(Rule_list,Activated_rules).
```

```
/* System control cycle */

control_cycle:-                                  /* forward chain */
once(first_pass_activation(Rule_list)),
once(second_pass_activation(Rule_list,Activated_rules)),
once(schedule_rules(Rule_no,Premise_no,Activated_rules)),
action(Rule_no,Premise_no,Conclusions),
control_cycle.

control_cycle.

/* kick off system */


run_wean:-
generate_premise_requirements,
retractall(current_value/3),
control_cycle.



/* ------------------- Meta rules -------------------- */

schedule_rules(Rule_no,Premise_no,Activated_rules):-
member((Rule_no,Premise_no,goal),Activated_rules).

schedule_rules(Rule_no,Premise_no,Activated_rules):-
member((Rule_no,Premise_no,weaning),Activated_rules).

schedule_rules(Rule_no,Premise_no,Activated_rules):-
member((Rule_no,Premise_no,regression),Activated_rules).

schedule_rules(Rule_no,Premise_no,Activated_rules):-
member((Rule_no,Premise_no,progression),Activated_rules).

schedule_rules(Rule_no,Premise_no,Activated_rules):-
member((Rule_no,Premise_no,request),Activated_rules).

/* ask a question if no rules fire */

schedule_rules(Rule_no,Premise_no,[(terminate,terminate)]):-
premise_requirements(Rule_no,Premise_no,request,A),
clause(action/3,action(Rule_no,Premise_no,A):-G).

schedule_rules(terminate,1,[(terminate,terminate)]).
```

```
/* ---------------------  Operators  ------------------------ */


equals(Rule_no,Premise_no,Type,A,B):-
current_value(A,T,B).

equals(Rule_no,Premise_no,Type,A,B):-
increase_premise_requirements(Rule_no,Premise_no,Type,A),!,
fail.



not_equal(Rule_no,Premise_no,Type,A,B):-
not current_value(A,T,B).

not_equal(Rule_no,Premise_no,Type,A,B):-
increase_premise_requirements(Rule_no,Premise_no,Type,A),!,
fail.

greater_than_or_equal(Rule_no,Premise_no,Type,A,B):-
current_value(A,T,V),
A>=B.

greater_than_or_equal(Rule_no,Premise_no,Type,A,B):-
increase_premise_requirements(Rule_no,Premise_no,Type,A),!,
fail.

less_than_or_equal(Rule_no,Premise_no,Type,A,B):-
current_value(A,T,V),
A=<B.

less_than_or_equal(Rule_no,Premise_no,Type,A,B):-
increase_premise_requirements(Rule_no,Premise_no,Type,A),!,
fail.

one_of(Rule_no,Premise_no,Type,A,B):-
current_value(A,T,V),
member(V,B).

one_of(Rule_no,Premise_no,Type,A,B):-
retract(premise_requirements(Rule_no,Premise_no,Type,[])),
assert(premise_requirements(Rule_no,Premise_no,Type,[A])),!,
fail.
```

```
/* -------------------------- Actions -------------------- */

conclude(Rule_no,Premise_no,A,Observation_type,B):-
predicate_size(premise_requirements/4,N),
reduce_premise_requirements(A,N),
clause
(premise/4,(premise(Rule_no,Premise_no,Type,Dependents):-G),_),
increase_premise_requirements
                    (Rule_no,Premise_no,Type,Dependents),!,
retractall(current_value(A,_,_)),
assert(current_value(A,Observation_type,B)).

convert_response("yes",yes).
convert_response("no",no).

request(Prompt,yes-no,Response):-
fedit(5,1,75,"Data Request",Prompt,"",white on blue,R),
convert_response(R,Response).



give_advice:-
write("That's all folks!"),nl,
listing(current_value),!,
fail.

/* fails so that control cycle ends after advice is given */



/* -------- managing premise requirements ------ */

/* generate initial premise requirements */

generate_premise_requirements:-
once(retractall(premise_requirements/4)),
clause
 (premise/4,(premise(Rule_no,Premise_no,Type,Dependents):-G),_),
assert(premise_requirements(Rule_no,Premise_no,Type,Dependents)),
fail.

generate_premise_requirements.


/* reduce premise requirements for all rules */

reduce_premise_requirements(A,1).

reduce_premise_requirements(A,N):-
retract(premise_requirements/4,premise_requirements(Rule_no,
            Premise_no,Rule_type,Requirements),N),
compound_delete(A,Requirements,New_requirements),
assertz(premise_requirements(Rule_no,
            Premise_no,Rule_type,New_requirements)),
Next_N is N-1,
reduce_premise_requirements(A,Next_N).
```

```
/* increase premise requirements for specified rule */

increase_premise_requirements(Rule_no,Premise_no,Rule_type,A):-
once(bagof(Requirements,
            Premise_no^premise_requirements(Rule_no,
            Premise_no,Rule_type,Requirements),Premise_list)),
member(Requirements,Premise_list),
once(retract(premise_requirements(Rule_no,
            Premise_no,Rule_type,Requirements))),
once(assert(premise_requirements(Rule_no,
            Premise_no,Rule_type,[A|Requirements]))), fail.

increase_premise_requirements(Rule_no,Premise_no,Rule_type,A).



/* ---------------------- utilities ------------------------- */

/* delete element from list */

delete(A,[],[]).

delete(A,[A|L],L).

delete(A,[B|L],[B|L1]):-
delete(A,L,L1).


/* delete element from list if it contains the target */

compound_delete(A,[],[]).

compound_delete(A,[E|L],L1):-
member(A,E),
compound_delete(A,L,L1).

compound_delete(A,[A|L],L1):-
compound_delete(A,L,L1).

compound_delete(A,[B|L],[B|L1]):-
compound_delete(A,L,L1).


/* member of a list */

member(A,[A|L]).

member(A,[B|L]):-
member(A,L).
```